ARRAY OPERATIONS

BY:
DR. SHEFALI ARORA
NIT JALANDHAR

- Traversal
- Insertion
- Deletion
- Searching
- Sorting

Traversal

- Each element in an array is visited
- Traversal proceeds from element at 0th index to element at the last index.
- 1,2,3,4,5,6,7,8,9,10
- for(int i=0;i<10;i++)
- {
-   printf(“%d”,a[i]);
- }

Program to traverse an array and give number of pos/neg elements

```
int a[10]; int neg,pos,zero;
for(int i=0;i<10;i++)
{
scanf("%d",&a[i]);
}
for(int i=0;i<10;i++)
{
if(a[i]<0)
{
neg+=1;
}
else if (a[i]>=0)
{
pos+=1;
}
else if(a[i]==0)
{zero+=1;
}
}
```

Searching

- Search for an element at a given index
- Random access data structure
- Array can be searched from the beginning to the end.
- Known as Linear search

```c
#include<stdio.h>
int main()
{
    int a[5]={2,3,1,4,2}; int pos;
    printf("Enter position of the element");
    scanf("%d",&pos);
    int num=a[pos-1];
    printf("Element is %d",num);
}
```

```c
#include<stdio.h>
int main()
{
int a[5]={1,2,3,4,5}; int num;
printf("Enter value to be searched");
scanf("%d",&num);
for(int i=0;i<5;i++)
{
    if(a[i]==num)
    {
        printf("Value found");
    }
}
return 0;
}
```

Binary Search

- **Binary Search** is a search algorithm that is used to find the position of an element (target value) in a sorted array. The array should be sorted prior to applying a binary search.
- Binary search is also known by these names, logarithmic search, binary chop, half interval search.
- **Working**
- The binary search algorithm works by comparing the element to be searched by the middle element of the array and based on this comparison follows the required procedure.
- **Case 1** − element = middle, the element is found return the index.
- **Case 2** − element > middle, search for the element in the sub-array starting from middle+1 index to n.
- **Case 3** − element < middle, search for element in the sub-array starting from 0 index to middle -1.

- 1 5 16 19 23 26 27
- Start index =4
- End index = 4
- Middle = 4
- A[4]= 23 middle
- Search : 23

```
int n = 7;
int element = 16;
int start_index=0;
int end_index=n-1;
int flag=0;
int middle;
while (start_index <= end_index){
    middle = start_index + (end_index- start_index )/2;
    if (array[middle] == element)
    {
        flag=1;
        break;
    }
    if (array[middle] < element)
        start_index = middle + 1;
    else if(array[middle]>element)
        end_index = middle - 1;

}

if(flag==1)
{
     printf("Found at %d",middle);

}
```

Deletion

- Remove element from a given position
- To delete ith element , all elements from the right have to be shifted to the left by one step.

- 1 2 3 4 5
- 1 3 4 5

```
    printf( " Define the position of the array element where you want to delete: \n ");
    scanf (" %d", &pos);

        for (i = pos - 1; i < num -1; i++)
        {
            arr[i] = arr[i+1];
        }

        printf (" \n The resultant array is: \n");

        for (i = 0; i< num - 1; i++)
        {
            printf (" arr[%d] = ", i);
            printf (" %d \n", arr[i]);
        }

    return 0;
}
```

Insertion

i=4
i>=2
A[5]=a[4]

i=3
i>=2
A[4]=a[3]

i=2
i>=2
A[3]=a[2]

A[2]=6

```
using namespace std;

int main()
{
    int a[5]={1,2,3,4,5};
    printf("Enter position and element");
    int pos,ele;
    int n=5;
    scanf("%d %d",&pos,&ele);
    for(int i=n-1;i>=pos;i--)
    {
        a[i+1]=a[i];
    }
    a[pos]=ele;

    for(int i=0;i<6;i++)
    {
        printf("%d\n",a[i]);
    }
    n=n+1;
    return 0;
}
```

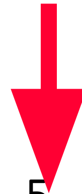
- A[5]={1,2,3,4,5}
- 1,2,6,3,4,5
- Insertion

- 2, 6
- 4

Sorting

- **Sorting takes an unordered collection and makes it an ordered one.**

1	2	3	4	5	6
77	42	35	12	101	5



1	2	3	4	5	6
5	12	35	42	77	101

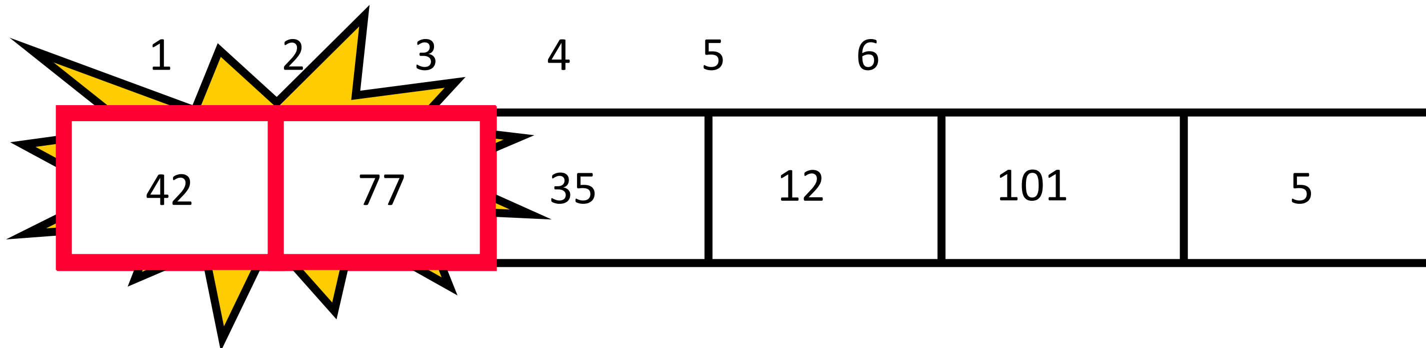
"Bubbling Up" the Largest Element

- **Traverse a collection of elements**
  - **Move from the front to the end**
  - **"Bubble" the largest value to the end using pair-wise comparisons and swapping**

1	2	3	4	5	6
77	42	35	12	101	5

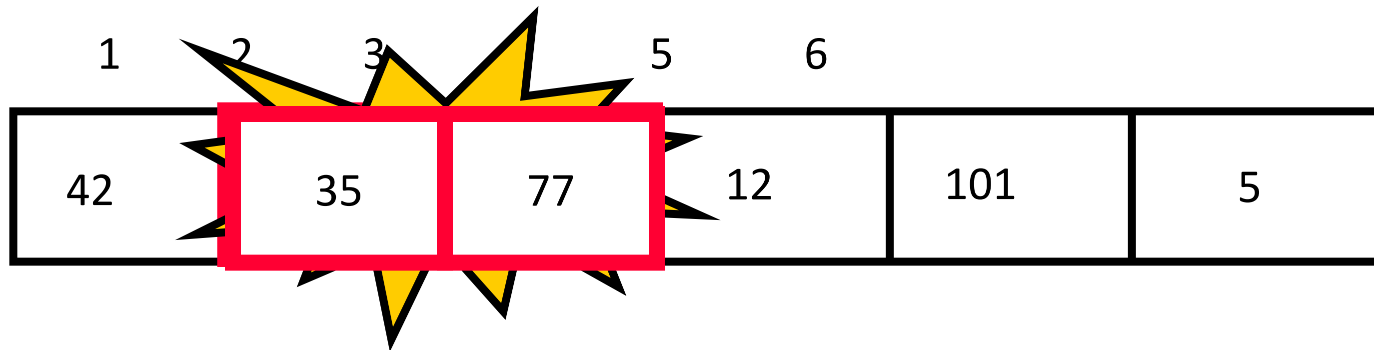
"Bubbling Up" the Largest Element

- **Traverse a collection of elements**
  - **Move from the front to the end**
  - **“Bubble” the largest value to the end using pair-wise comparisons and swapping**



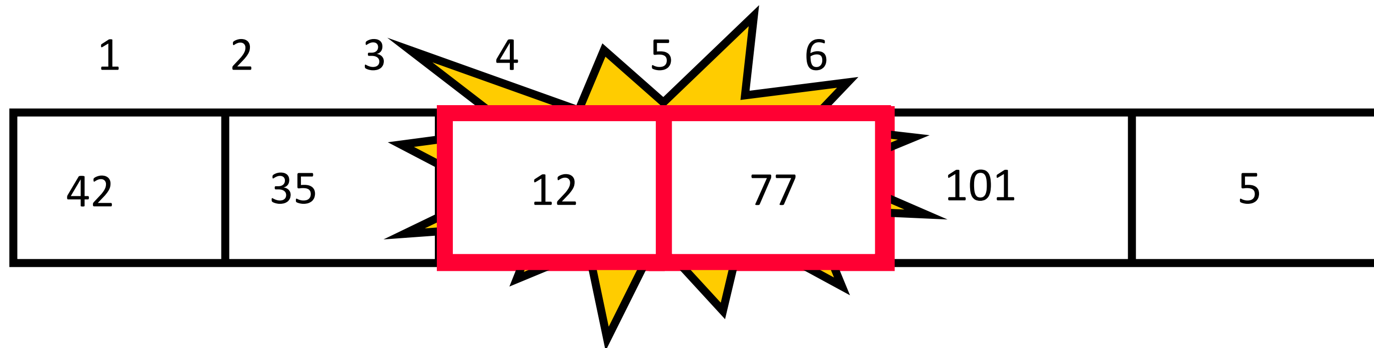
"Bubbling Up" the Largest Element

- **Traverse a collection of elements**
  - **Move from the front to the end**
  - **"Bubble" the largest value to the end using pair-wise comparisons and swapping**



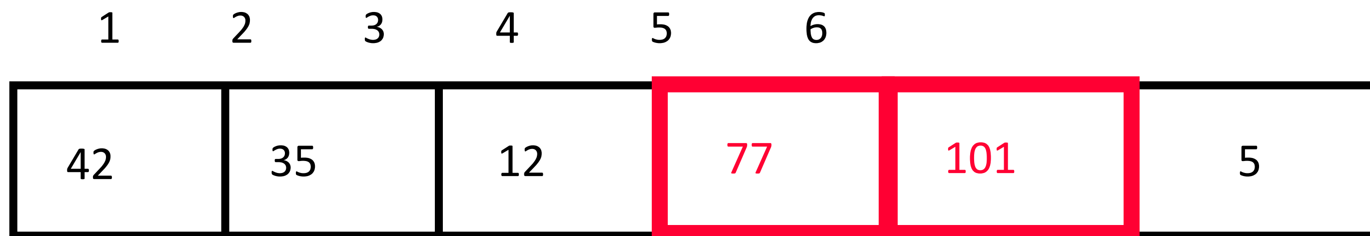
"Bubbling Up" the Largest Element

- **Traverse a collection of elements**
  - **Move from the front to the end**
  - **“Bubble” the largest value to the end using pair-wise comparisons and swapping**



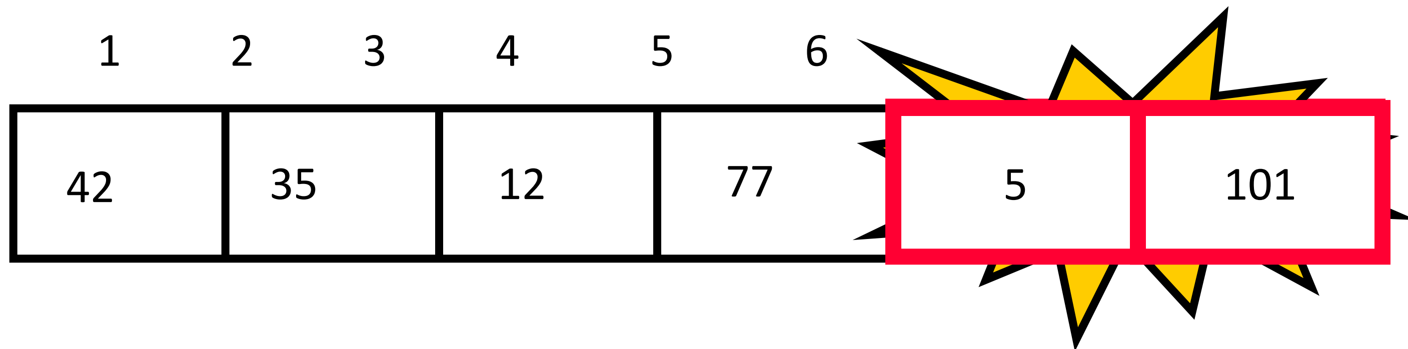
"Bubbling Up" the Largest Element

- **Traverse a collection of elements**
  - **Move from the front to the end**
  - **"Bubble" the largest value to the end using pair-wise comparisons and swapping**



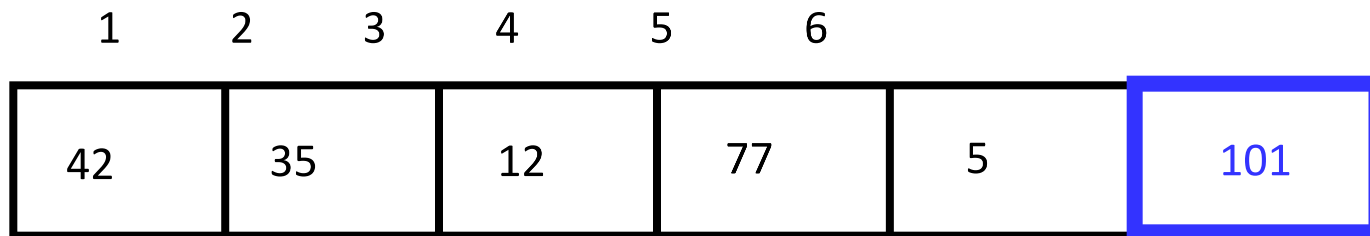
"Bubbling Up" the Largest Element

- **Traverse a collection of elements**
  - **Move from the front to the end**
  - **“Bubble” the largest value to the end using pair-wise comparisons and swapping**



"Bubbling Up" the Largest Element

- **Traverse a collection of elements**
  - **Move from the front to the end**
  - **“Bubble” the largest value to the end using pair-wise comparisons and swapping**



The “Bubble Up” Algorithm

```
index <- 1
last_compare_at <- n – 1

loop
  exitif(index > last_compare_at)
  if(A[index] > A[index + 1]) then
    Swap(A[index], A[index + 1])
  endif
  index <- index + 1
endloop
```

LB

No, Swap isn't built in.

```
Procedure Swap(a, b isoftype
  in/out Num)
   t isoftype Num
   t <- a
   a <- b
   b <- t
endprocedure // Swap
```

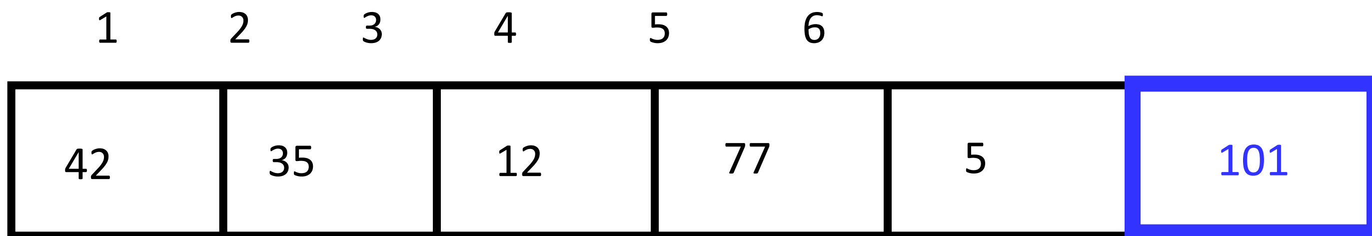
Items of Interest

- **Notice that only the largest value is correctly placed**
- **All other values are still out of order**
- **So we need to repeat this process**

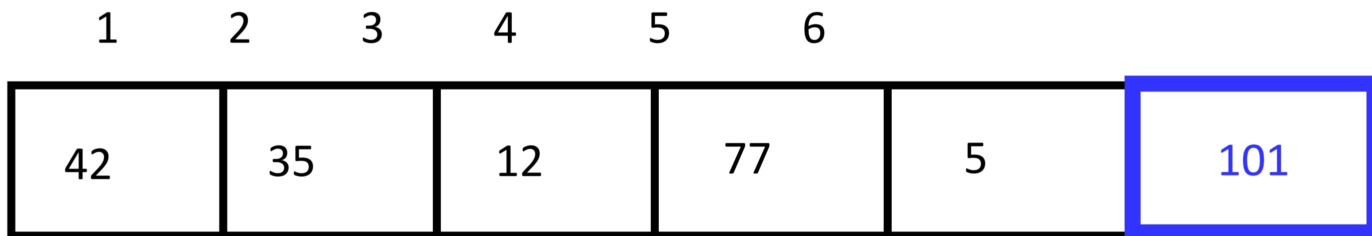
1	2	3	4	5	6
42	35	12	77	5	101

Largest value correctly placed

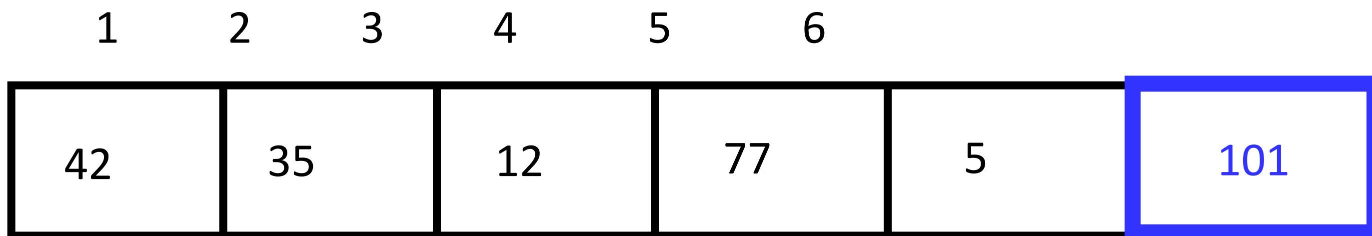
- Iteration 2:
- 35 42 12 77 5 101
- 35 12 42 77 5 101
- 35 12 42 5 77 101



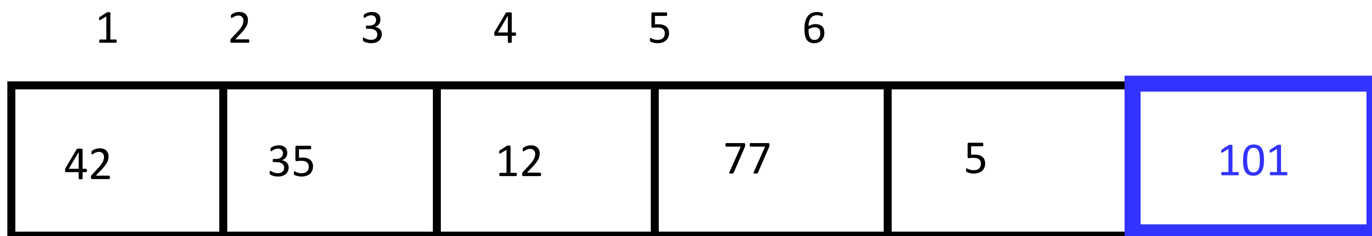
- Iteration 3:
- 35 12 42 5 77 101
- 12 35 42 5 77 101
- 12 35 5 42 77 101



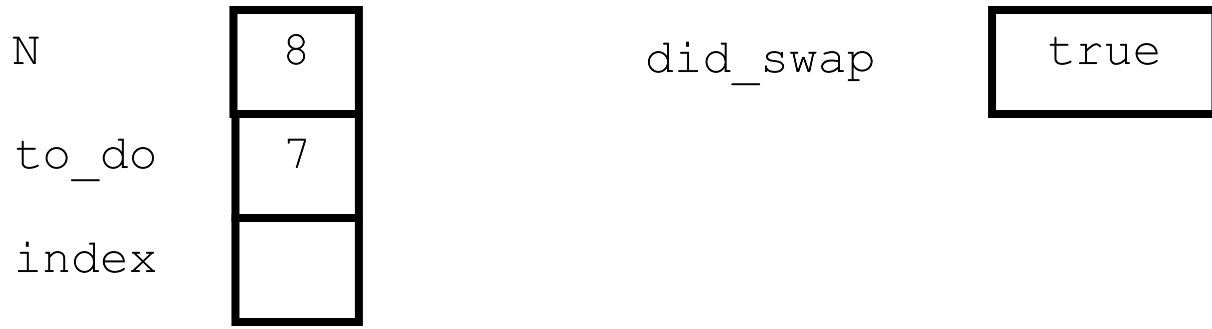
- Iteration 4:
- 12 35 5 42 77 101
- 12 5 35 42 77 101

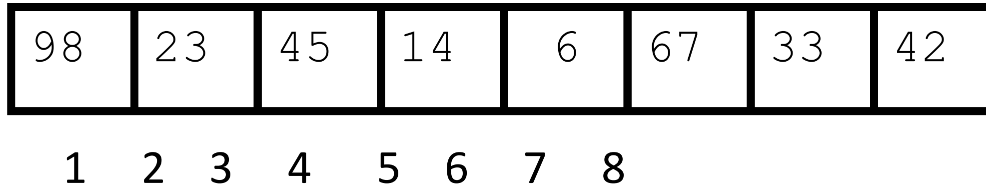


- Iteration 5:
- 12 5 35 42 77 101
- ->
- 5 12 35 42 77 101

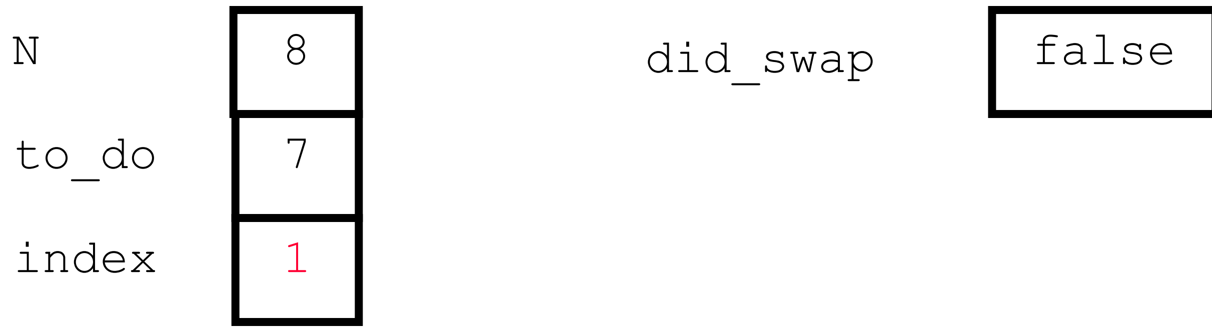


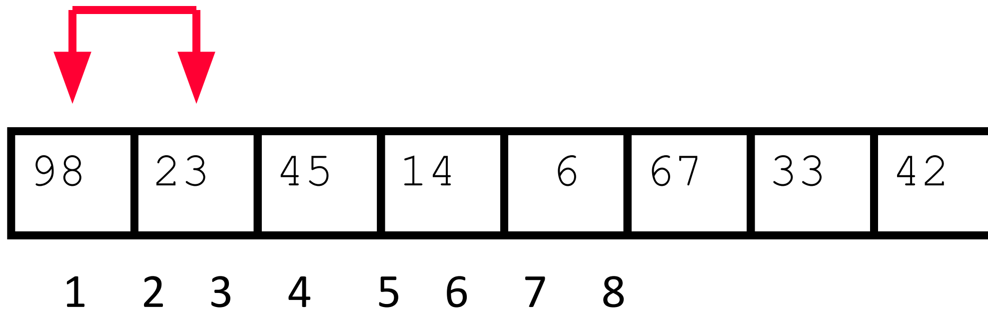
An Animated Example



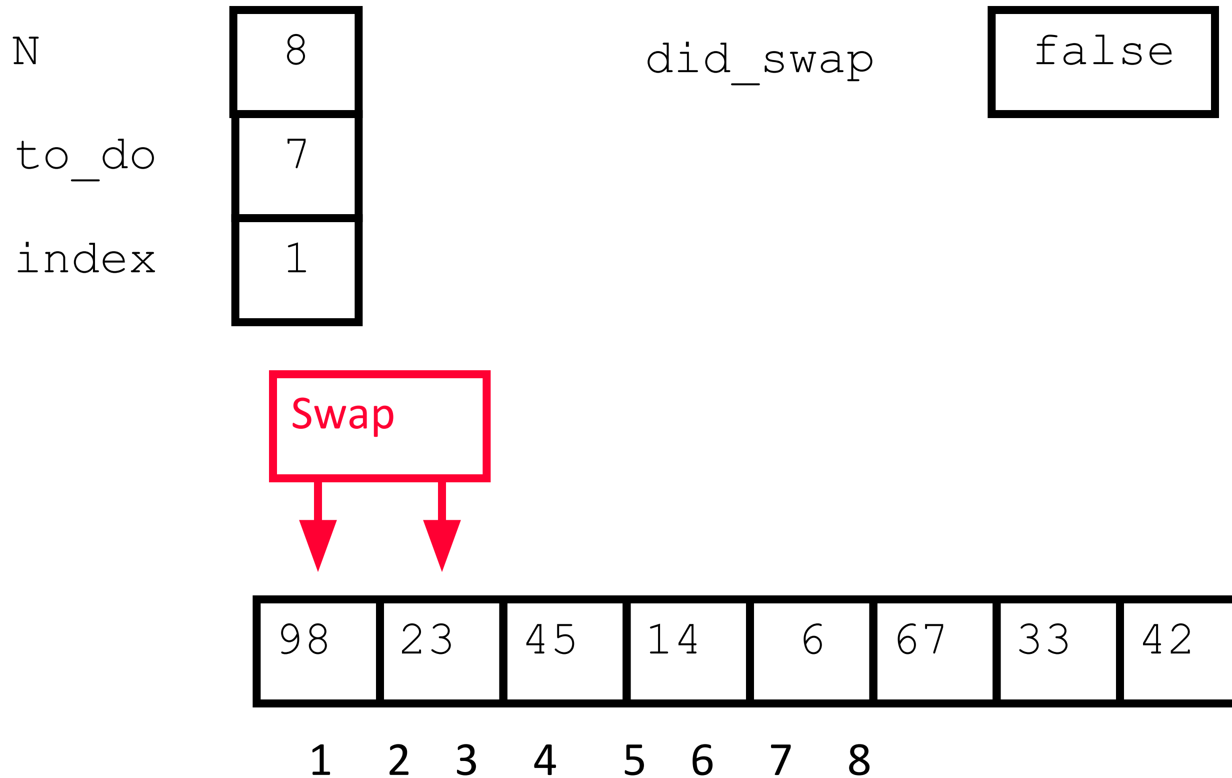


An Animated Example

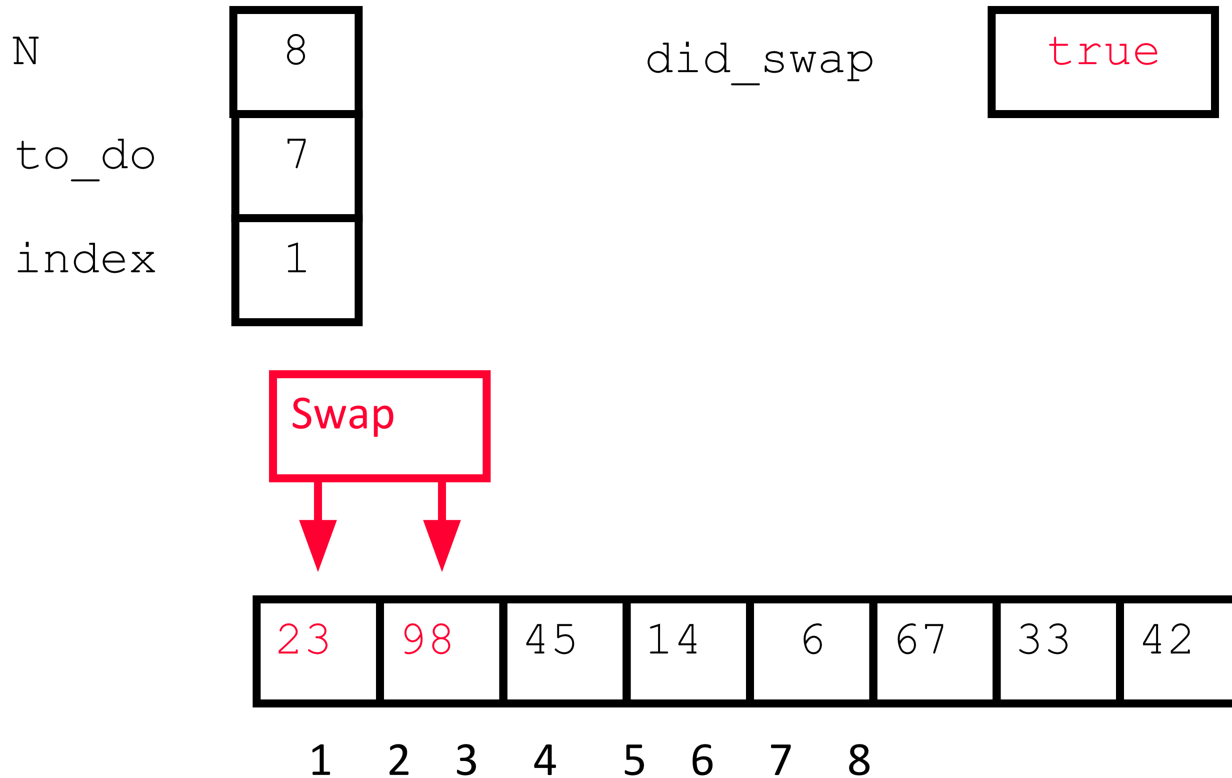




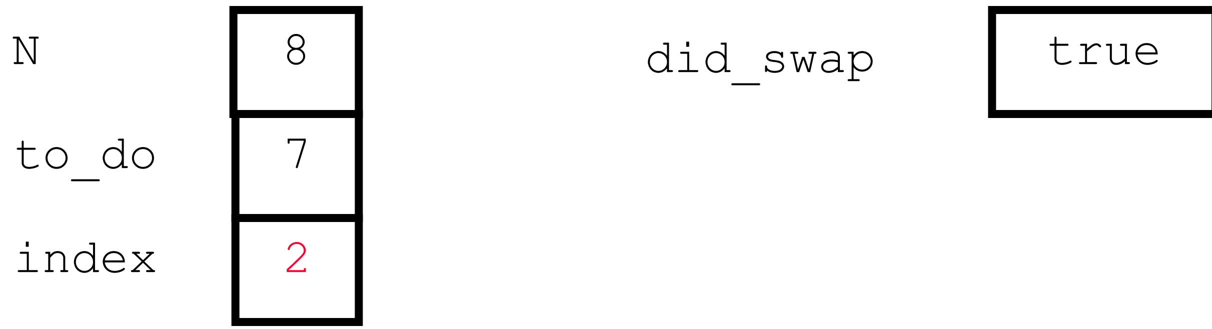
An Animated Example

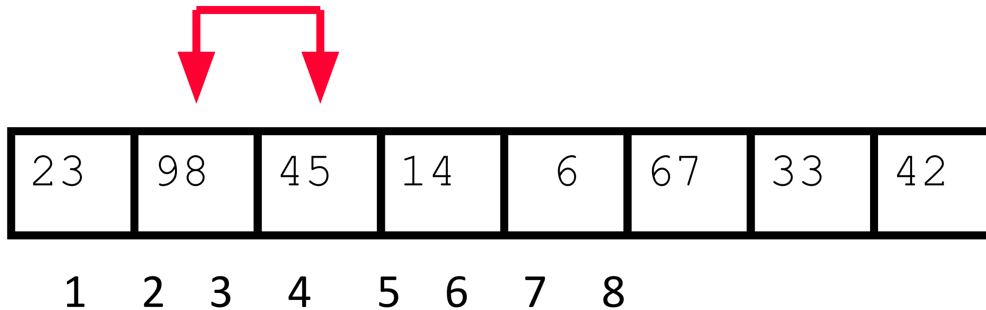


An Animated Example



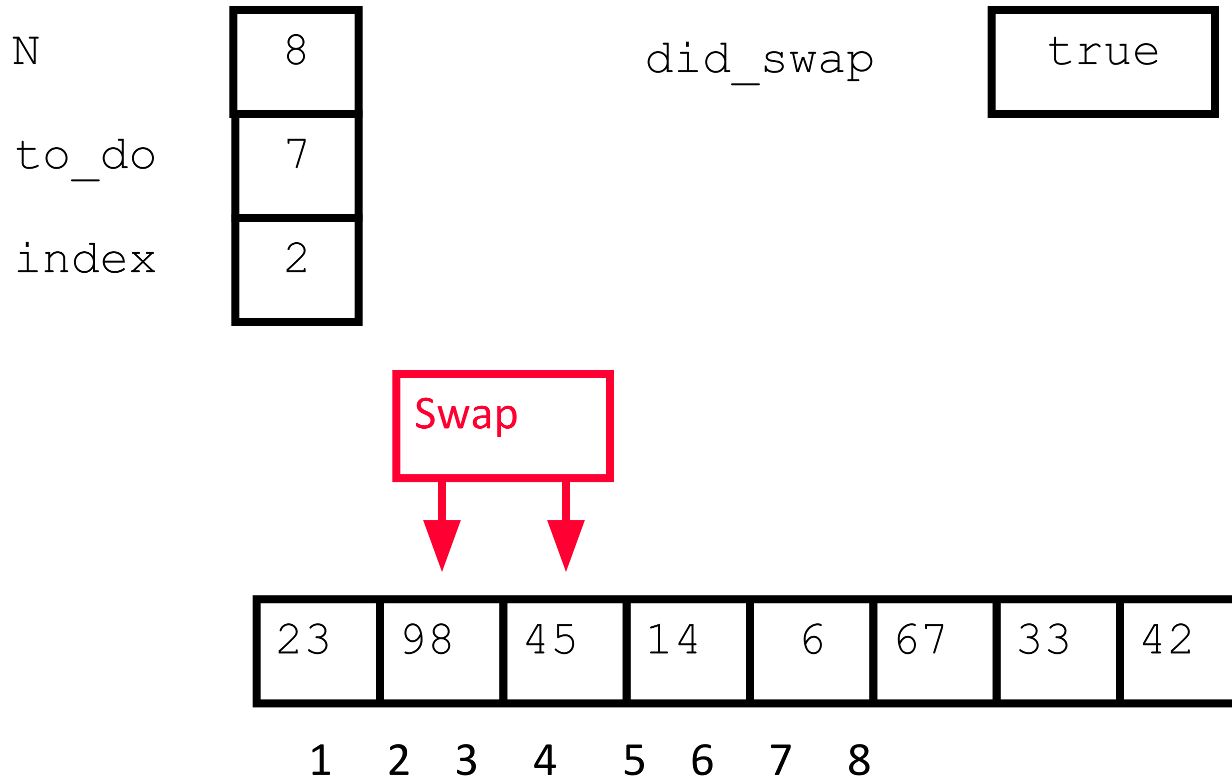
An Animated Example



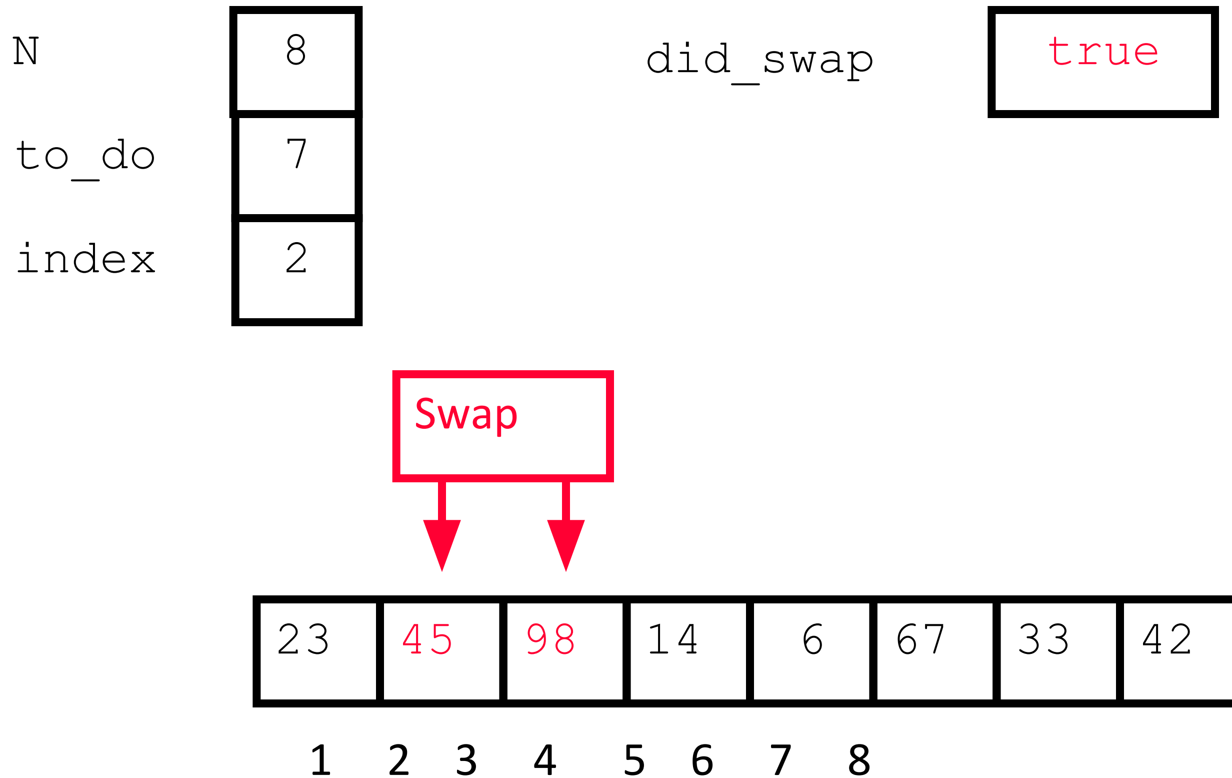


1 2 3 4 5 6 7 8

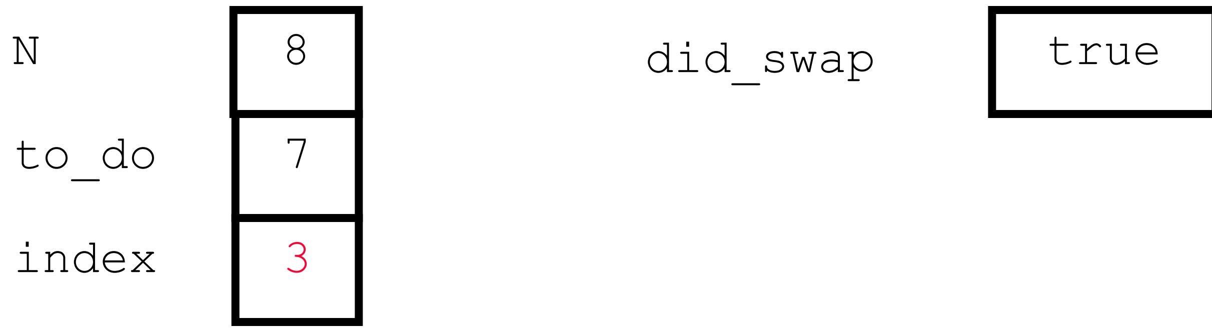
An Animated Example

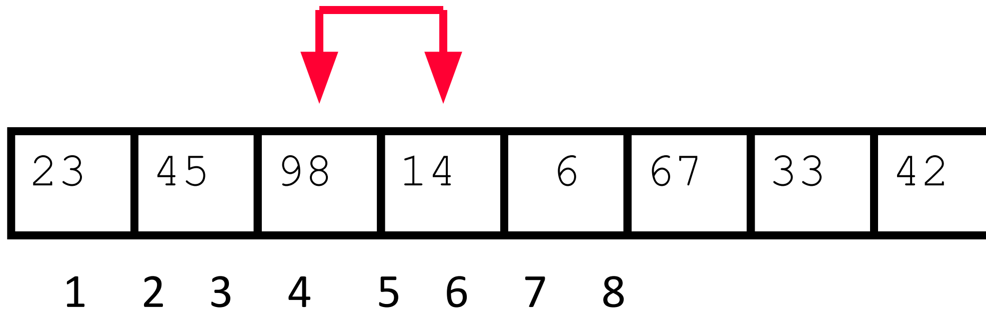


An Animated Example

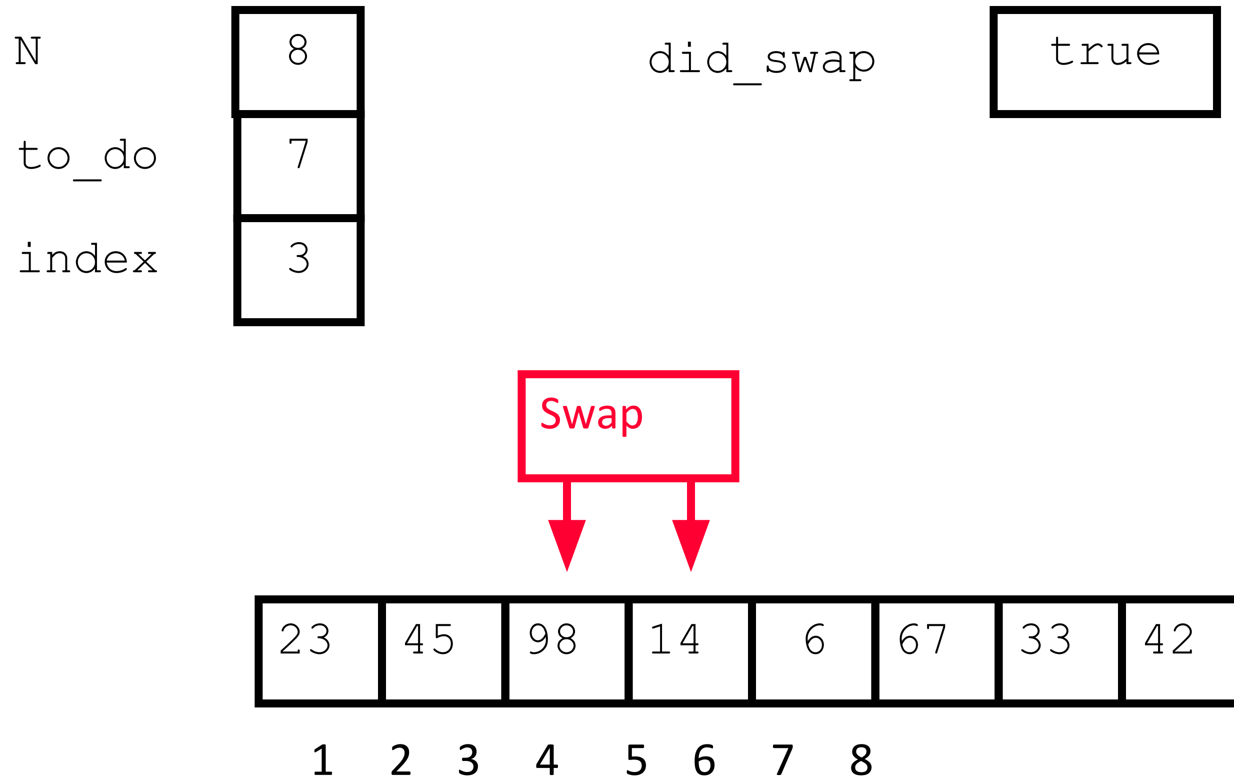


An Animated Example

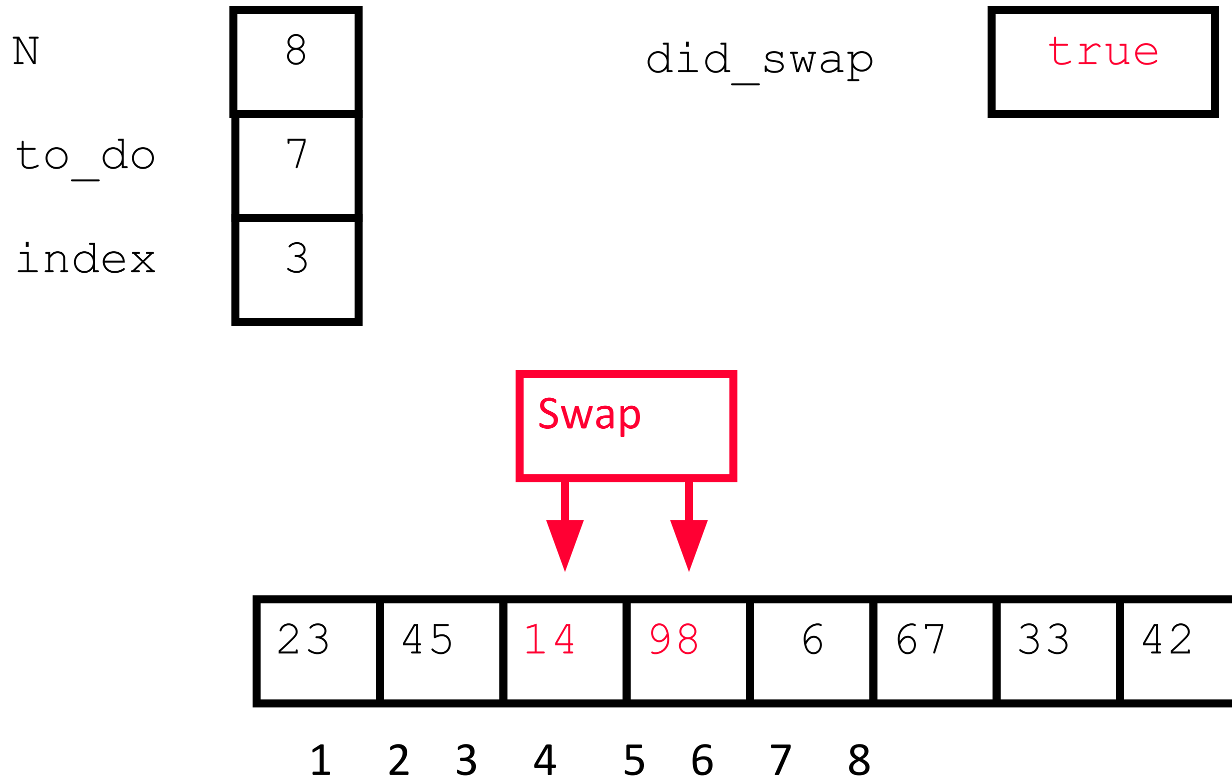




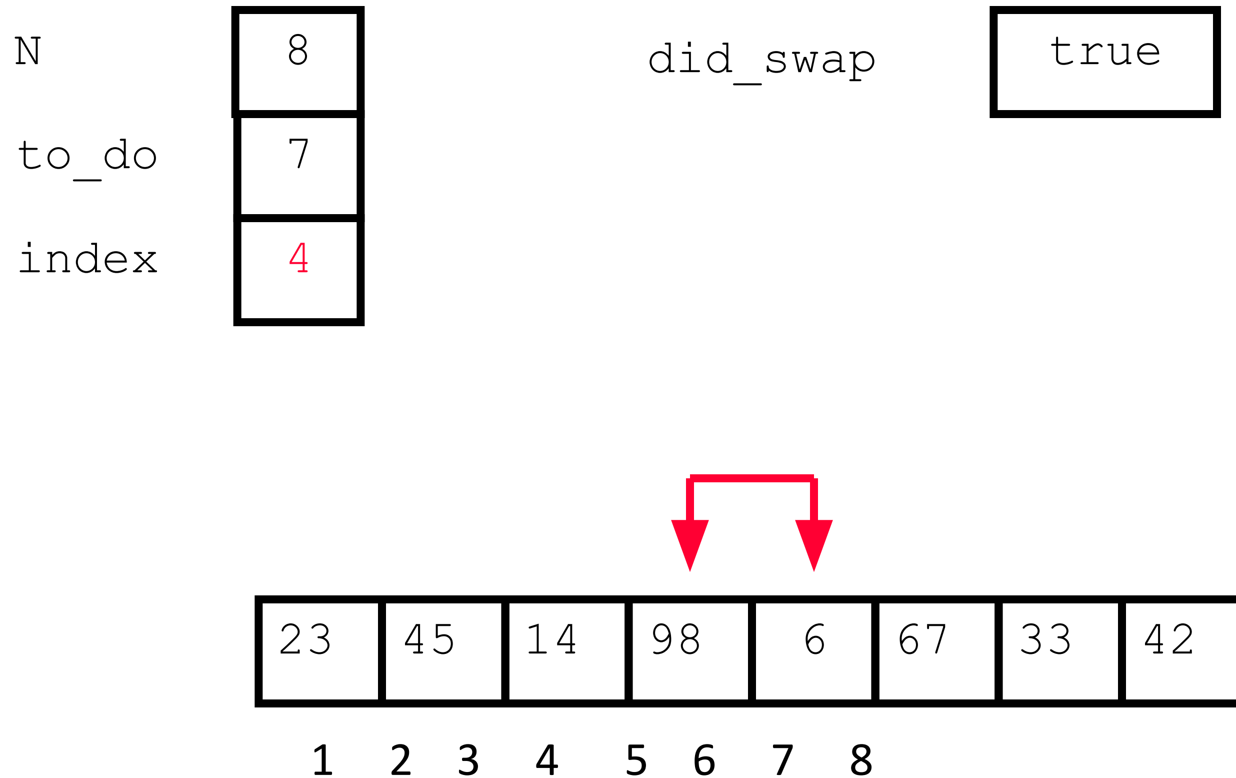
An Animated Example



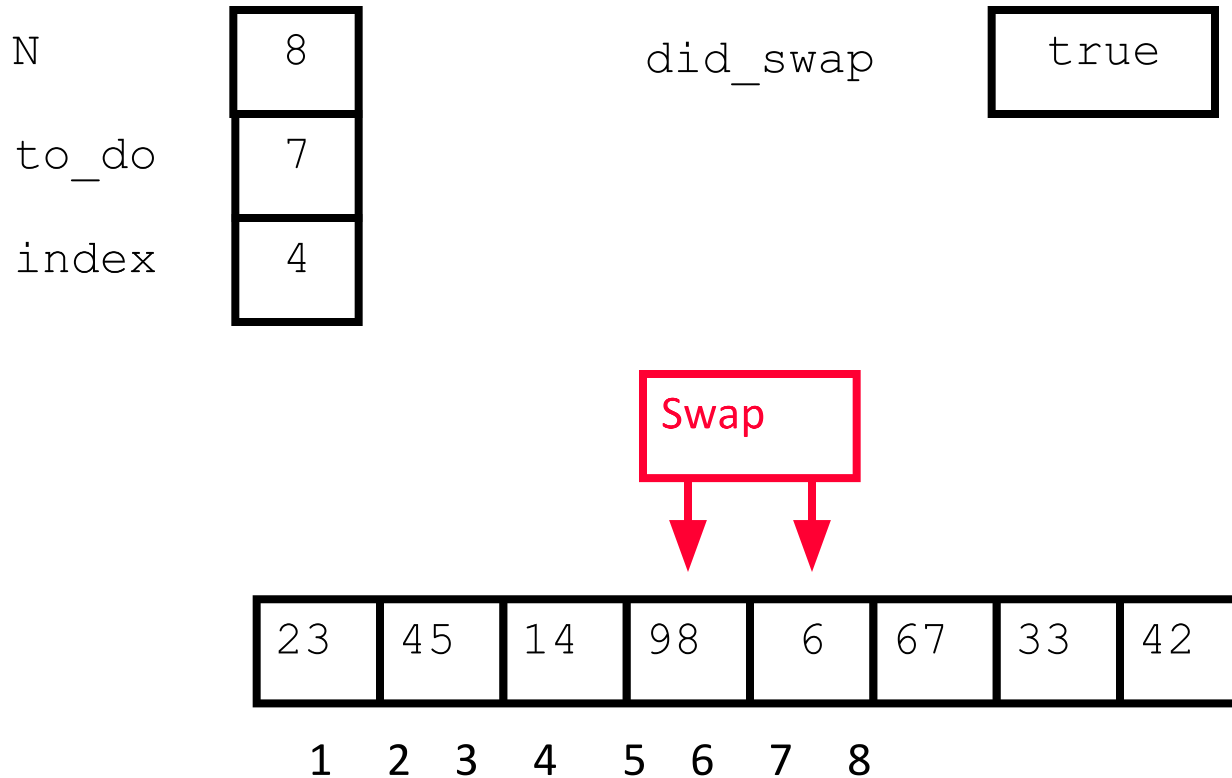
An Animated Example



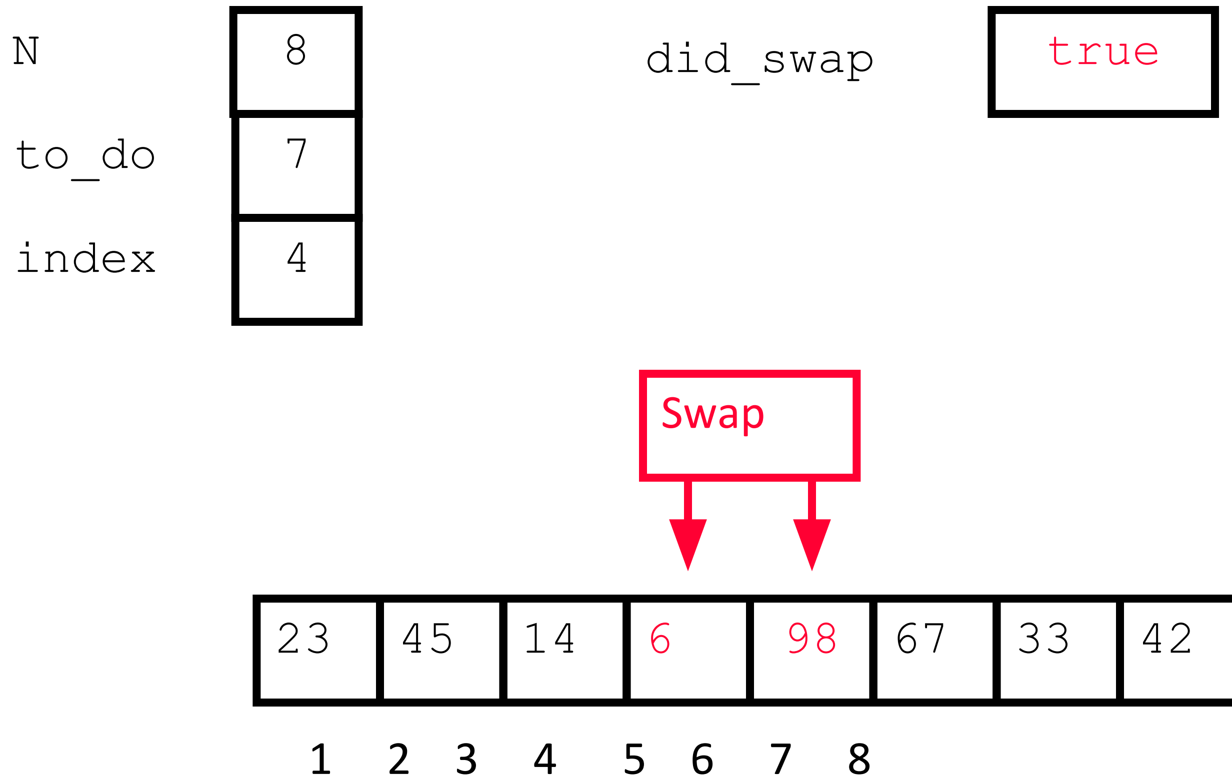
An Animated Example



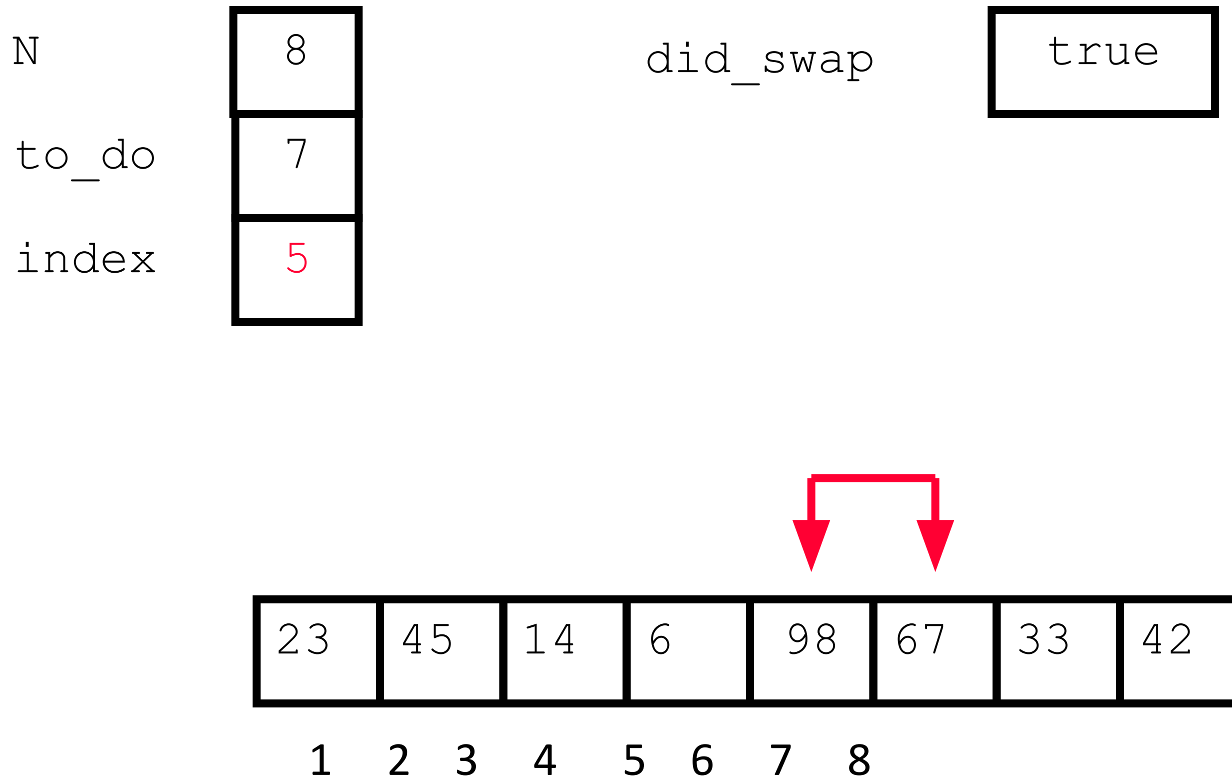
An Animated Example



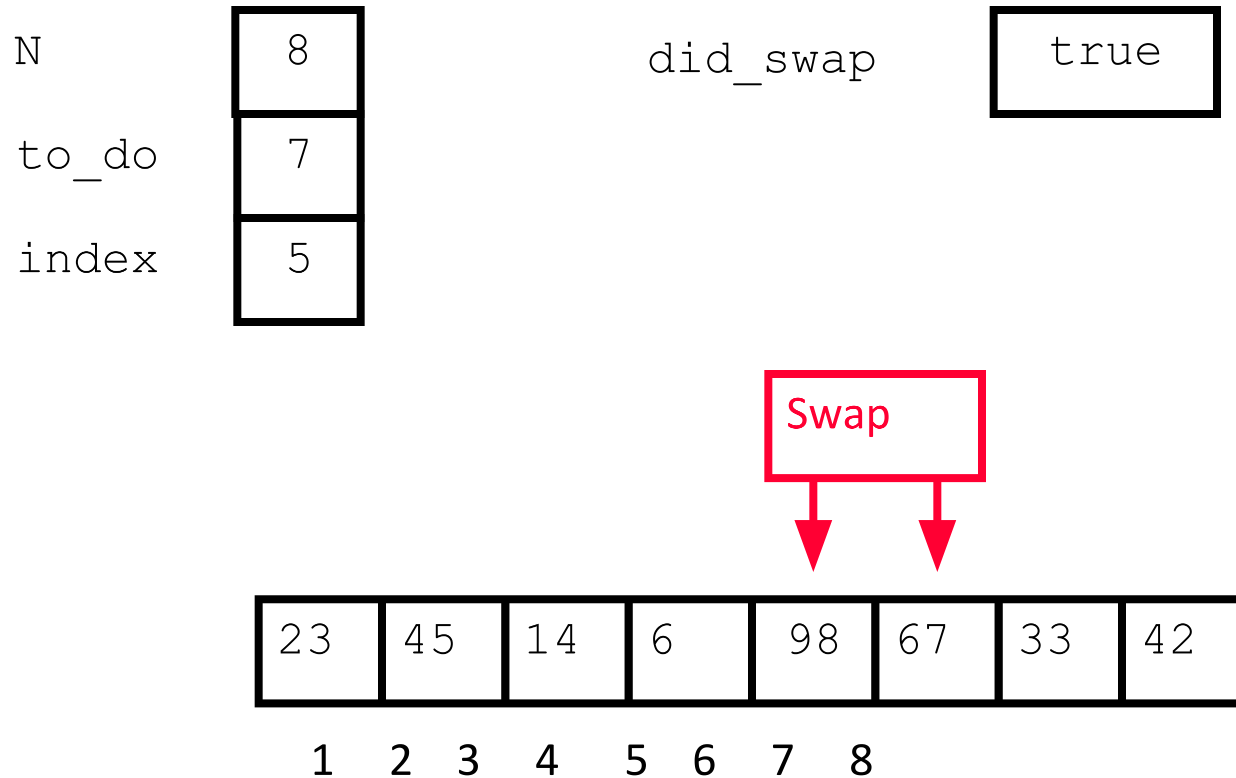
An Animated Example



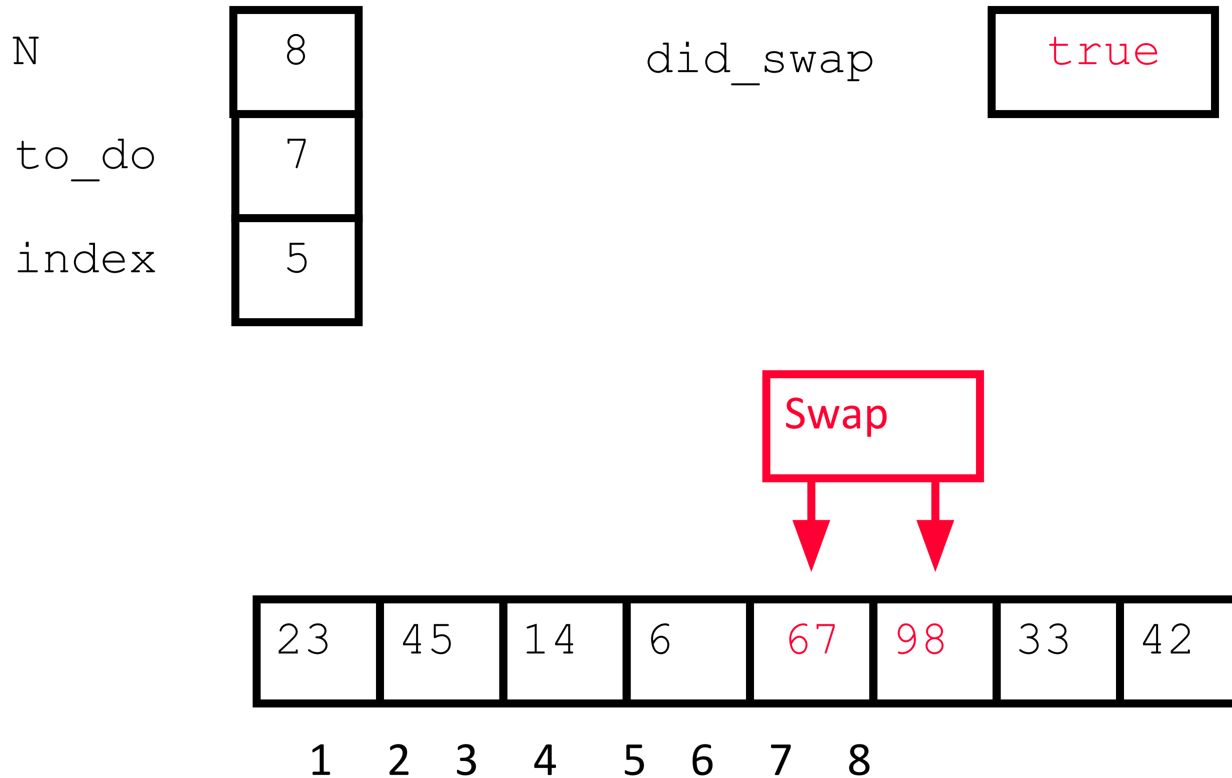
An Animated Example



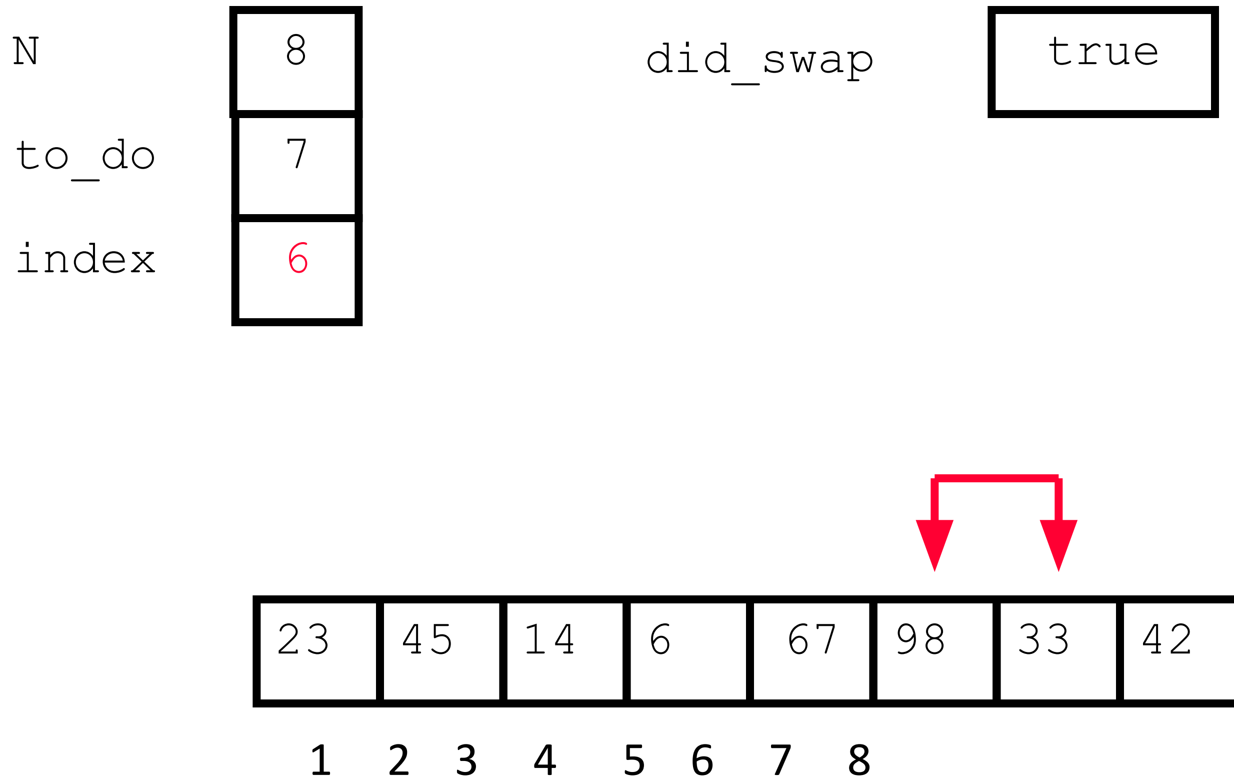
An Animated Example



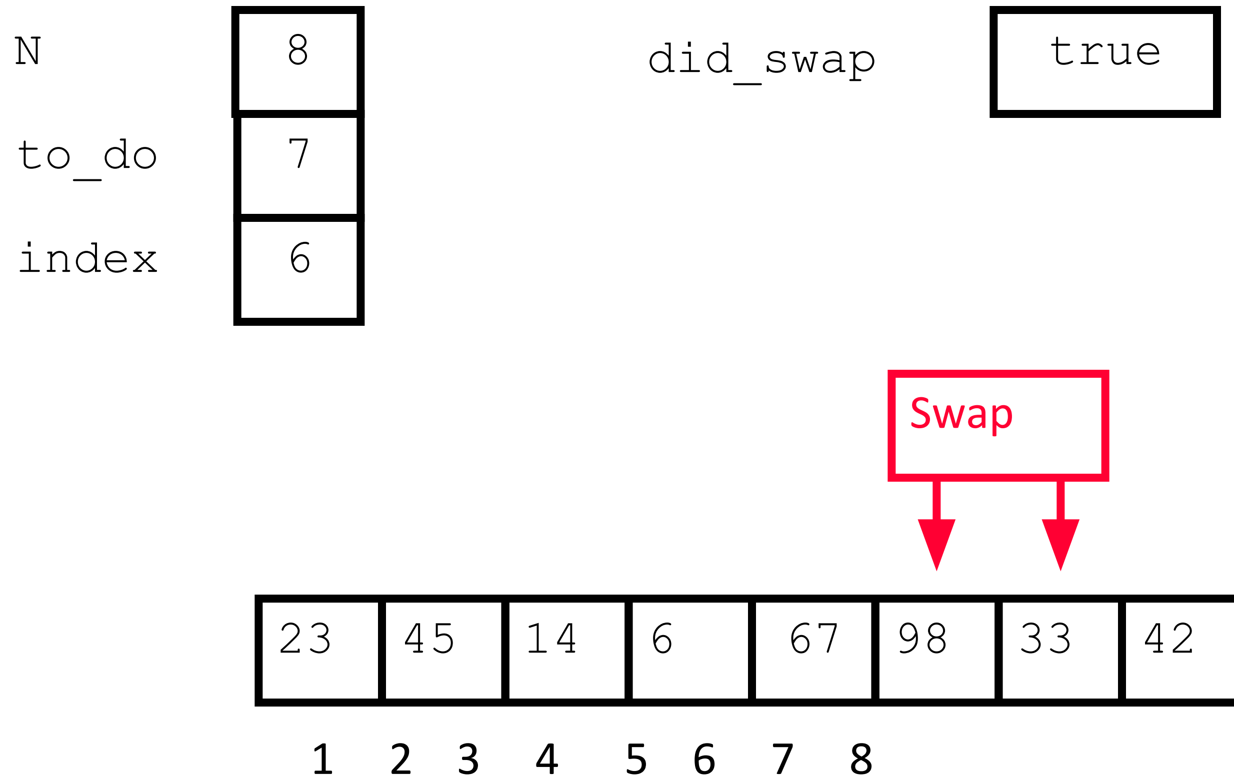
An Animated Example



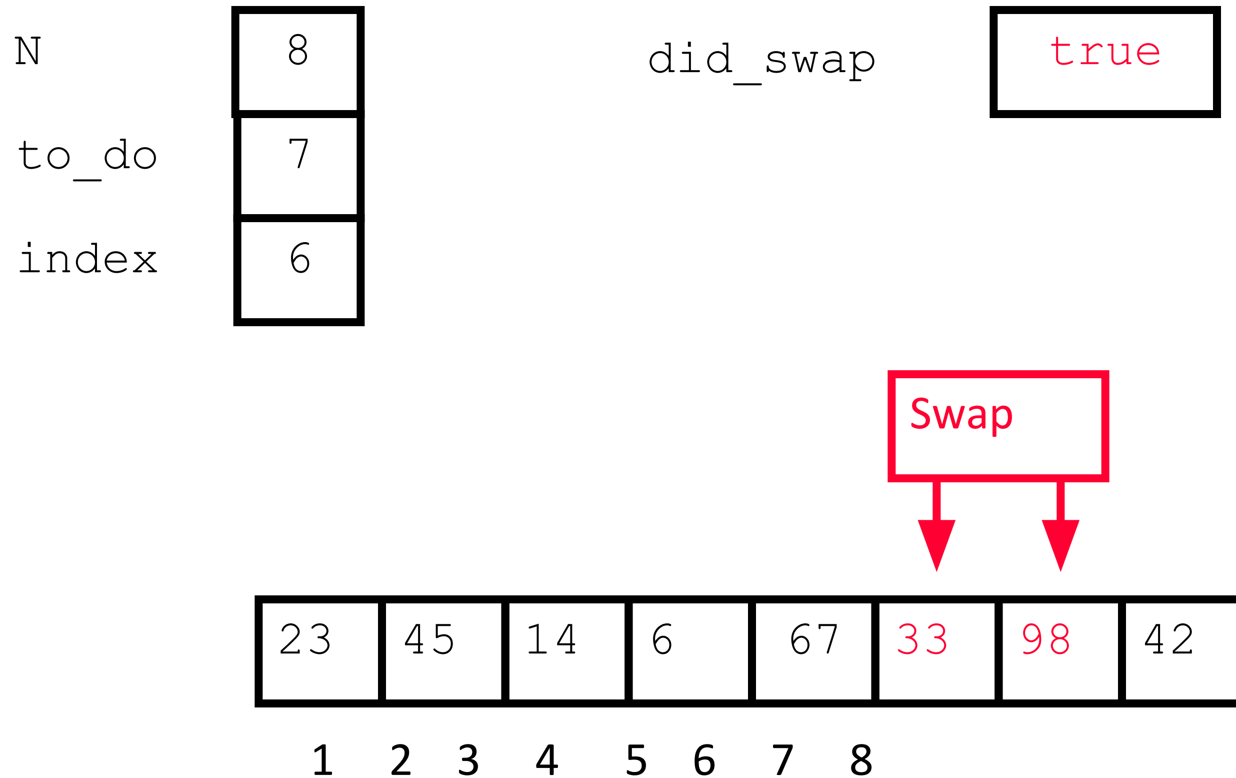
An Animated Example



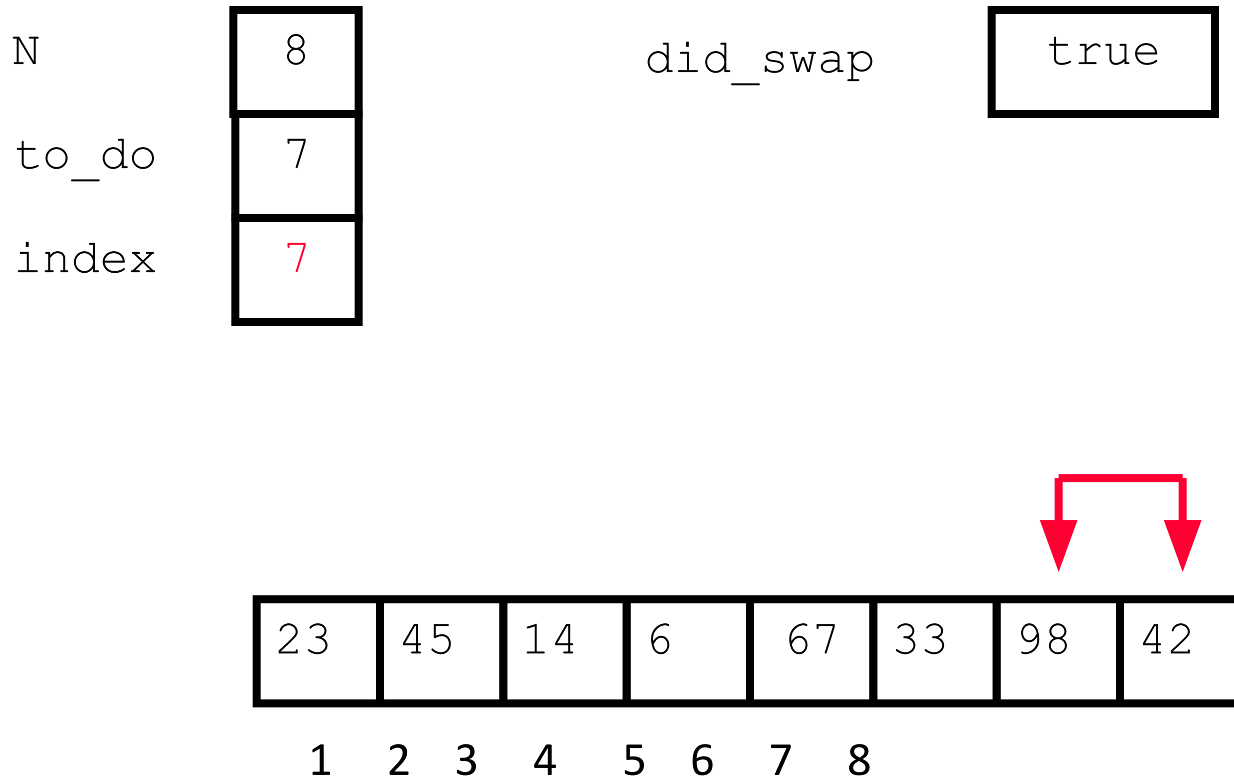
An Animated Example



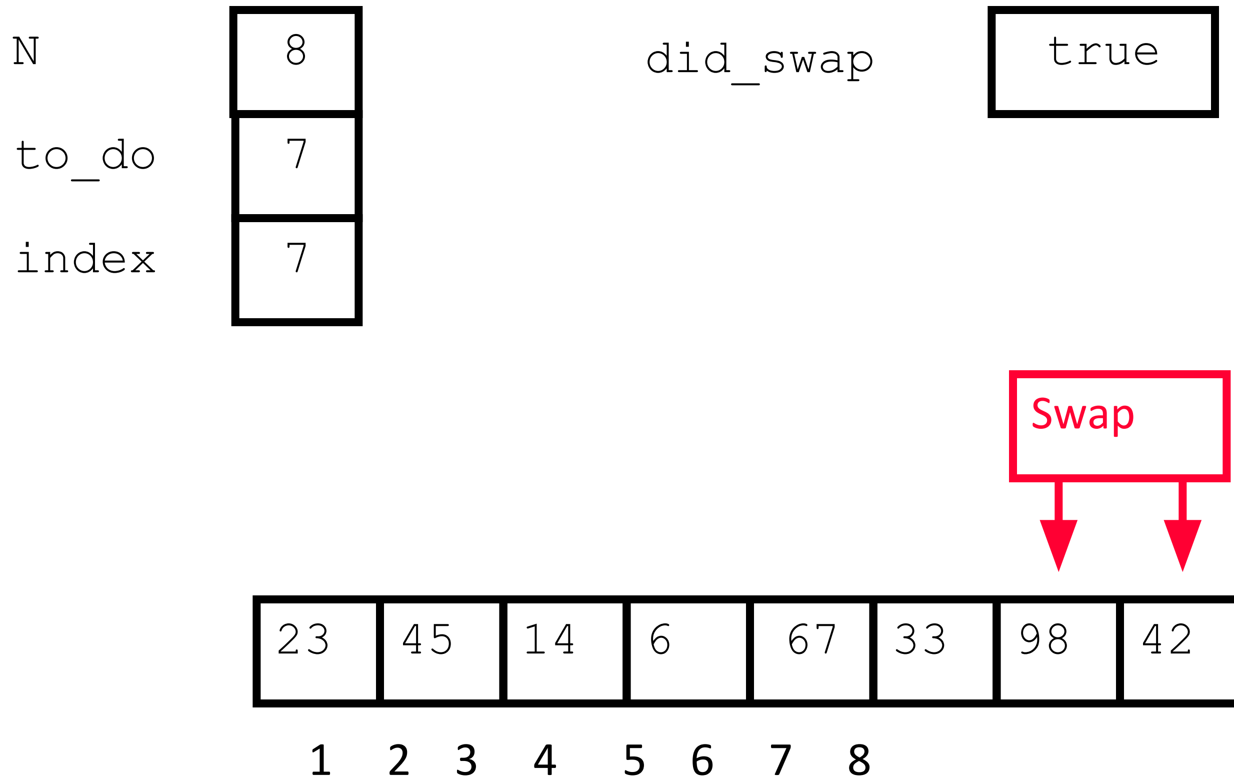
An Animated Example



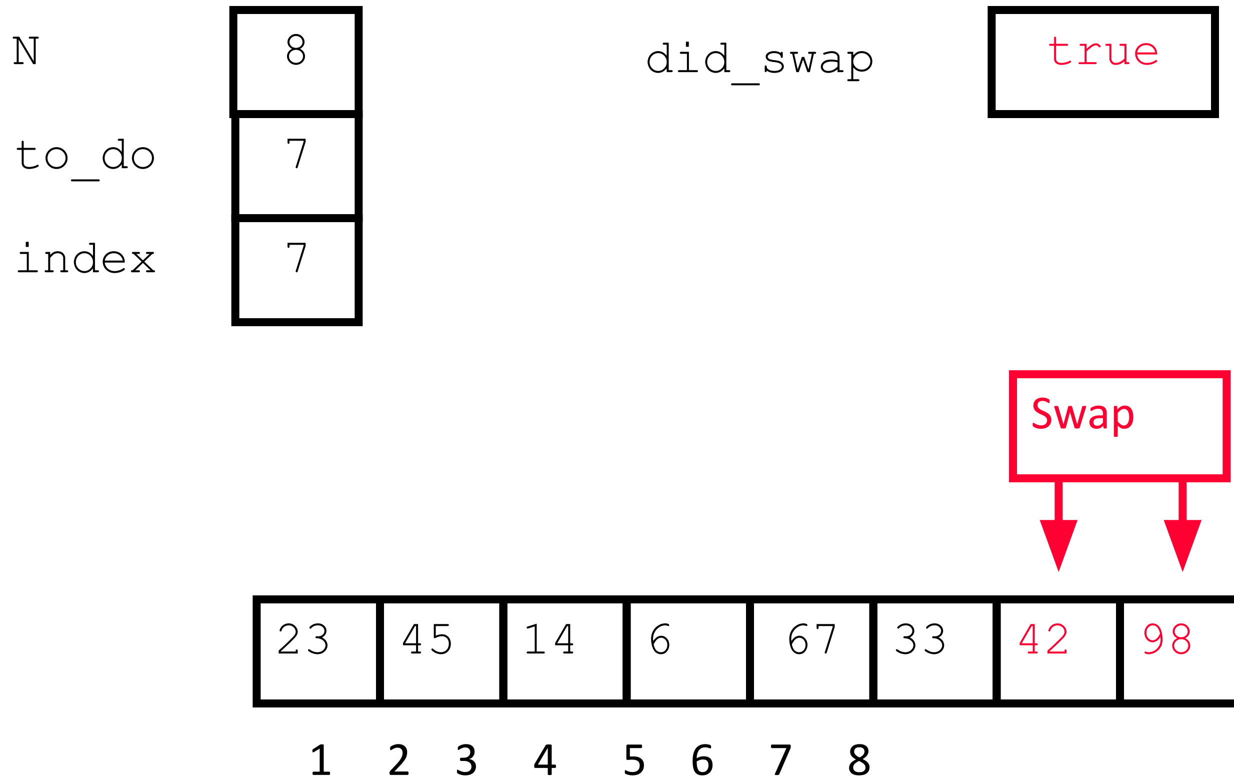
An Animated Example



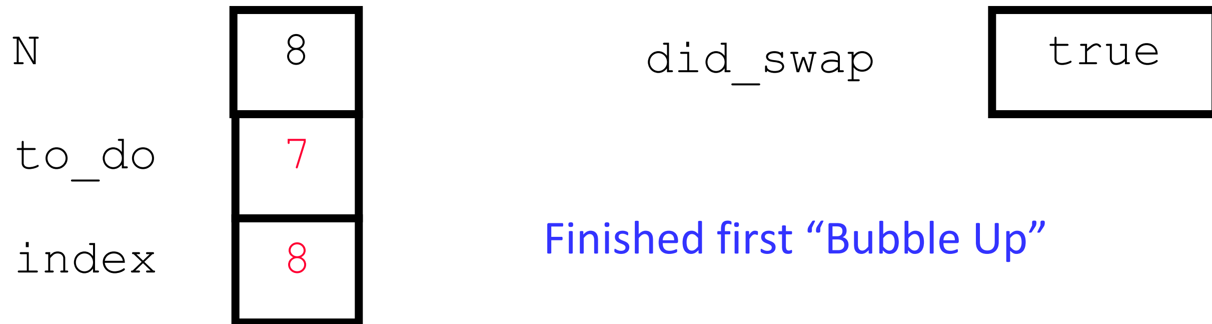
An Animated Example



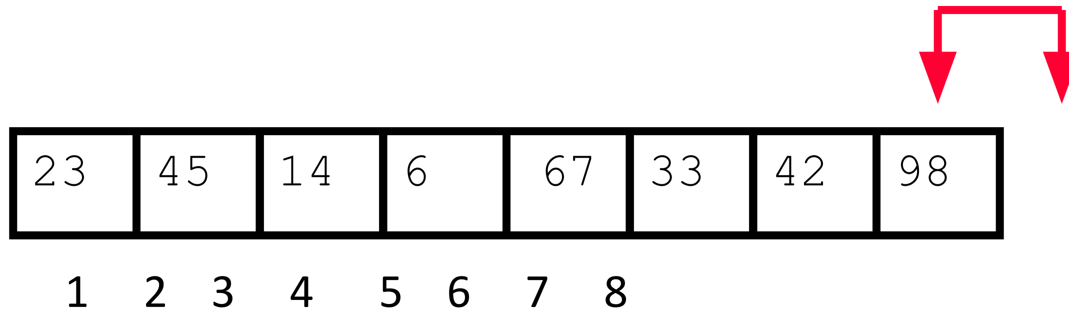
An Animated Example



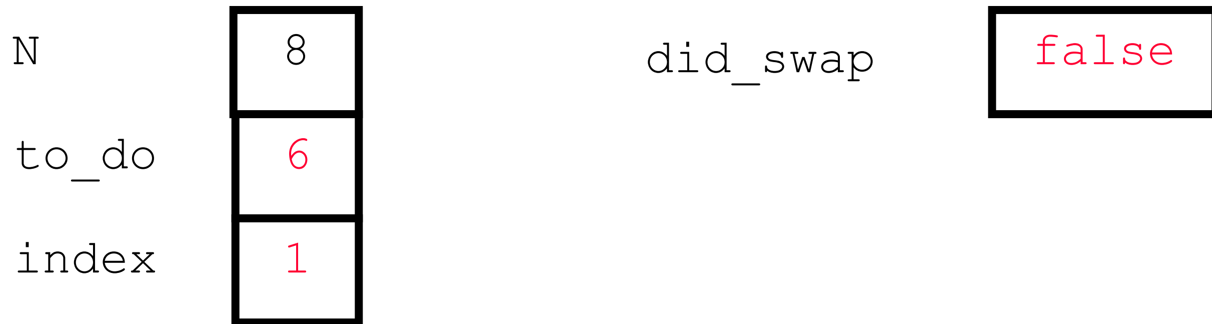
After First Pass of Outer Loop

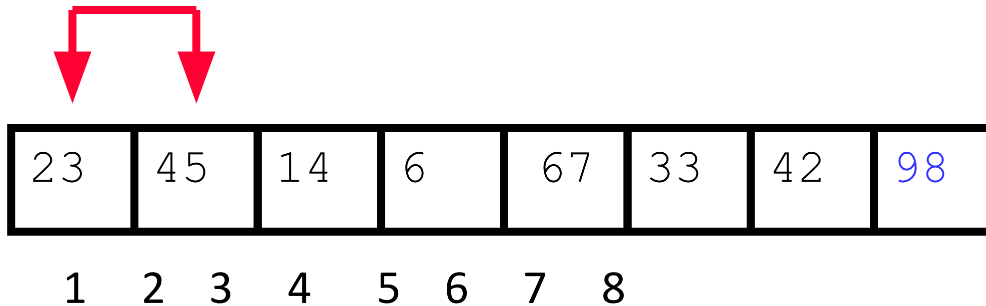


Finished first “Bubble Up”



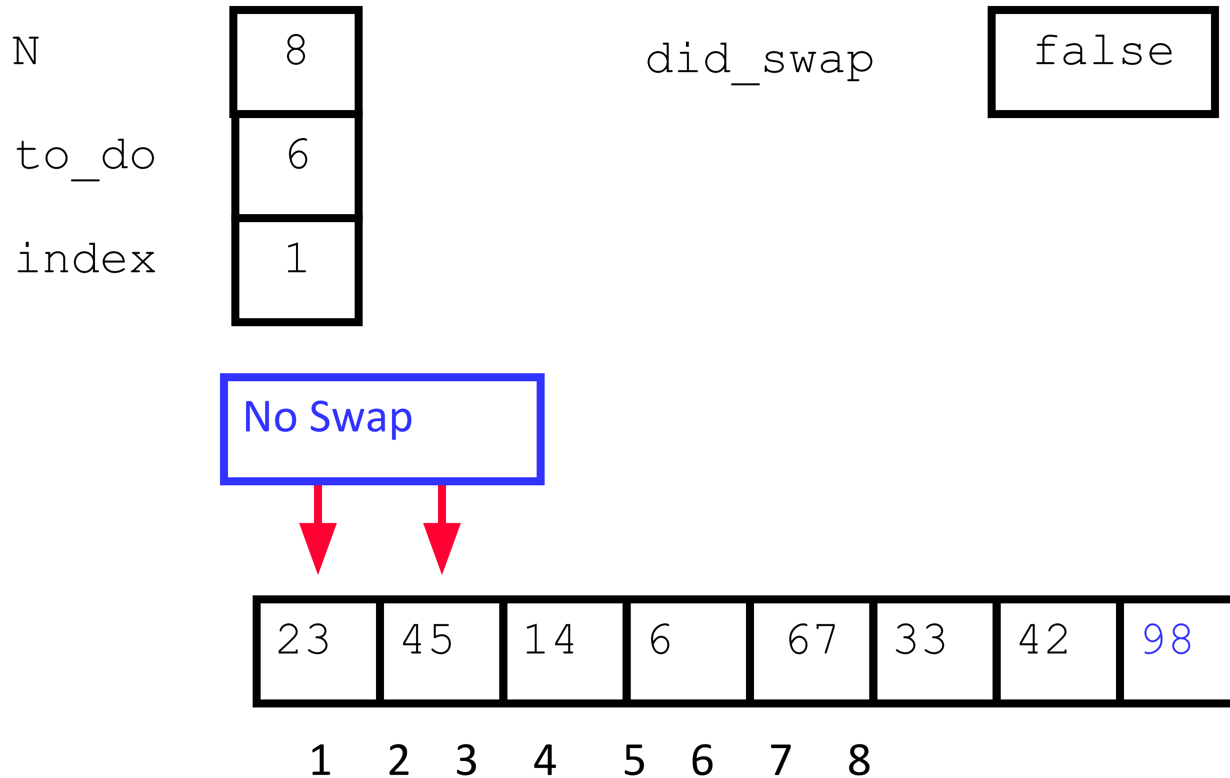
The Second "Bubble Up"



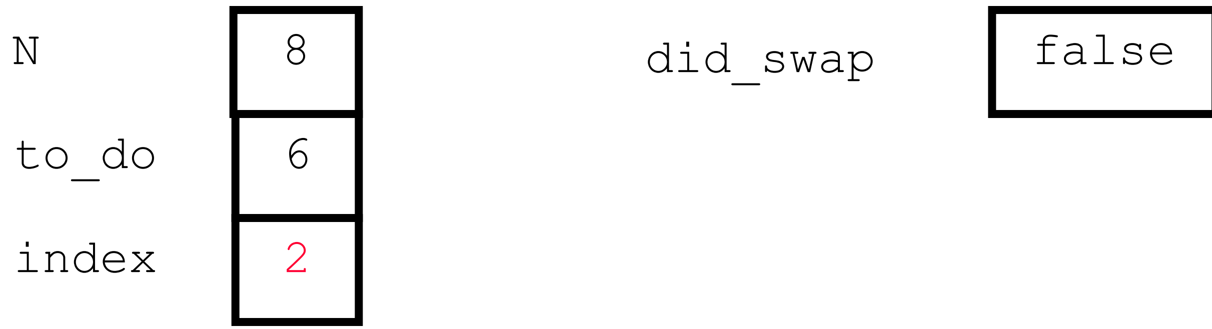


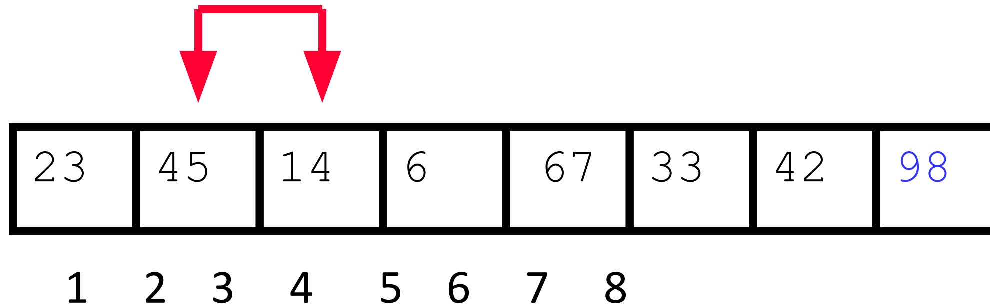
1 2 3 4 5 6 7 8

The Second "Bubble Up"

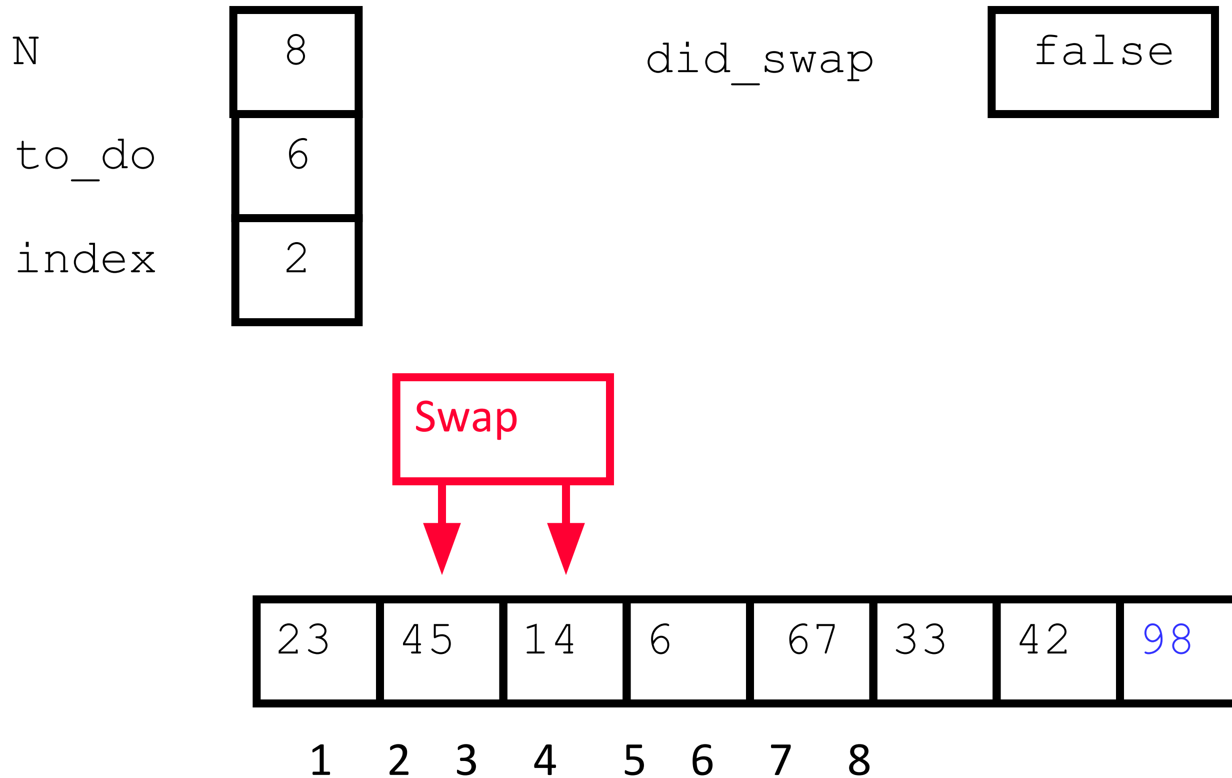


The Second “Bubble Up”

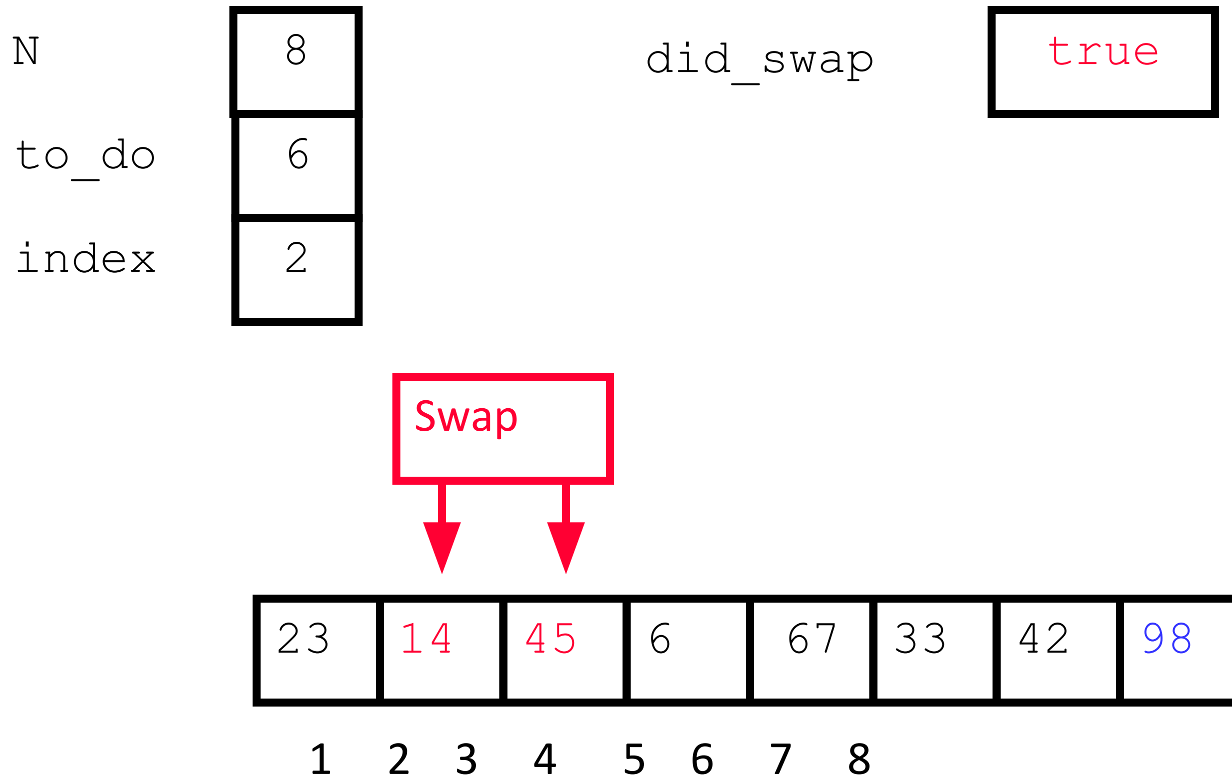




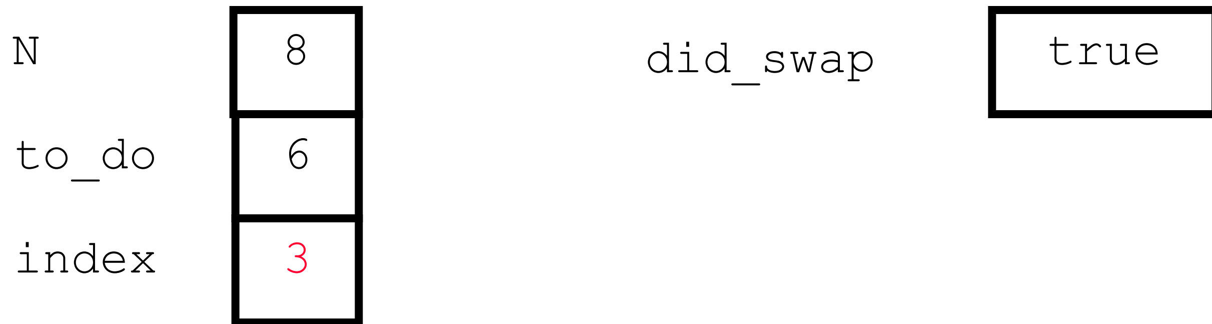
The Second "Bubble Up"

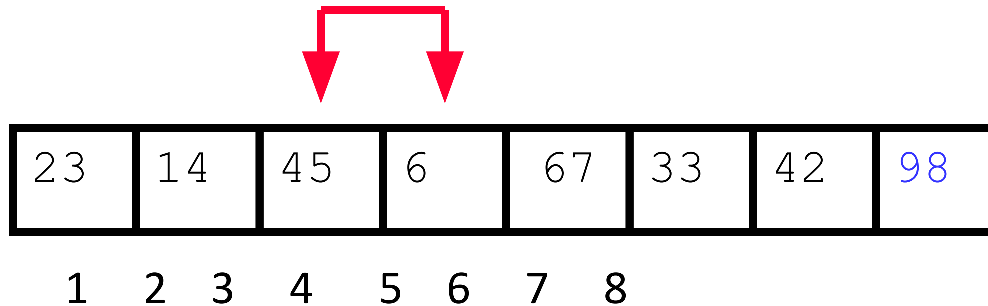


The Second “Bubble Up”

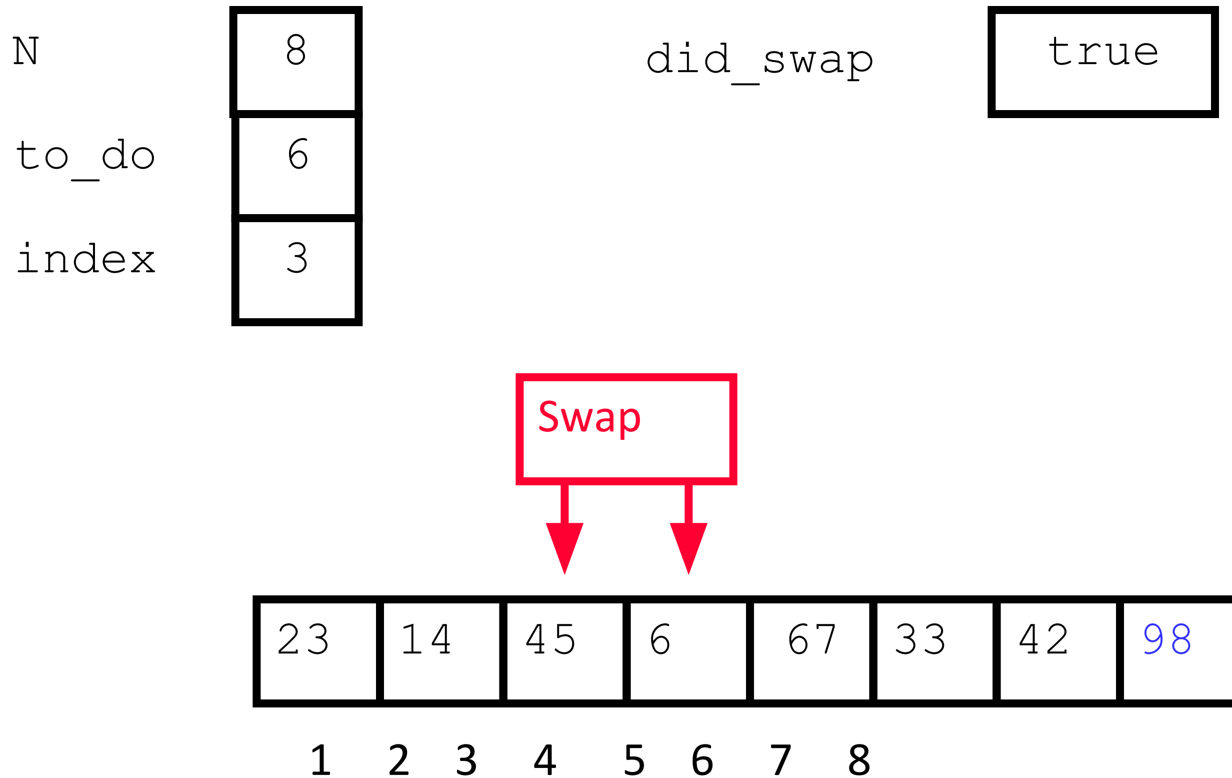


The Second "Bubble Up"

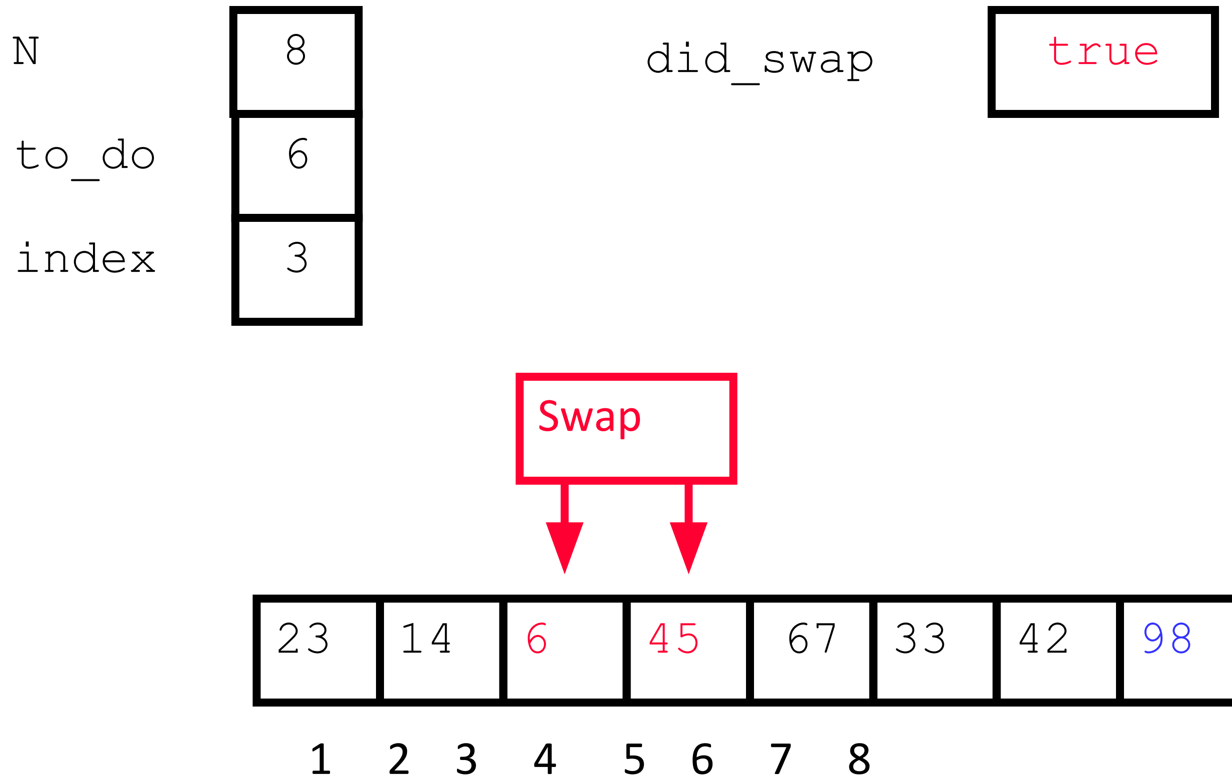




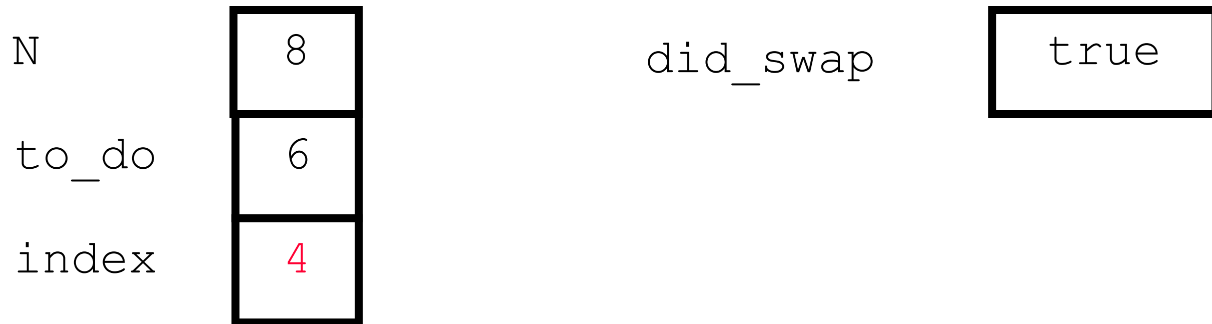
The Second "Bubble Up"

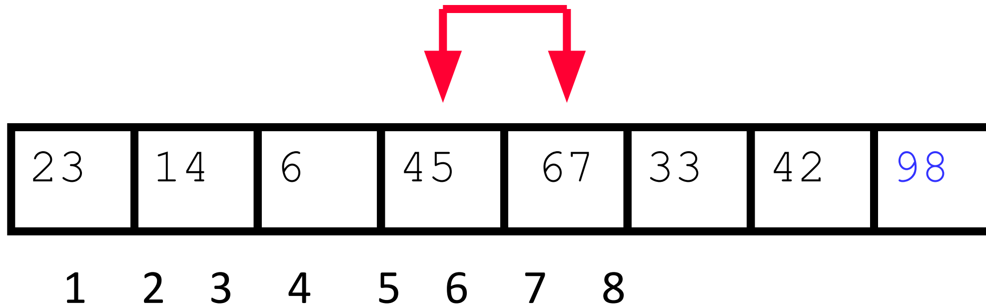


The Second "Bubble Up"

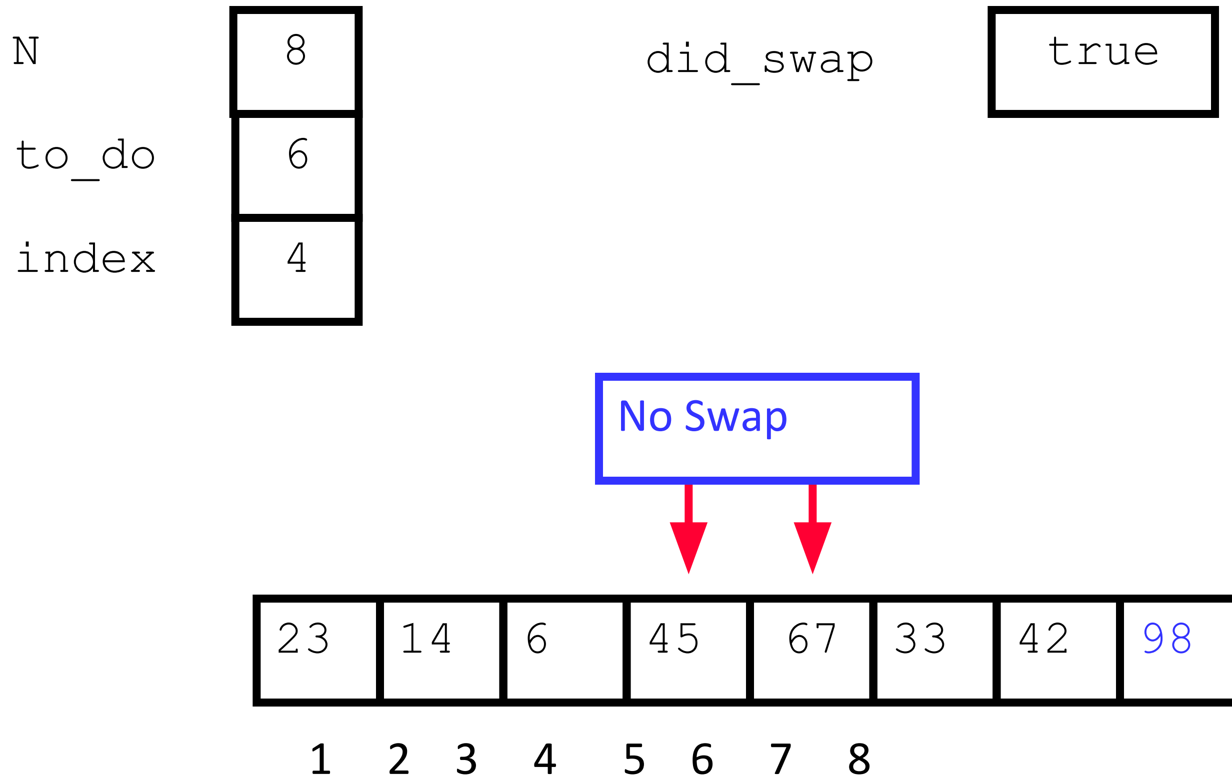


The Second "Bubble Up"

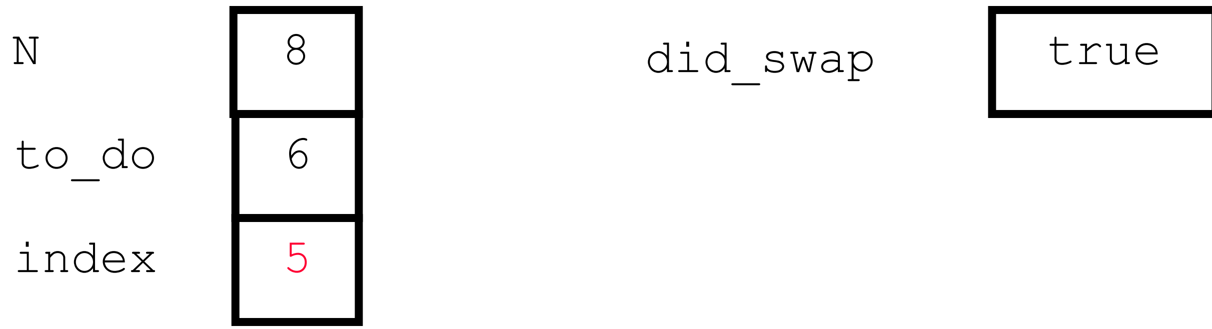


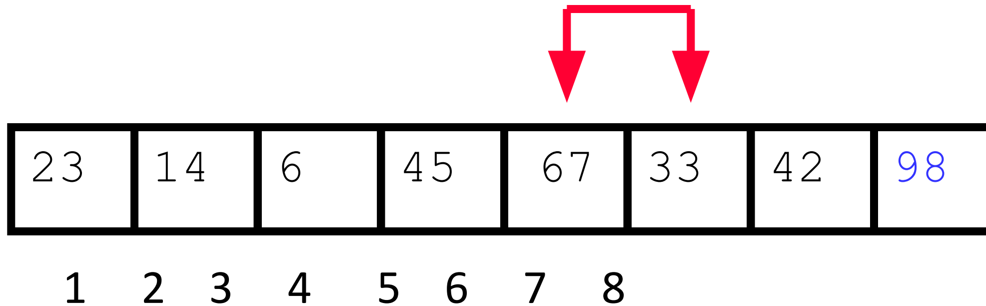


The Second "Bubble Up"

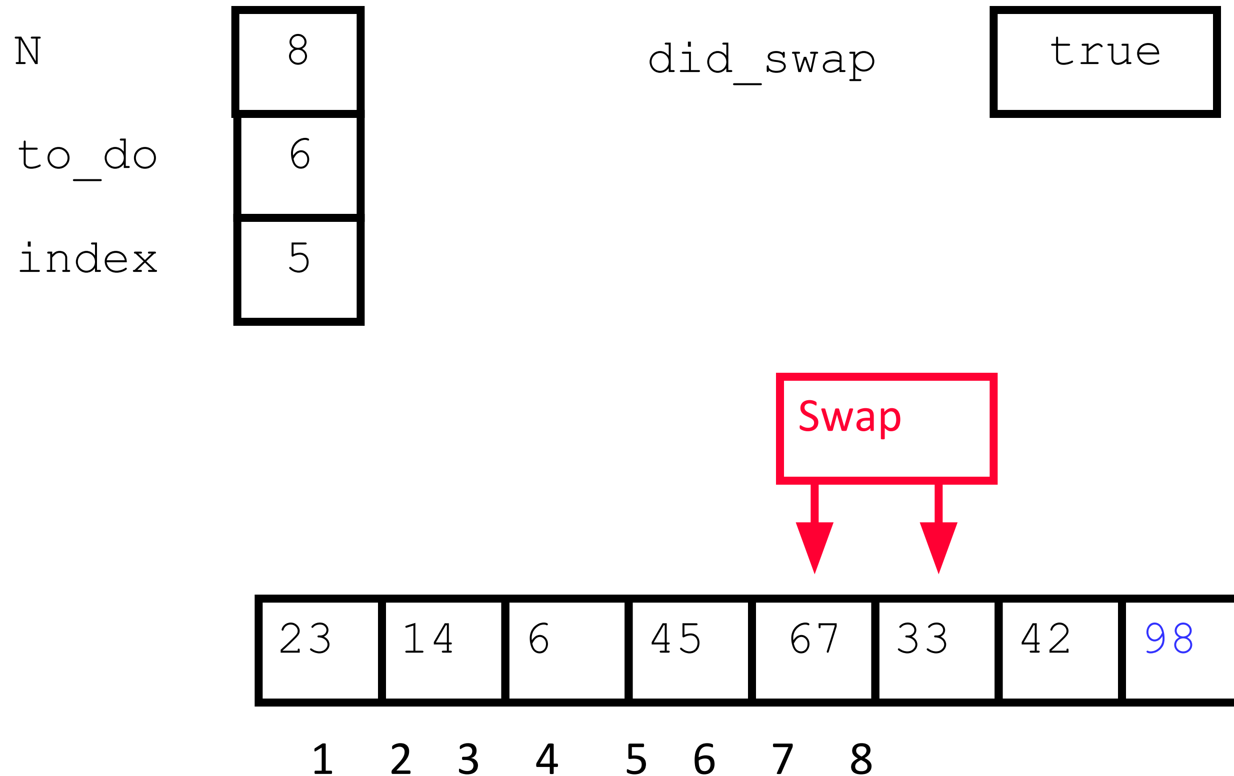


The Second “Bubble Up”

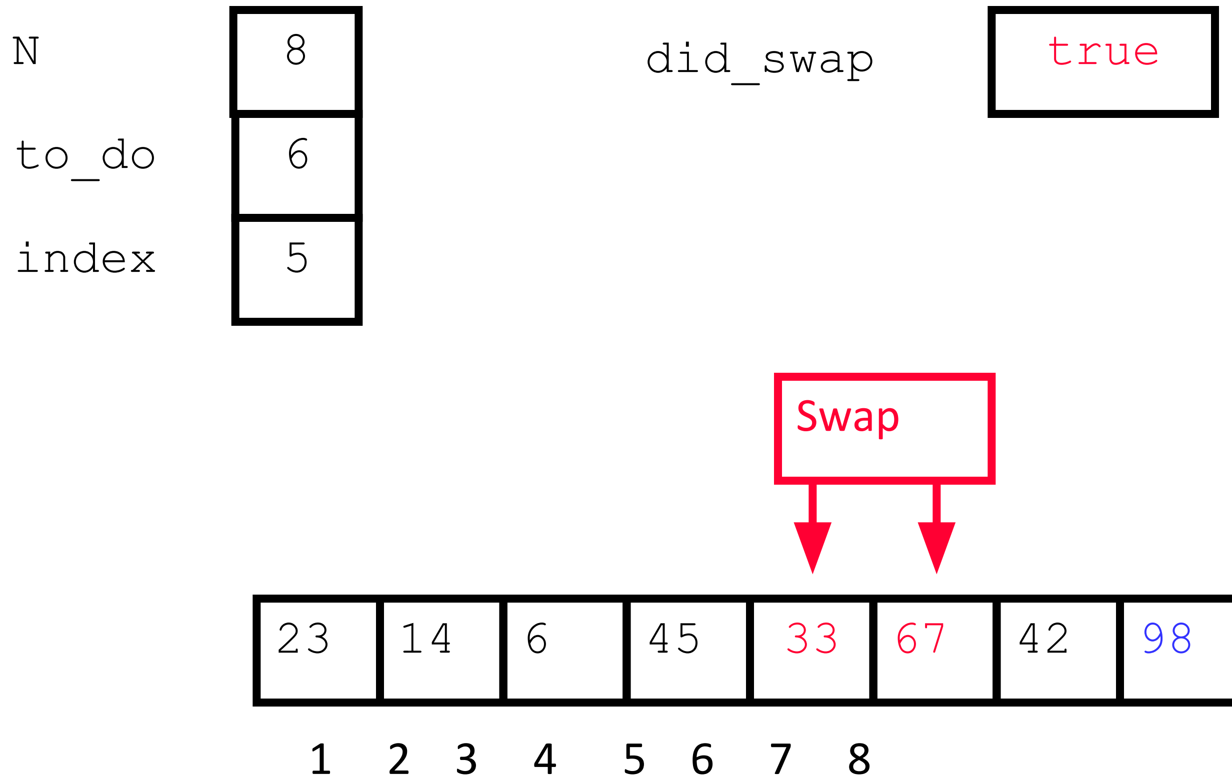




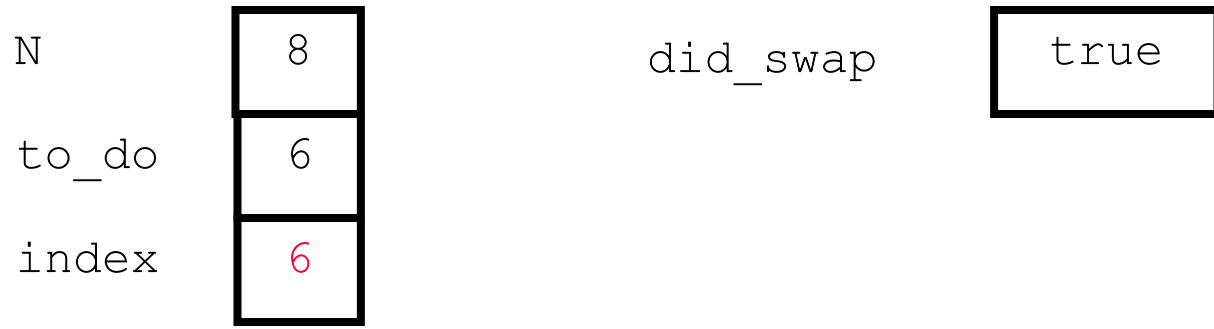
The Second "Bubble Up"

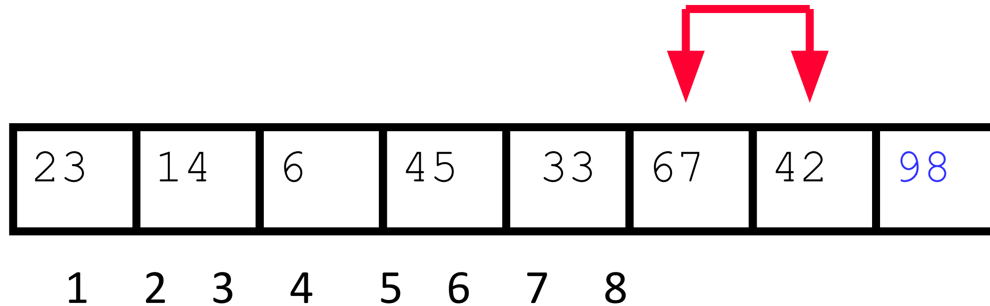


The Second "Bubble Up"



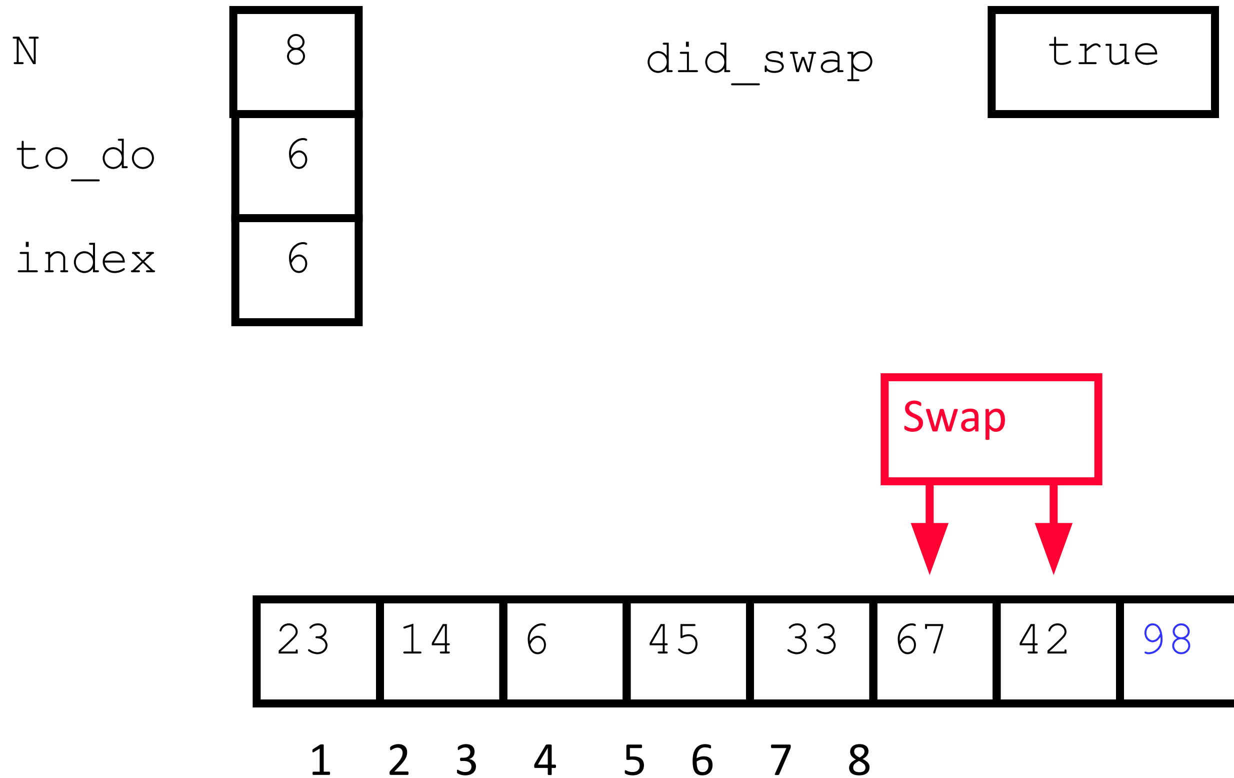
The Second "Bubble Up"



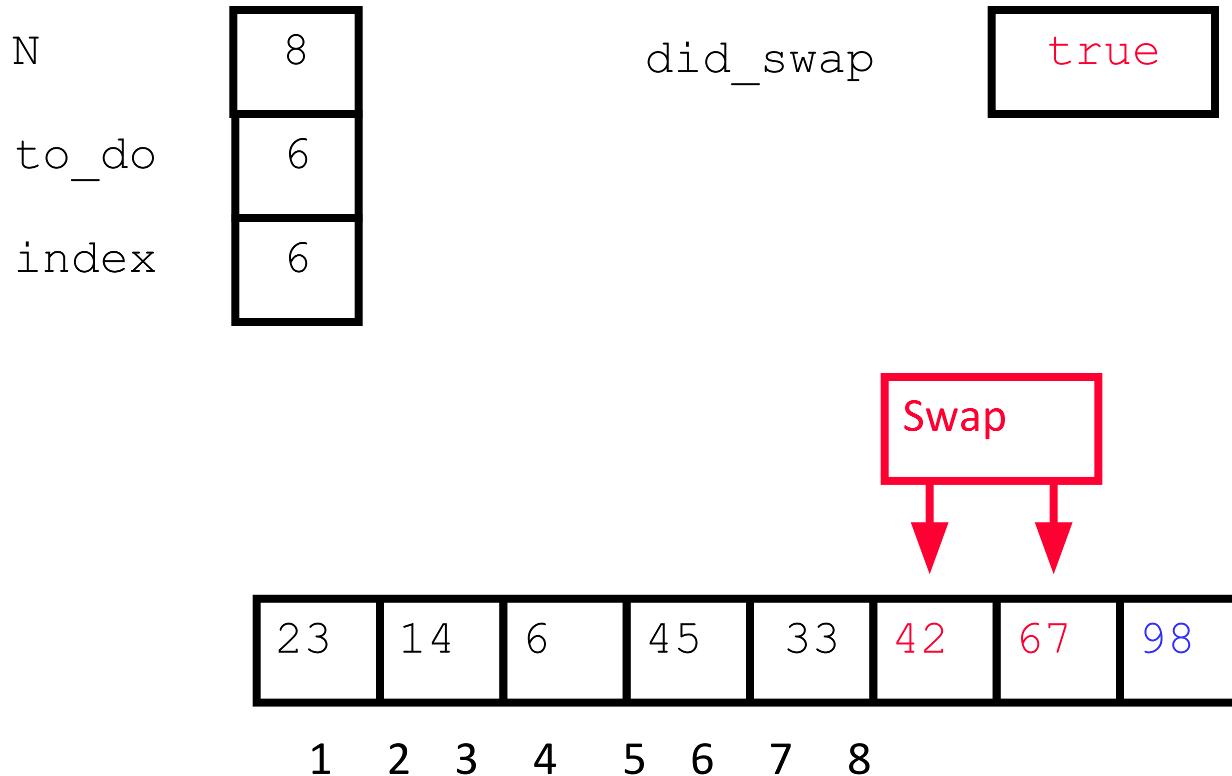


1 2 3 4 5 6 7 8

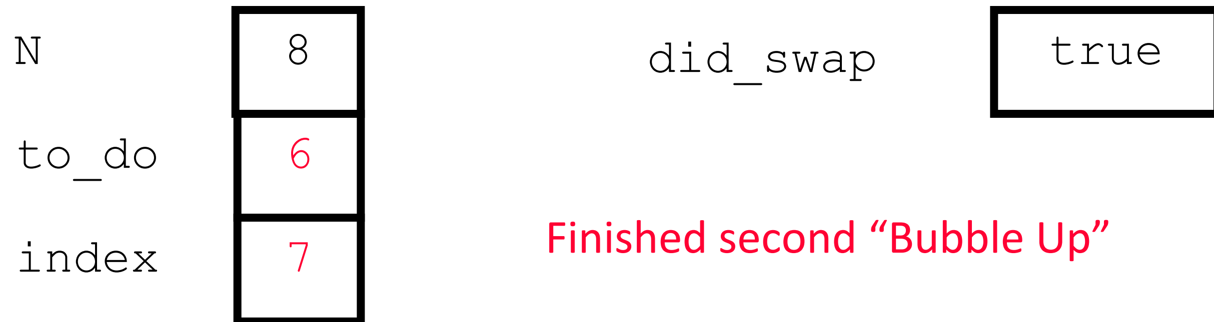
The Second "Bubble Up"

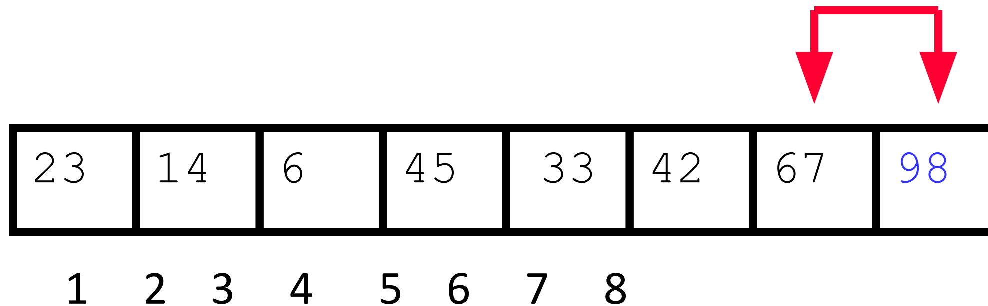


The Second "Bubble Up"

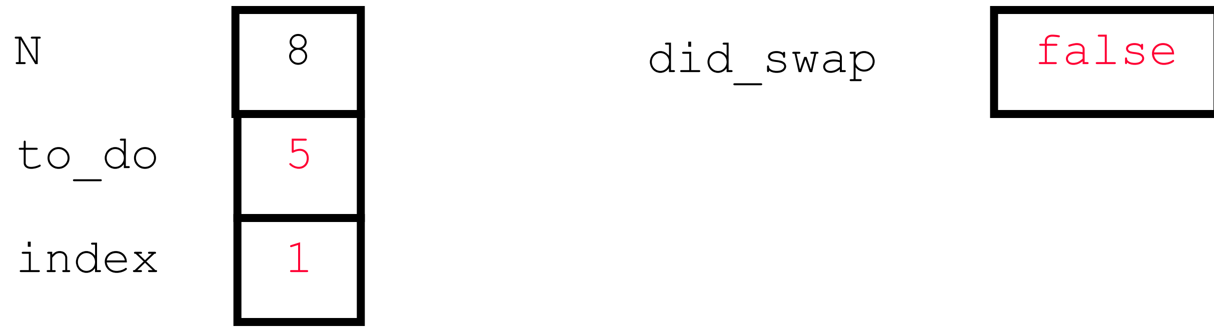


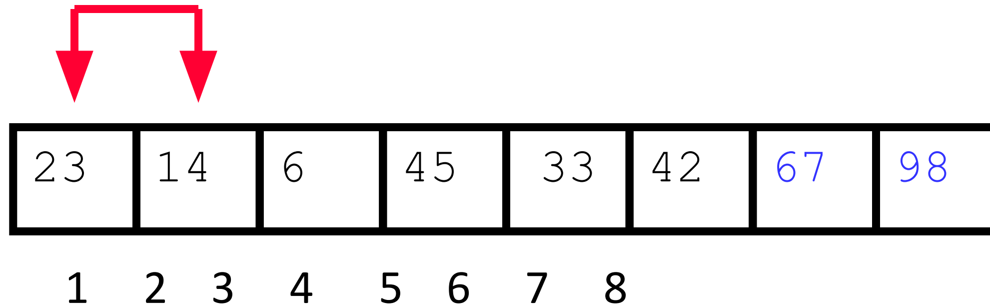
After Second Pass of Outer Loop



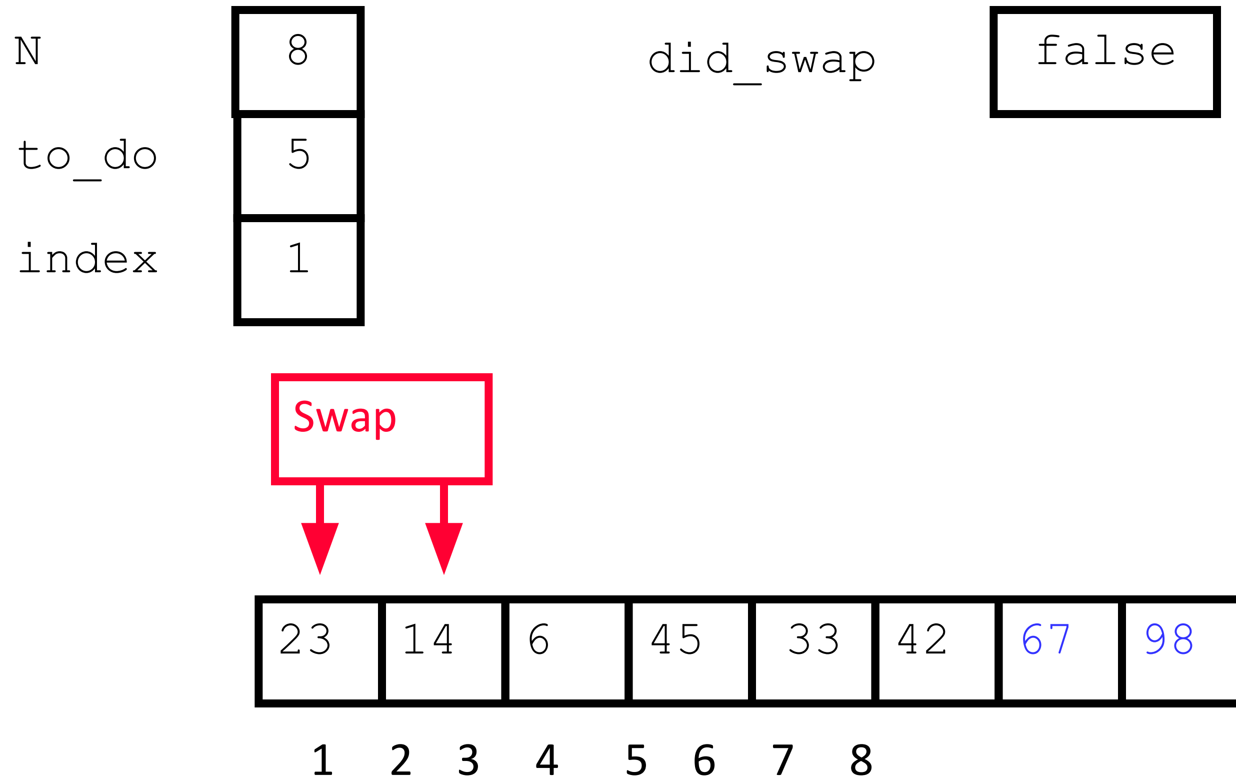


The Third "Bubble Up"

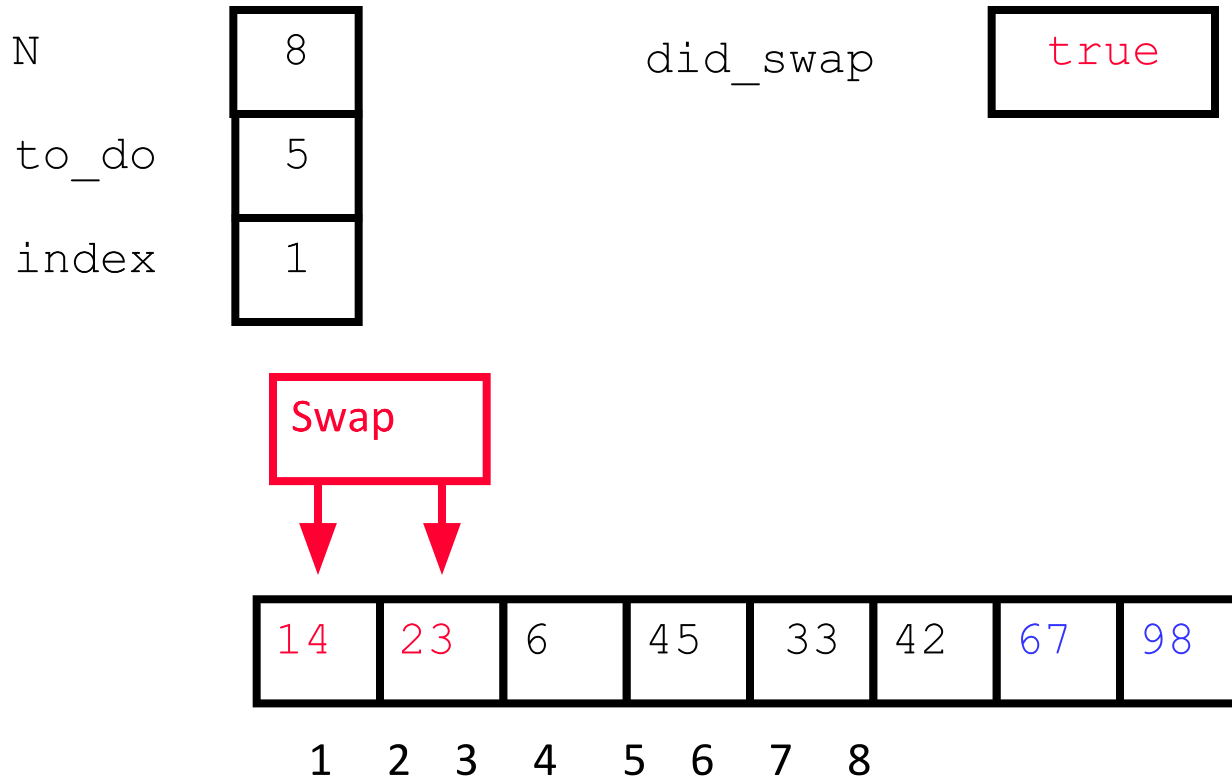




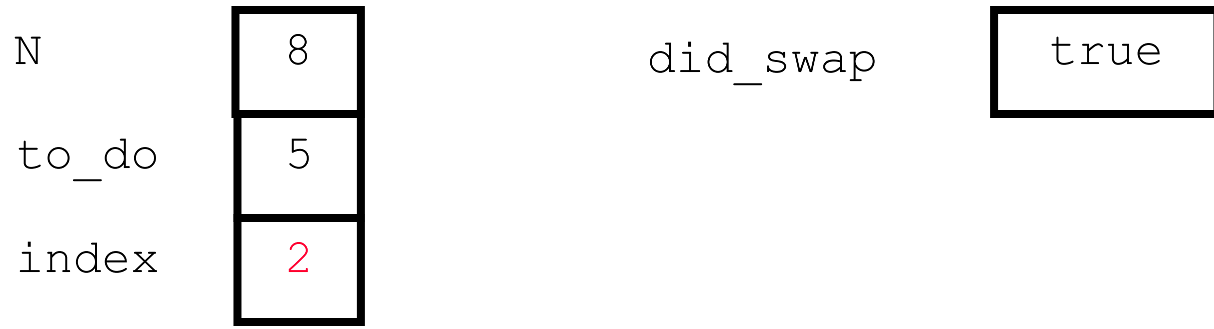
The Third "Bubble Up"

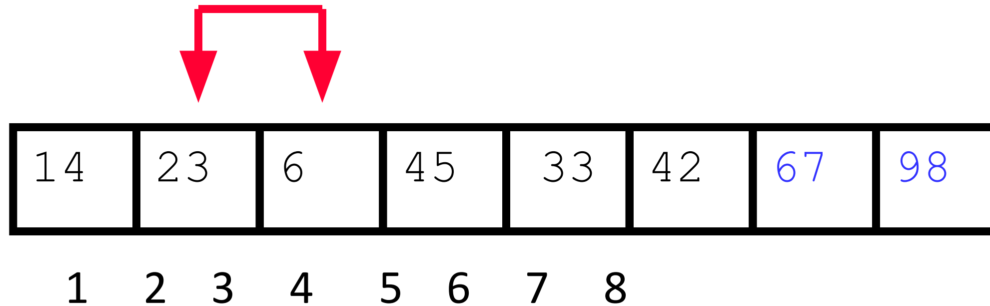


The Third "Bubble Up"

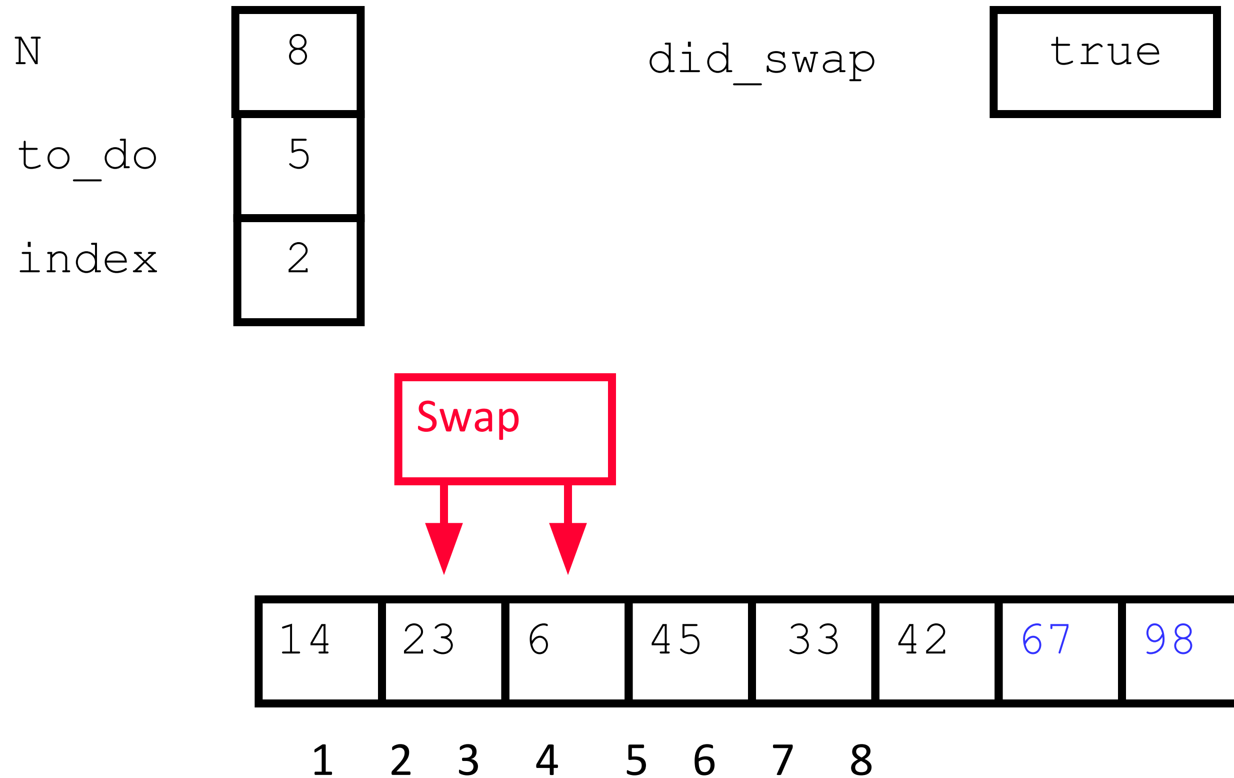


The Third "Bubble Up"

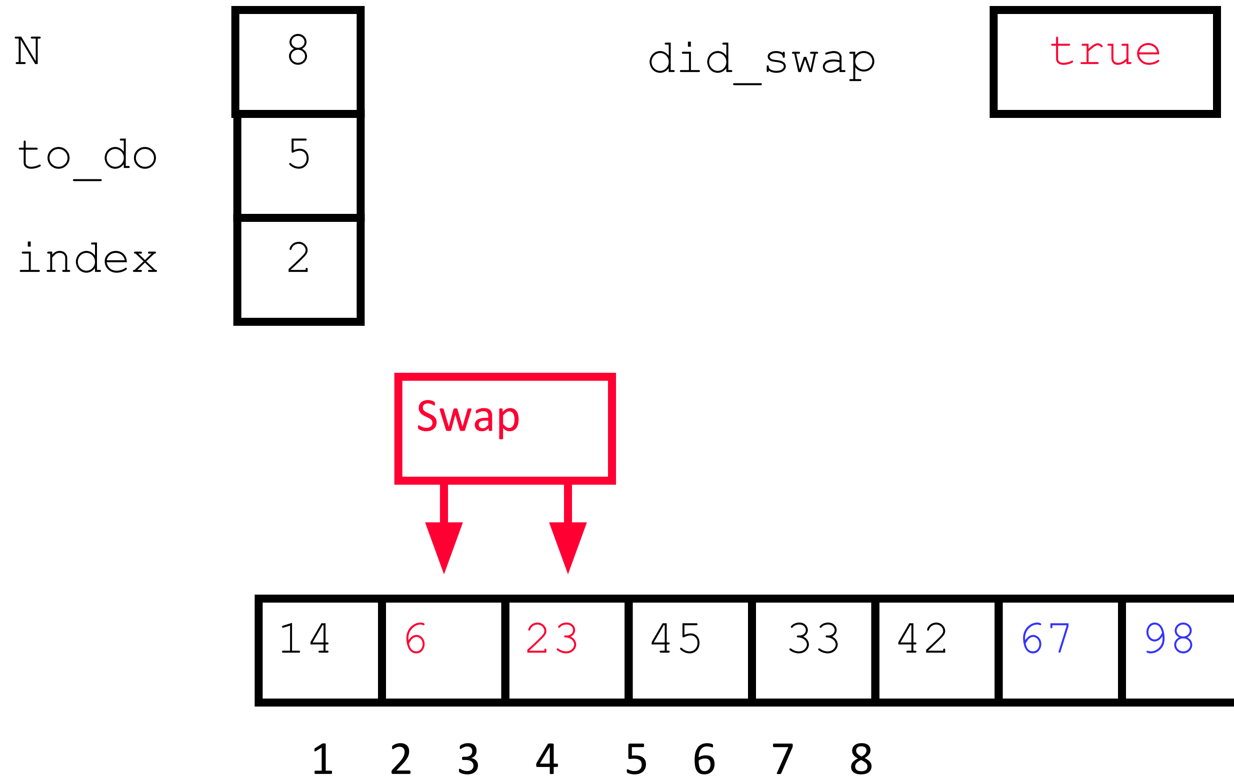




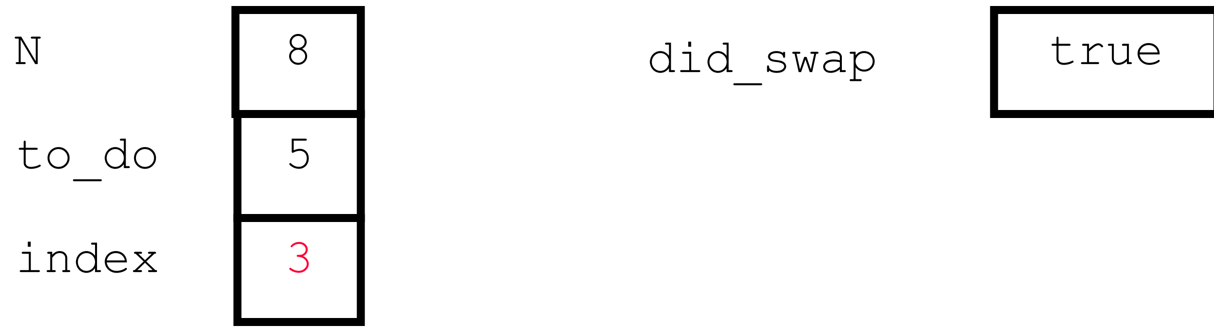
The Third "Bubble Up"

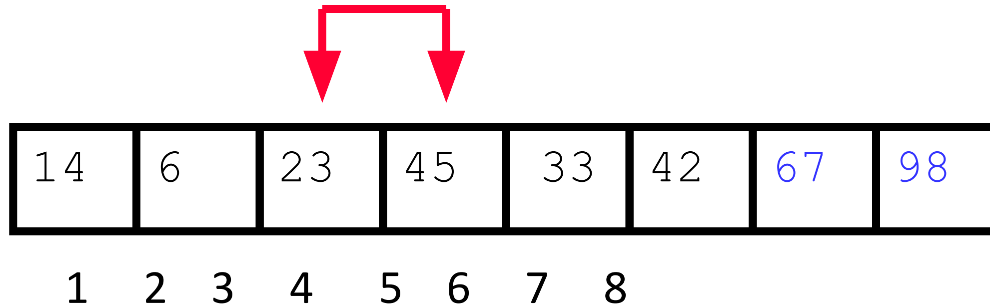


The Third "Bubble Up"

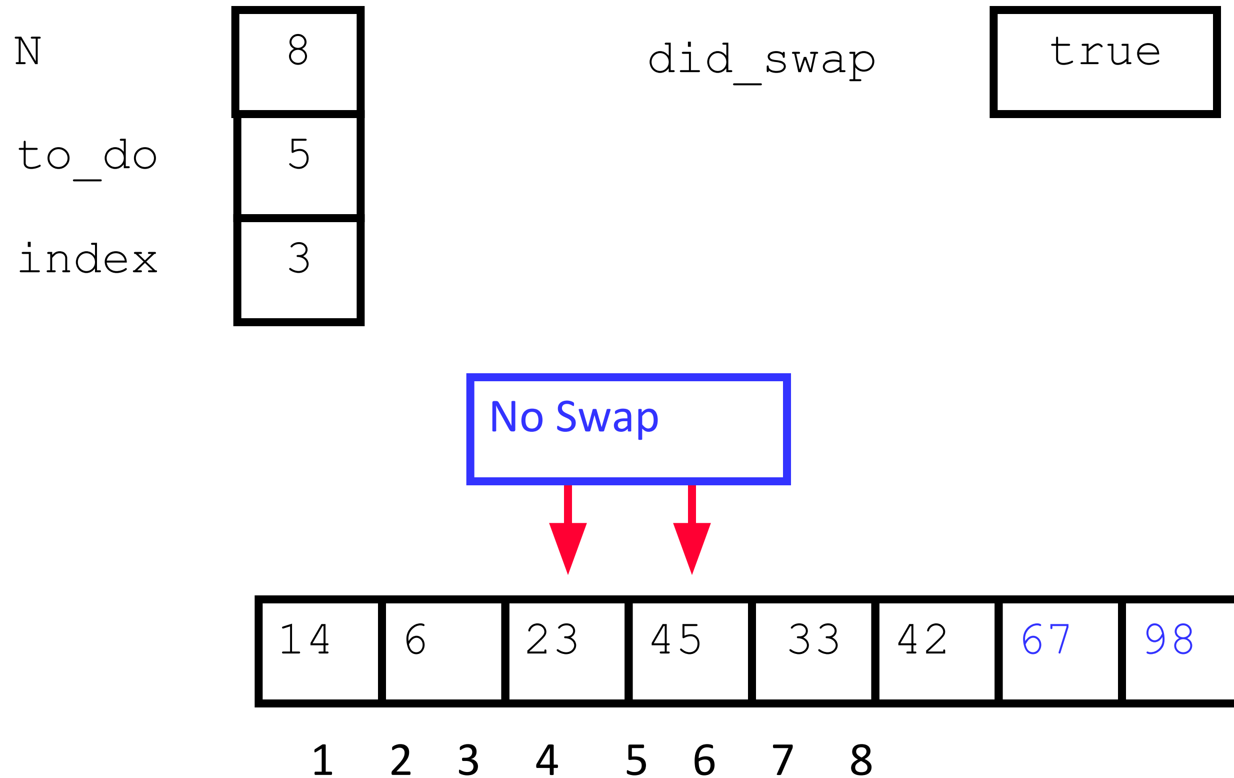


The Third "Bubble Up"

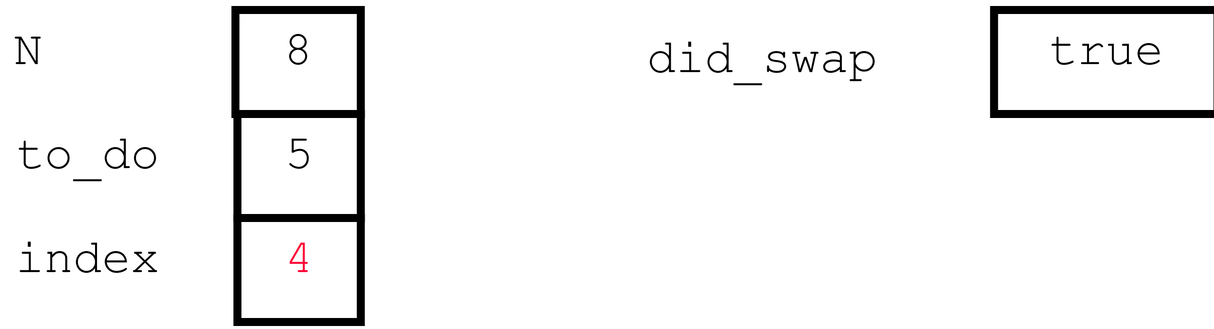


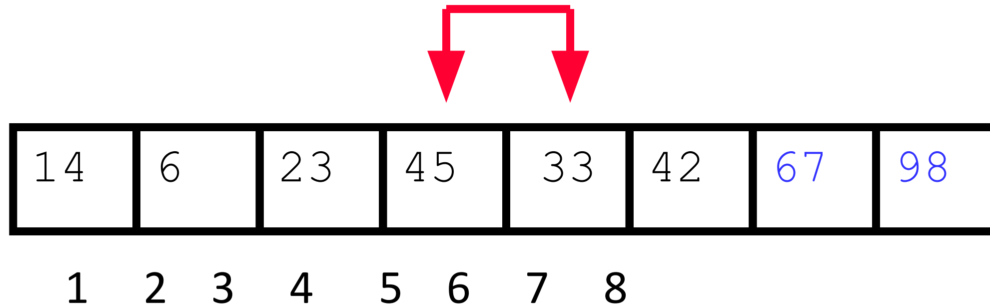


The Third "Bubble Up"

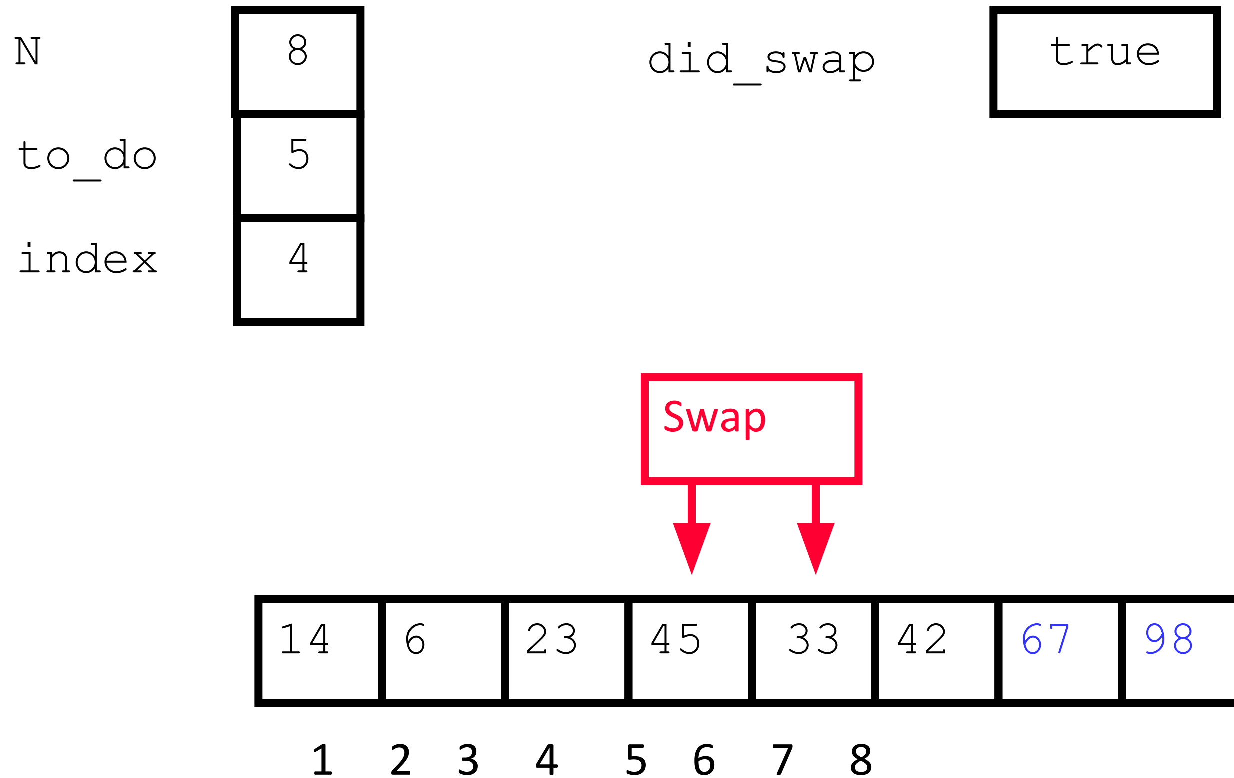


The Third "Bubble Up"

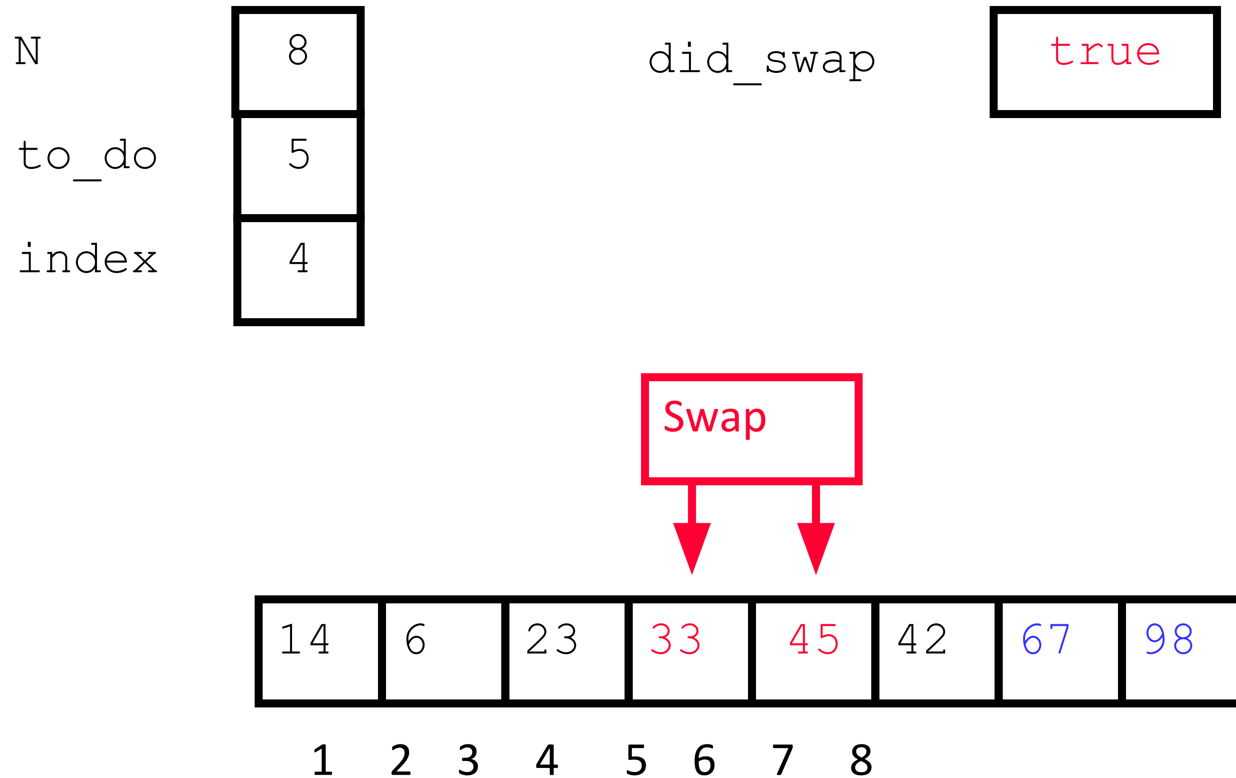




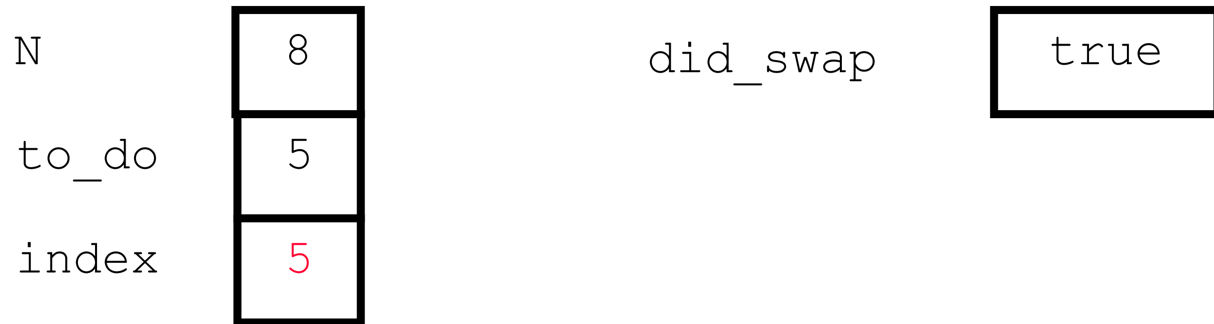
The Third "Bubble Up"

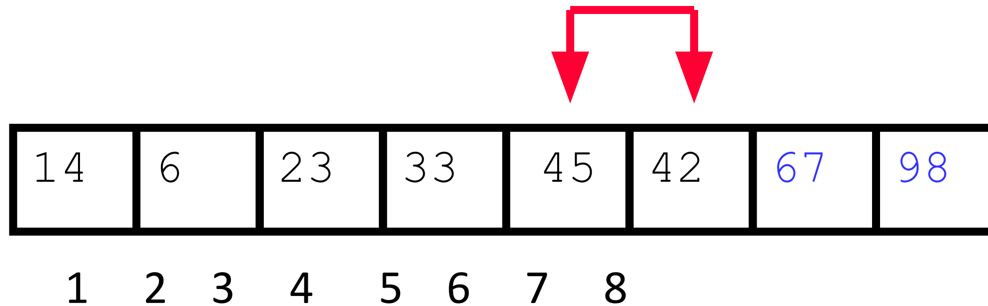


The Third "Bubble Up"

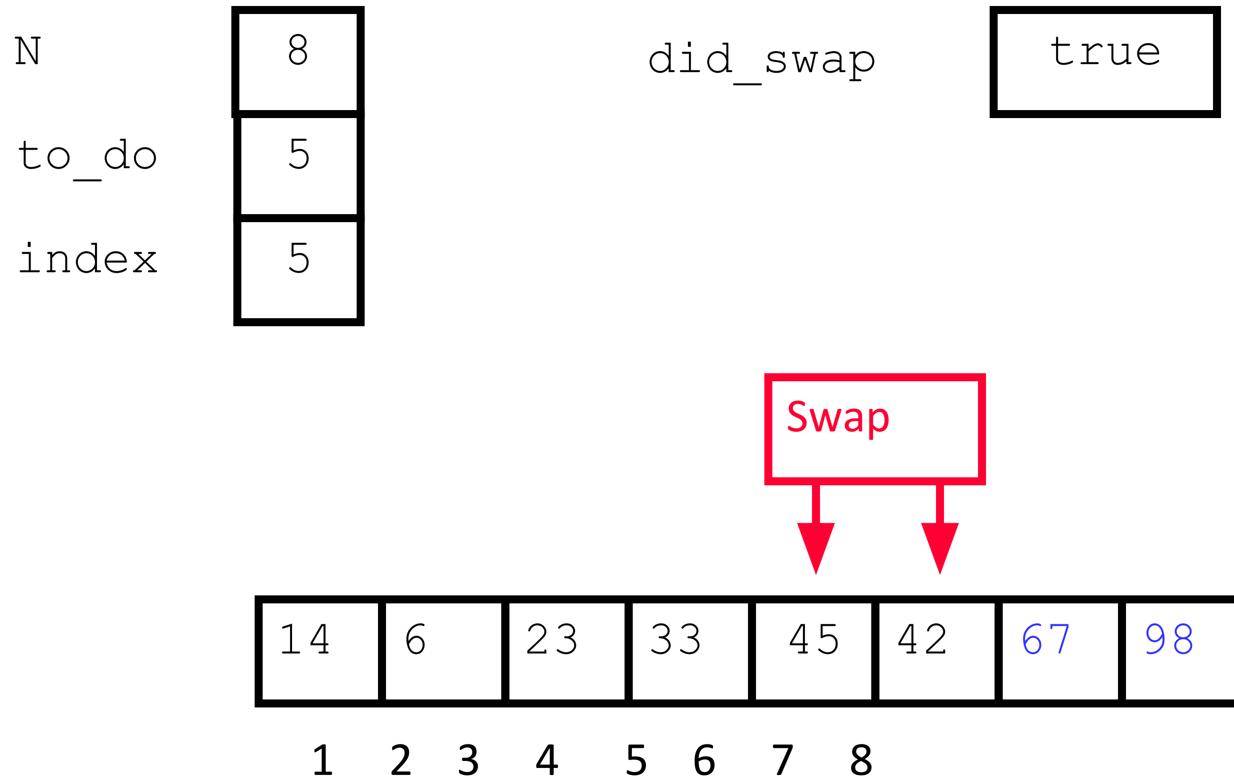


The Third "Bubble Up"

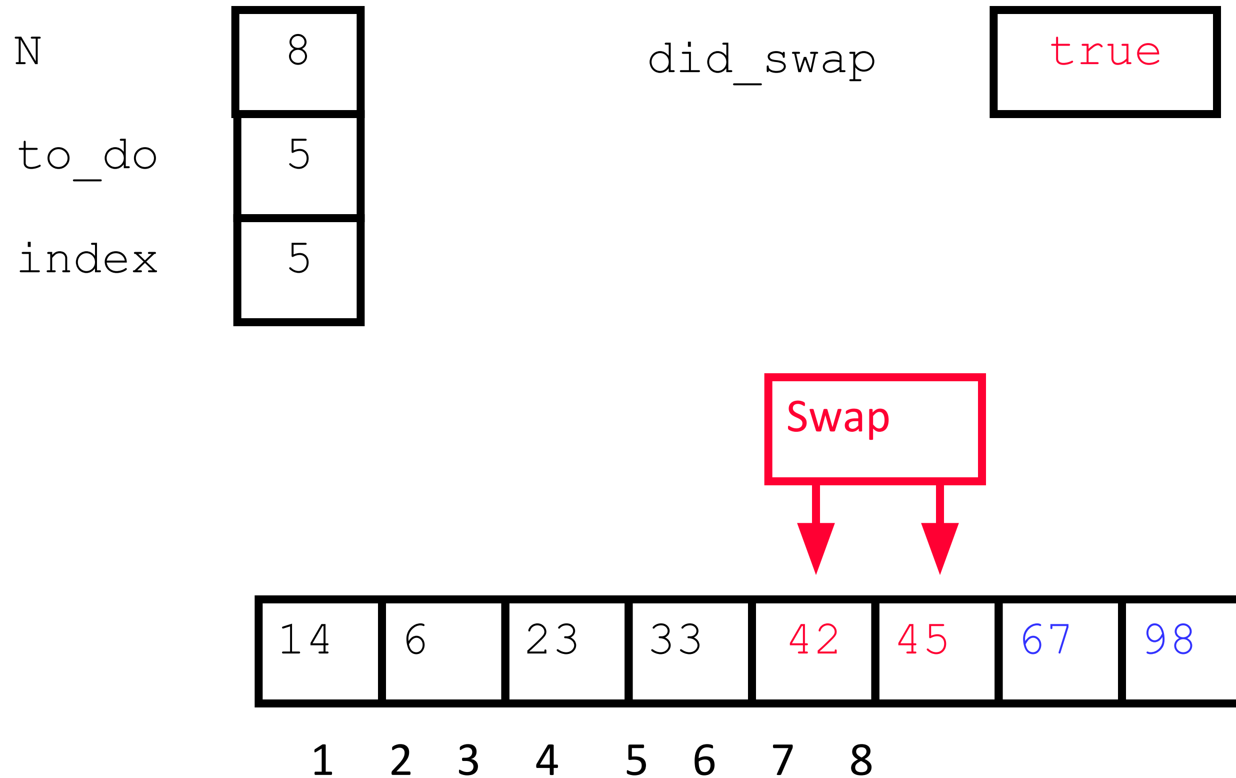




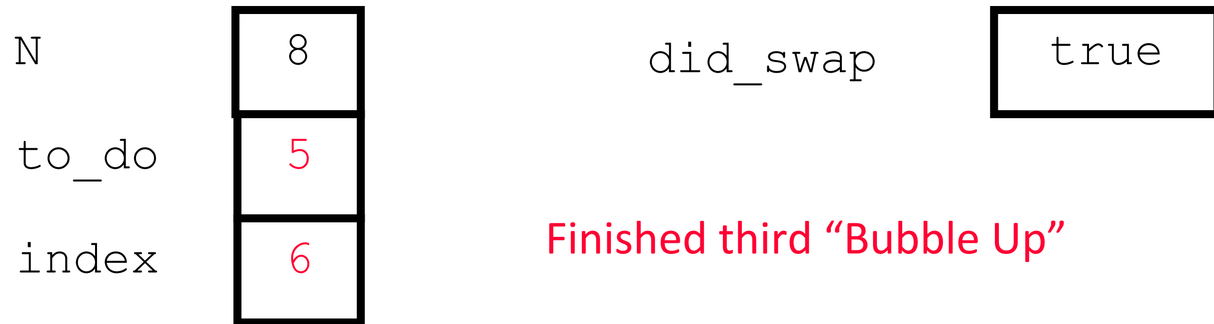
The Third "Bubble Up"

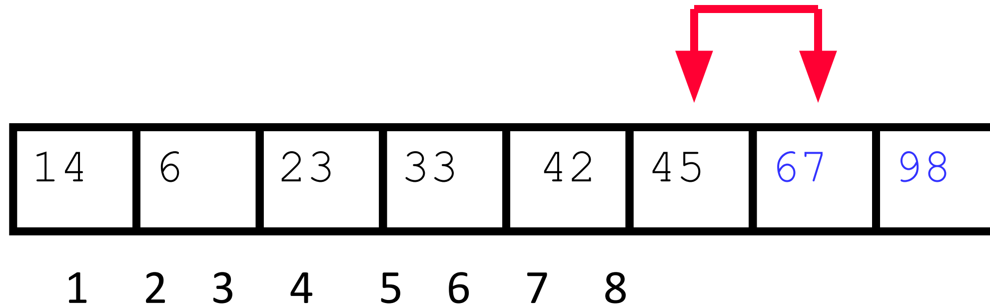


The Third "Bubble Up"

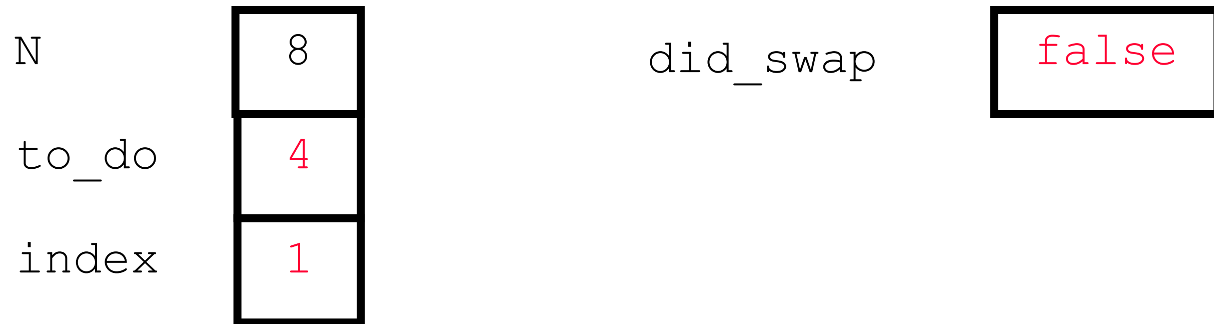


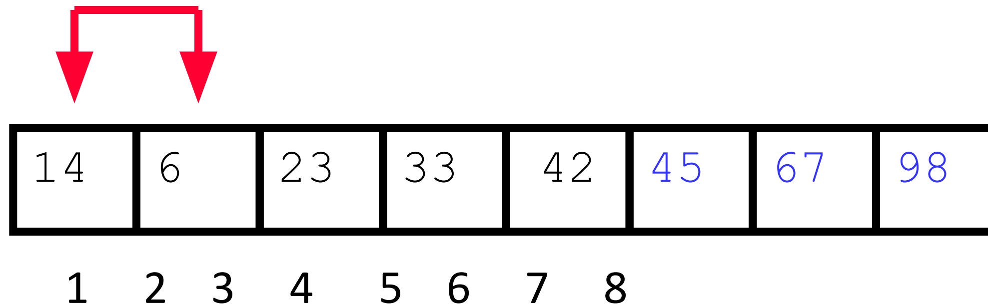
After Third Pass of Outer Loop





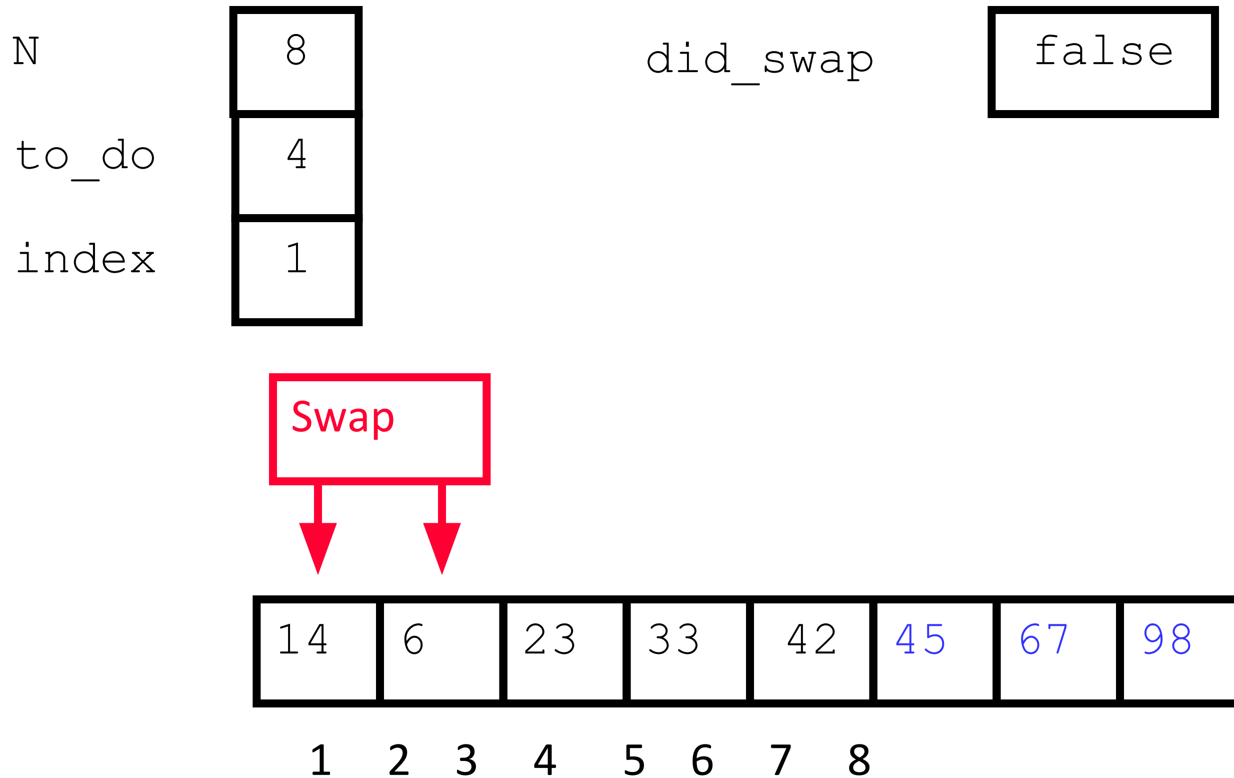
The Fourth "Bubble Up"



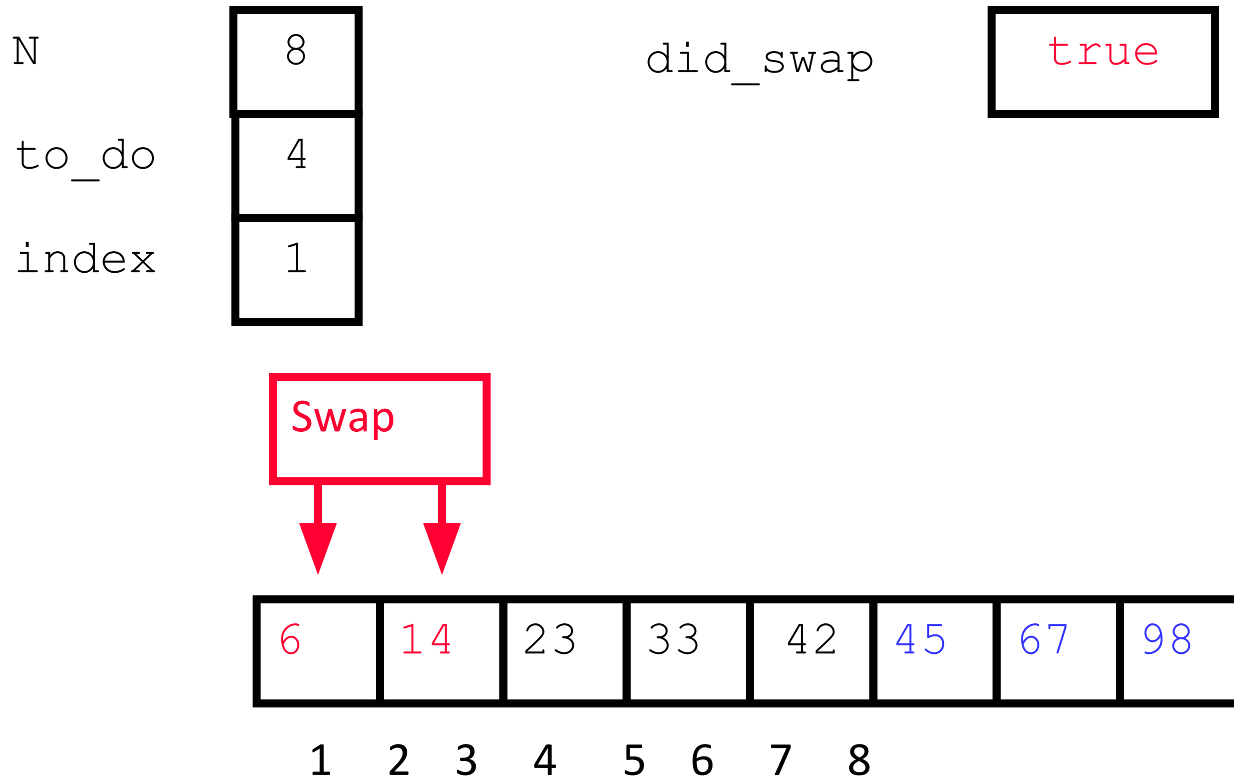


1 2 3 4 5 6 7 8

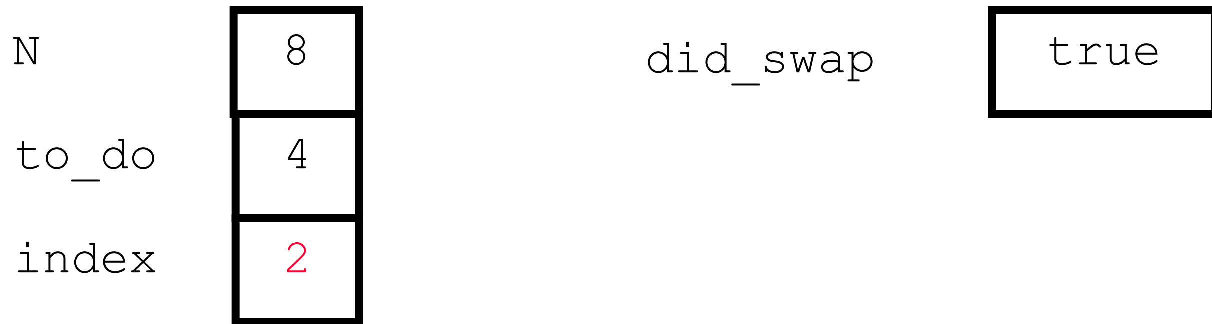
The Fourth "Bubble Up"

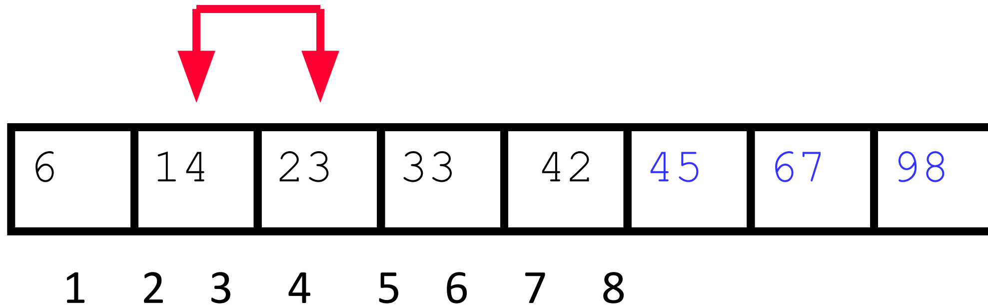


The Fourth "Bubble Up"

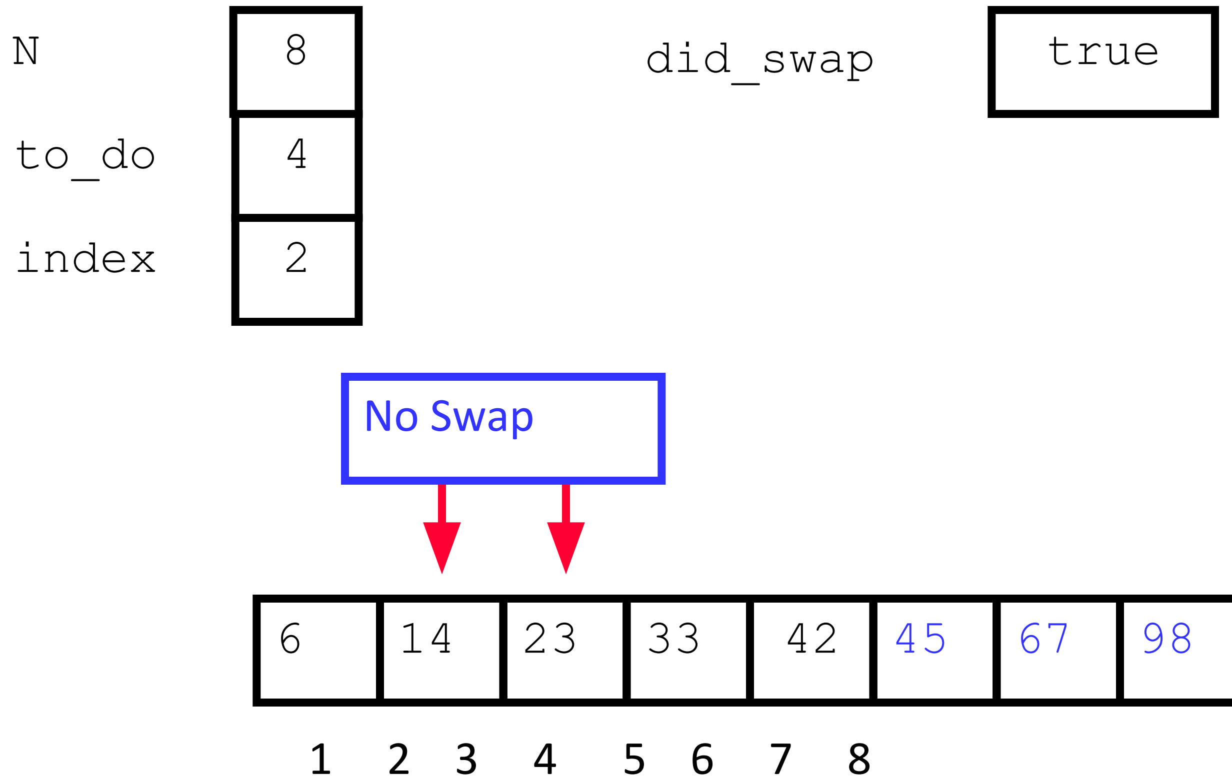


The Fourth "Bubble Up"

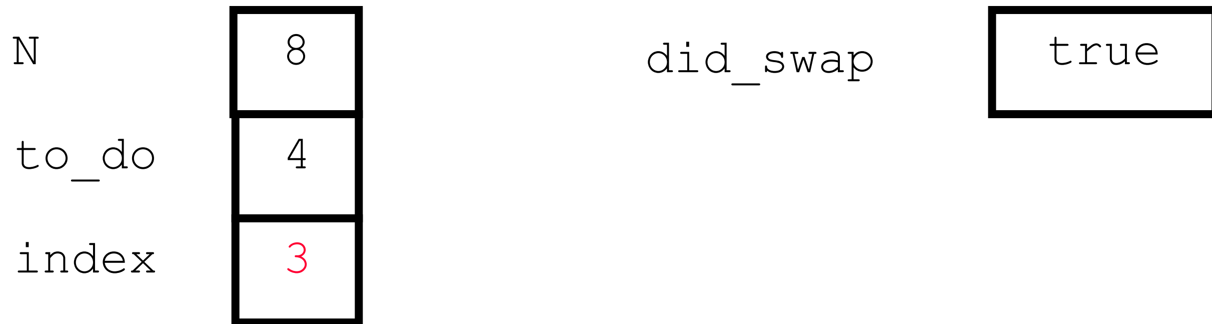


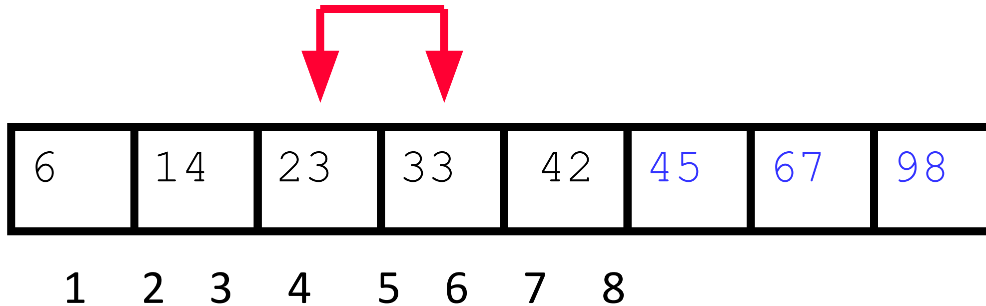


The Fourth "Bubble Up"

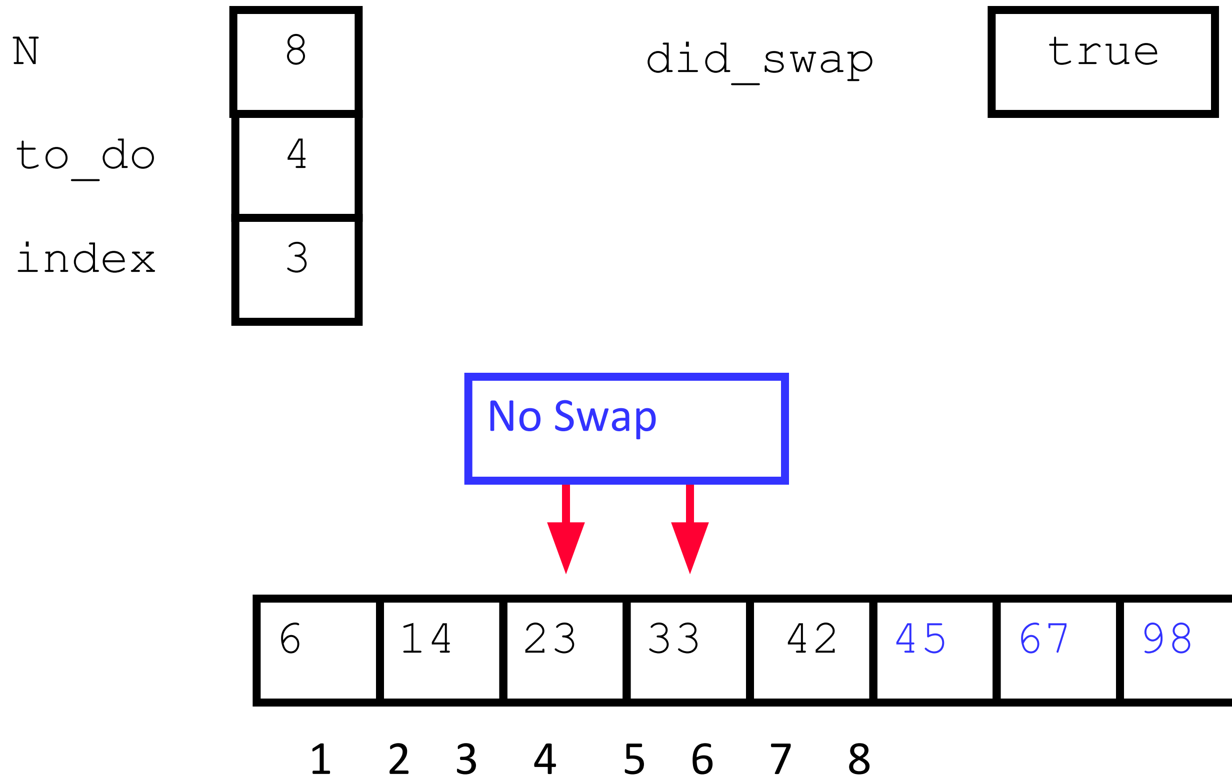


The Fourth “Bubble Up”

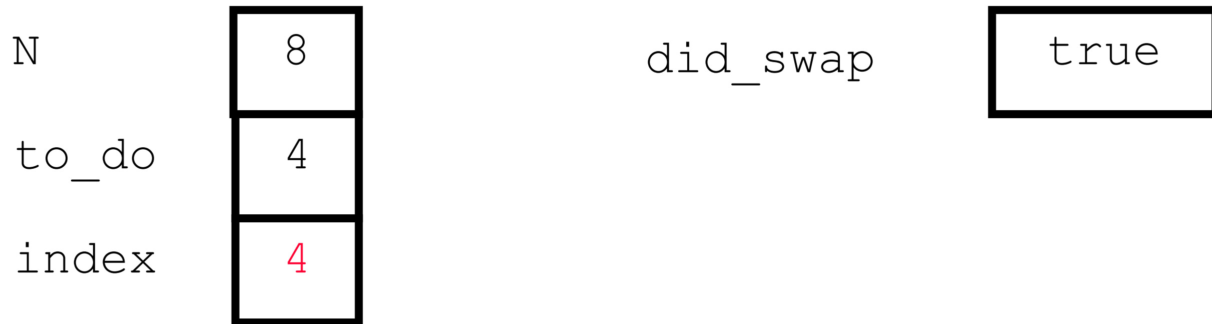


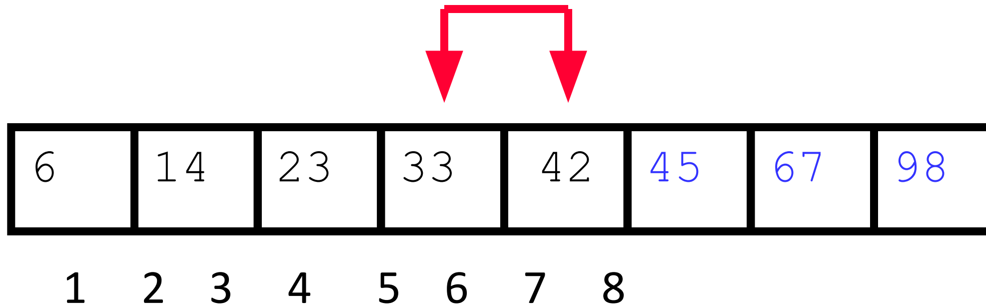


The Fourth "Bubble Up"

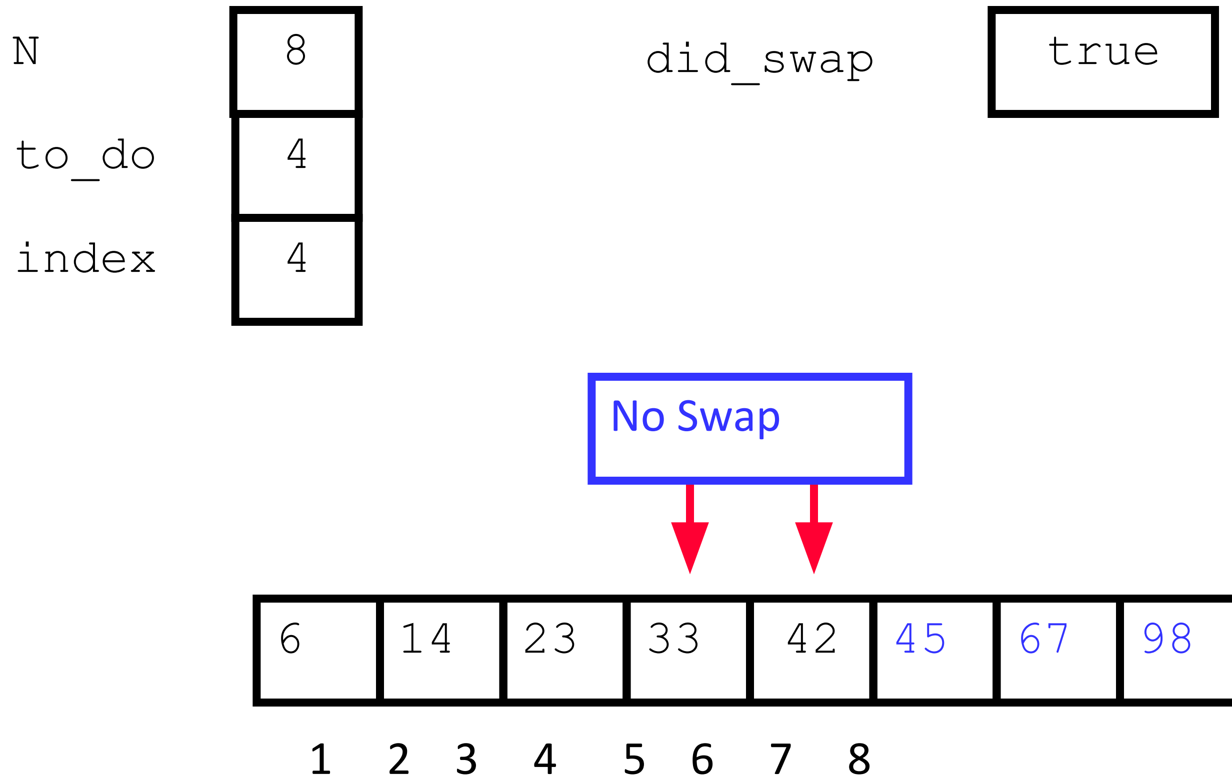


The Fourth "Bubble Up"

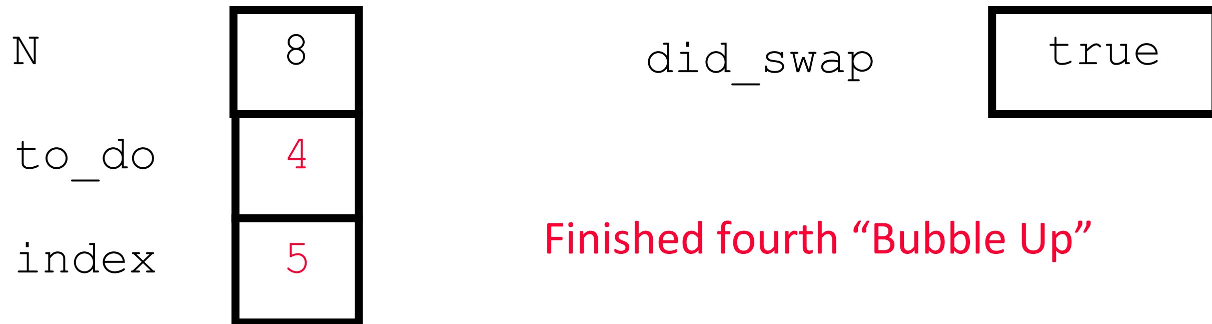


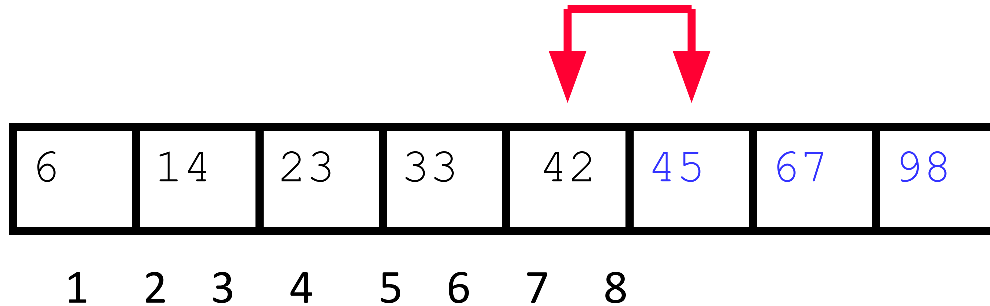


The Fourth "Bubble Up"

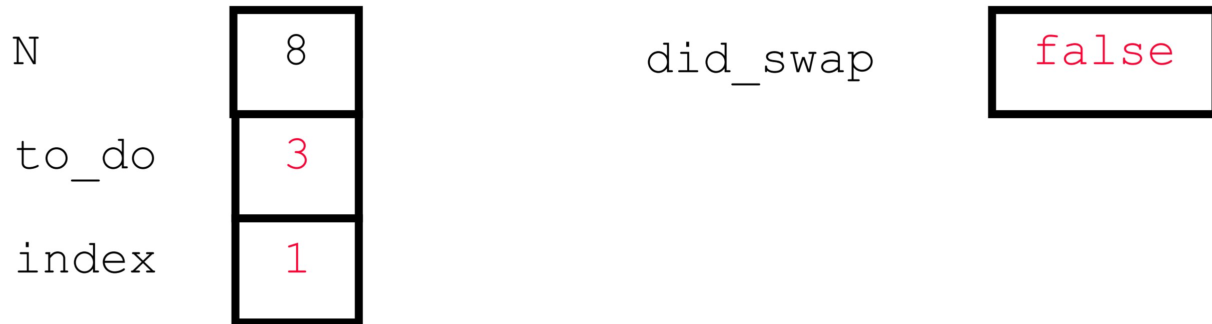


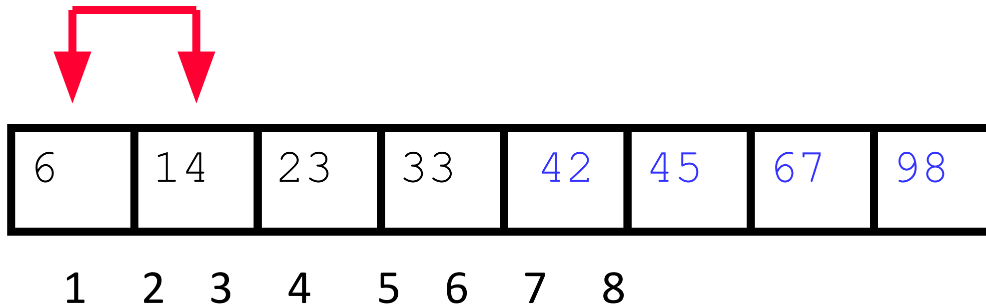
After Fourth Pass of Outer Loop



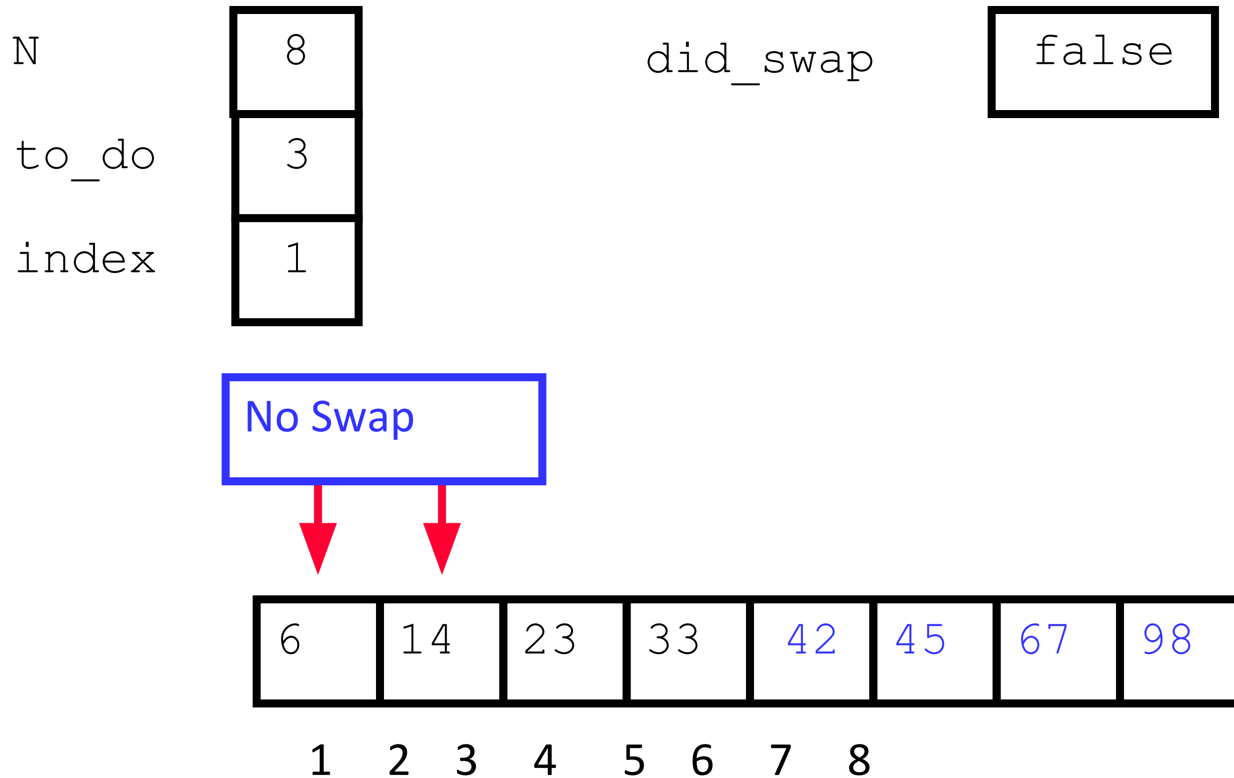


The Fifth “Bubble Up”

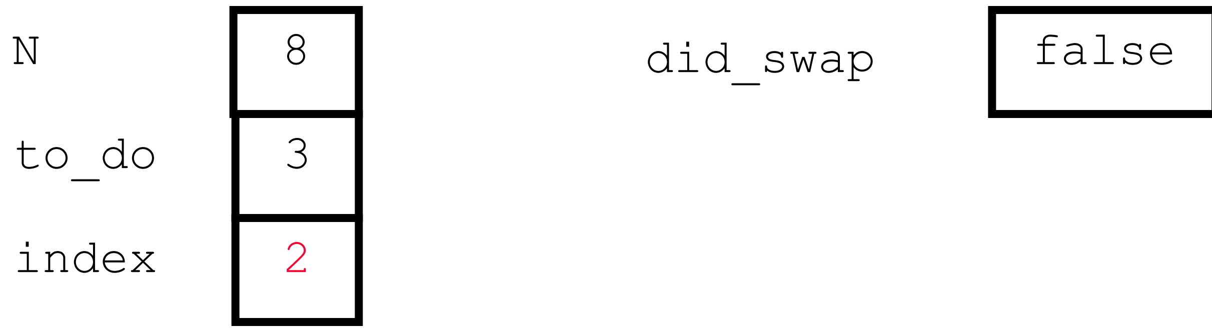


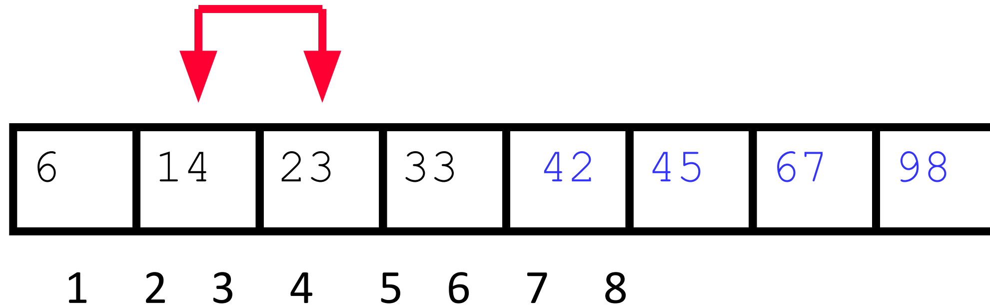


The Fifth "Bubble Up"

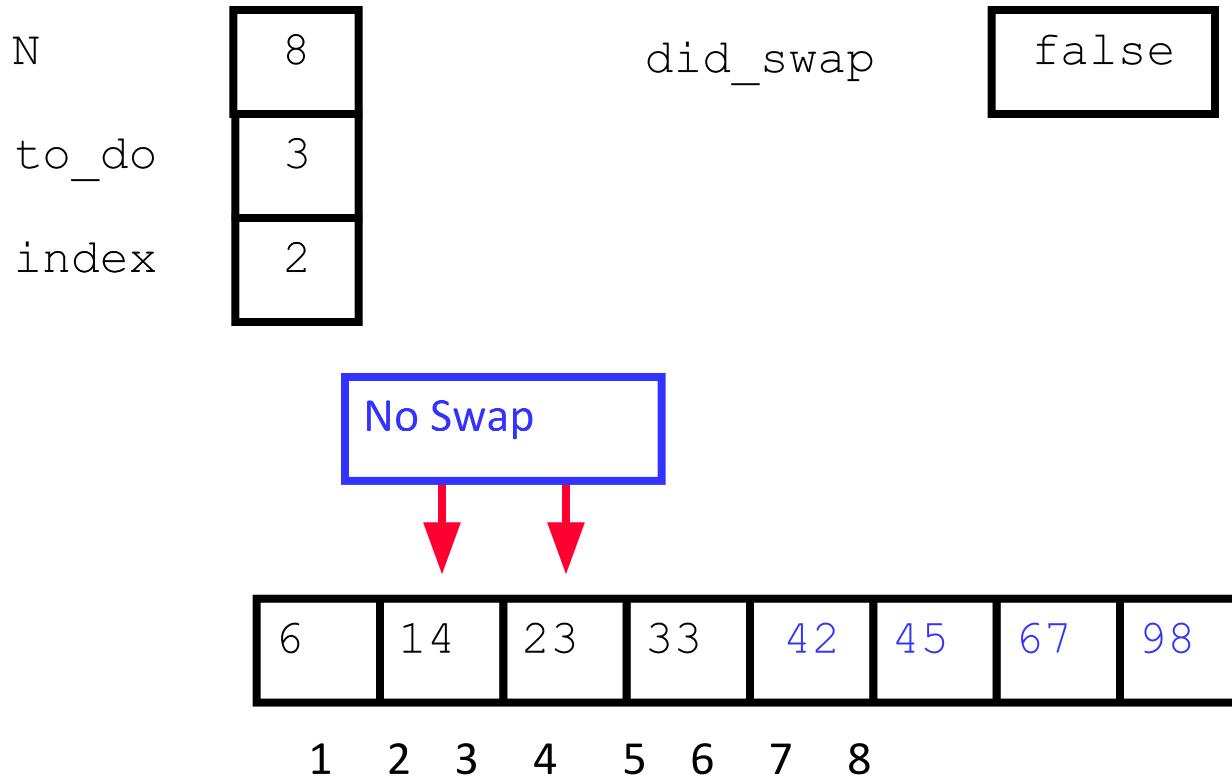


The Fifth "Bubble Up"

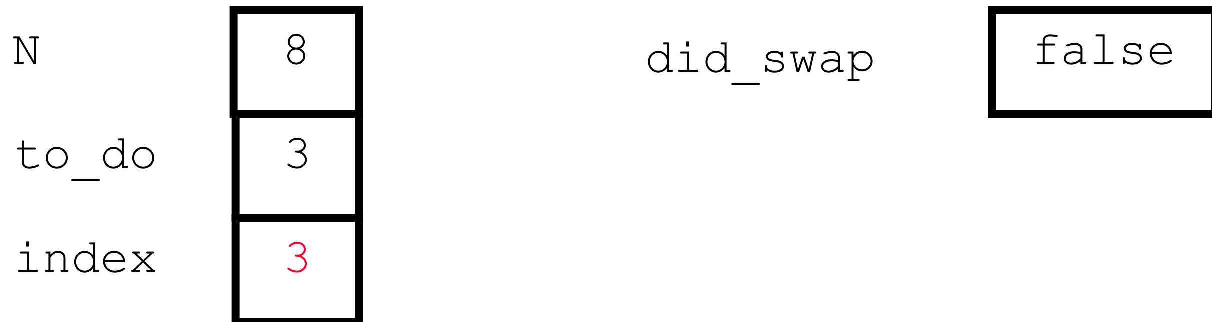


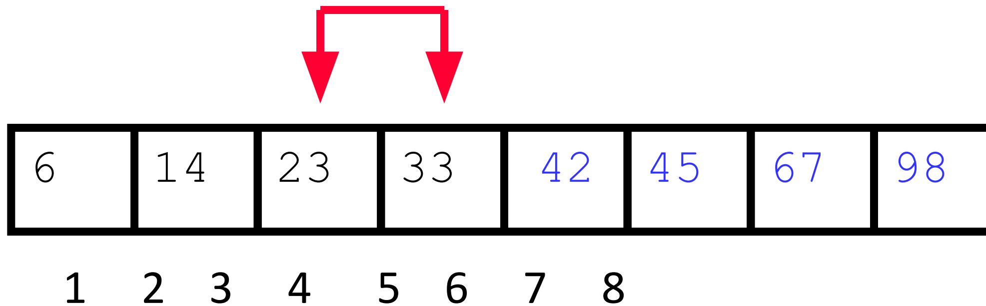


The Fifth "Bubble Up"

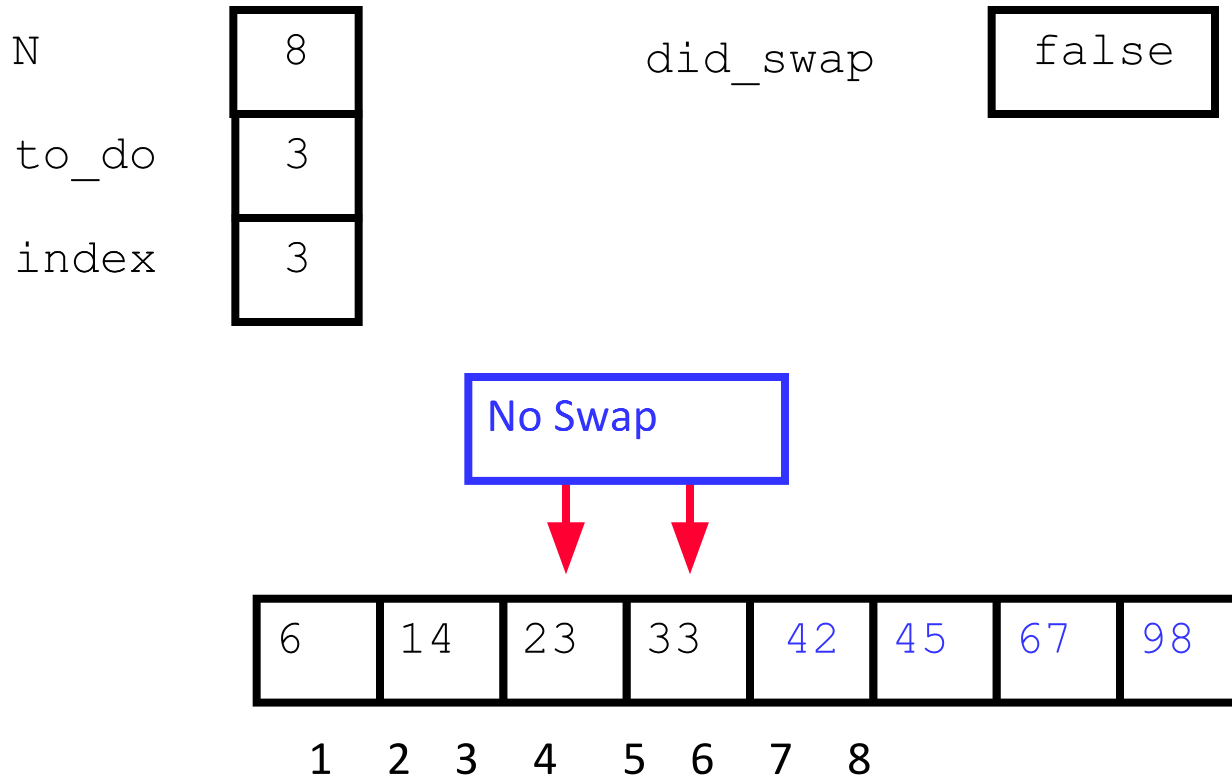


The Fifth "Bubble Up"

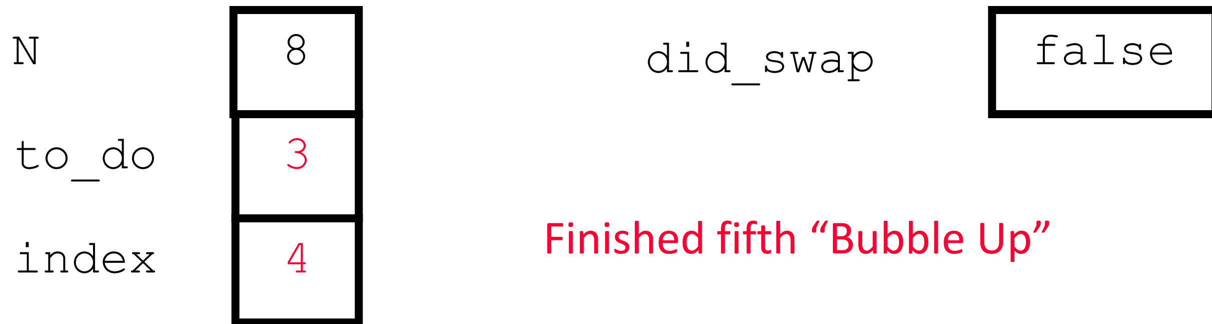


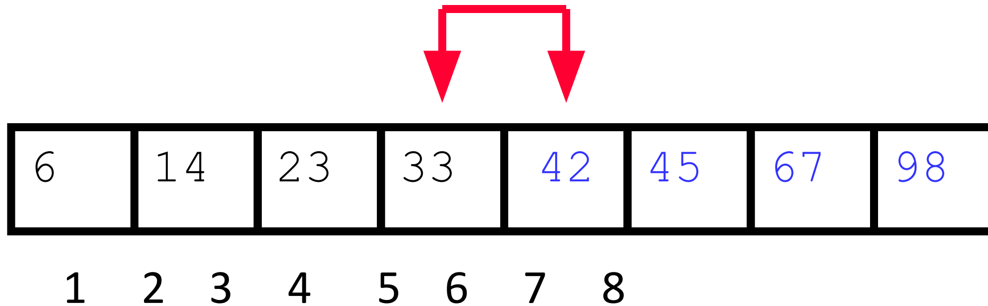


The Fifth "Bubble Up"

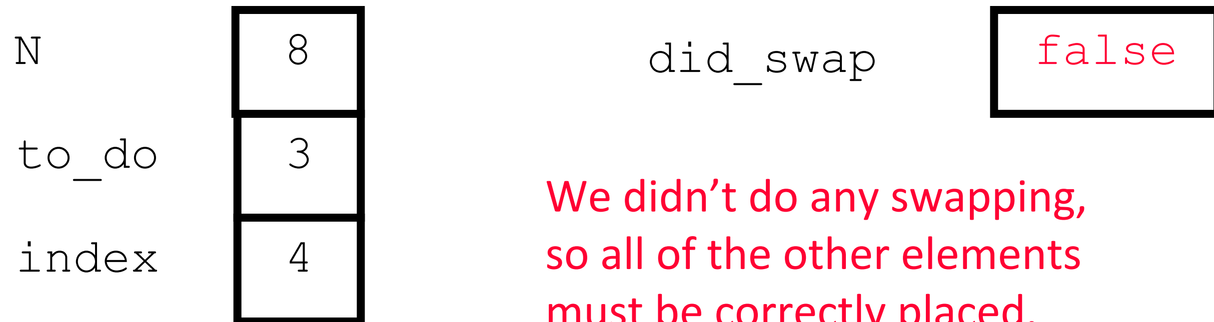


After Fifth Pass of Outer Loop

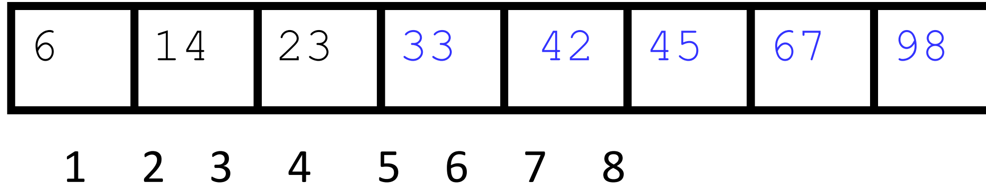




Finished "Early"



We can "skip" the last two passes of the outer loop.



Summary

- **"Bubble Up" algorithm will move largest value to its correct location (to the right)**
- **Repeat "Bubble Up" until all elements are correctly placed:**
  - **Maximum of N-1 times**
  - **Can finish early if no swapping occurs**
- **We reduce the number of elements we compare each time one is correctly placed**

- Merge Sort

Sorting

- **Sorting takes an unordered collection and makes it an ordered one.**

1	2	3	4	5	6
77	42	35	12	101	5

1	2	3	4	5	6
5	12	35	42	77	101

Divide and Conquer

- **Divide and Conquer cuts the problem in half each time, but uses the result of both halves:**
  - **cut the problem in half until the problem is trivial**
  - **solve for both halves**
  - **combine the solutions**

Mergesort

- **A divide-and-conquer algorithm:**
- **Divide the unsorted array into 2 halves until the sub-arrays only contain one element**
- **Merge the sub-problem solutions together:**
  - **Compare the sub-array's first elements**
  - **Remove the smallest element and put it into the result array**
  - **Continue the process until all elements have been put into the result array**

37	23	6	89	15	12	2	19

Algorithm

```
Mergesort(Passed an array)
  if array size > 1
    Divide array in half
    Call Mergesort on first half.
    Call Mergesort on second half.
    Merge two halves.

Merge(Passed two arrays)
  Compare leading element in each array
  Select lower and place in new array.
    (If one input array is empty then place
     remainder of other array in output array)
```

98	23	45	14	6	67	33	42

98	23	45	14	6	67	33	42

98	23	45	14

6	67	33	42

98	23	45	14	6	67	33	42

98	23	45	14

6	67	33	42

98	23

45	14

98	23	45	14	6	67	33	42

98	23	45	14

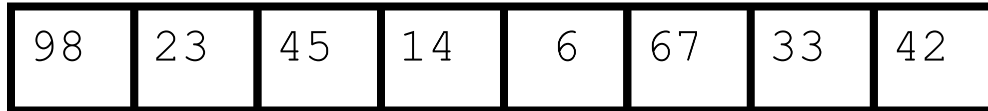
6	67	33	42

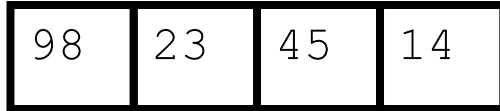
98	23

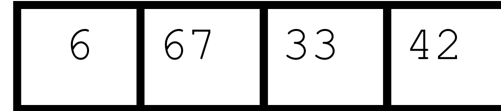
45	14

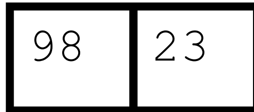
98

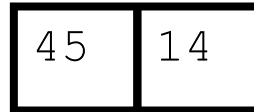
23

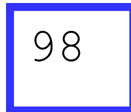


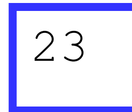




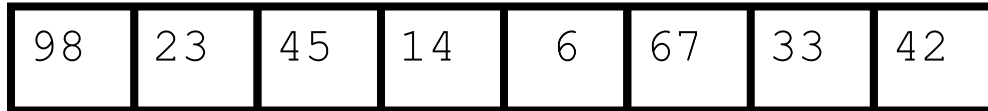


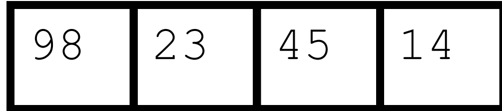


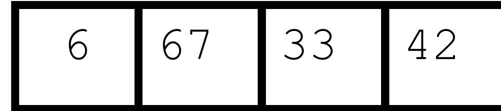


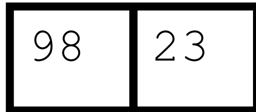


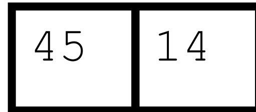
Merge

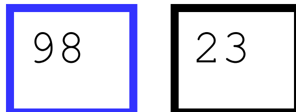


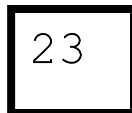




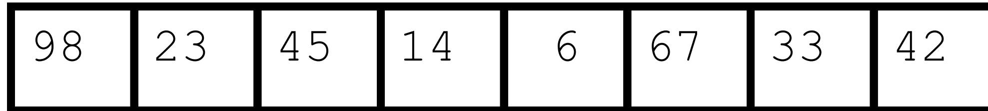


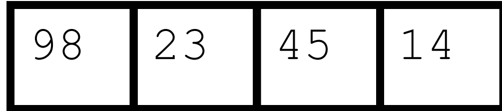


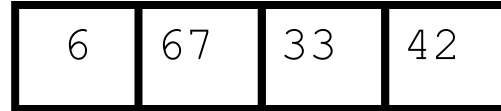


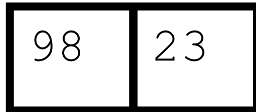


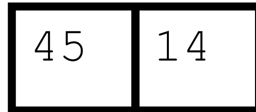
Merge

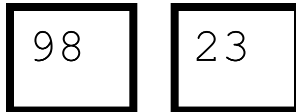


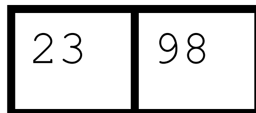












Merge

98	23	45	14	6	67	33	42

98	23	45	14

6	67	33	42

98	23

45	14

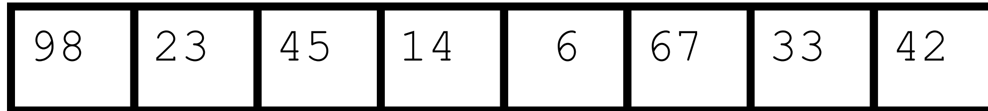
98

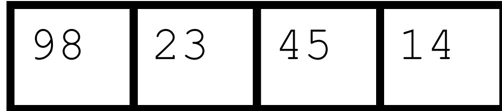
23

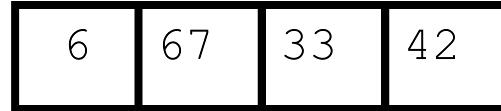
45

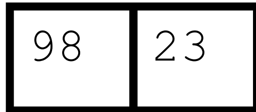
14

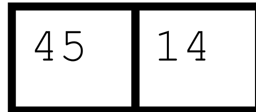
23	98

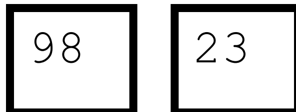


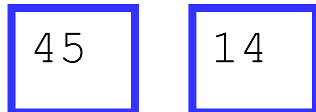


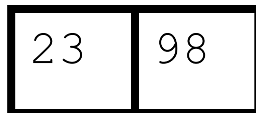




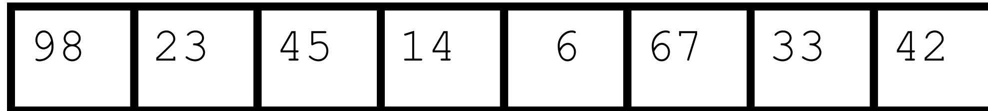


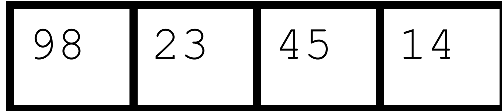


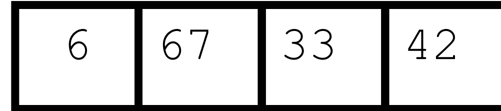


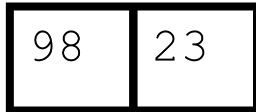


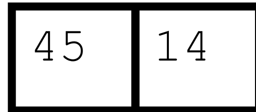
Merge

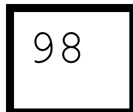


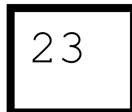


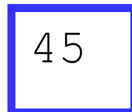


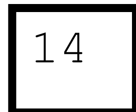


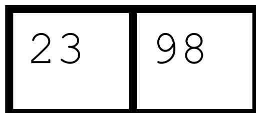


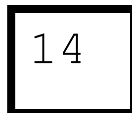




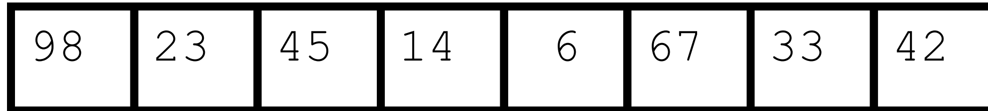


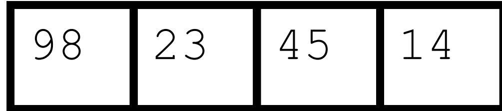


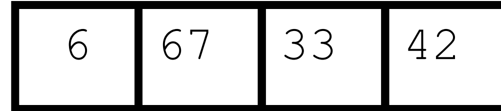


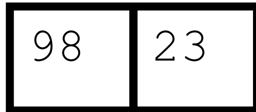


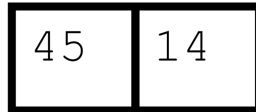
Merge

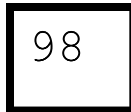


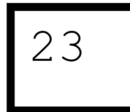


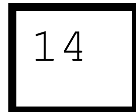


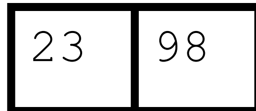


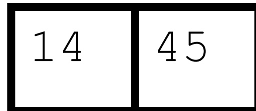




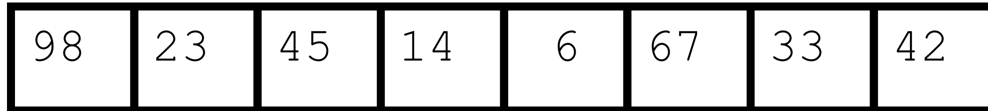


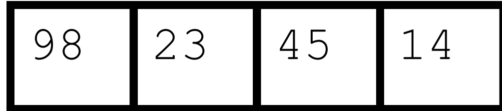


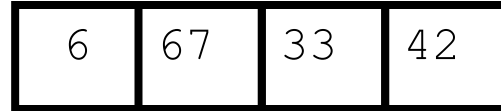


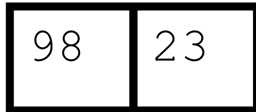


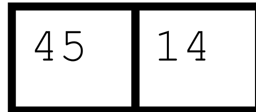
Merge

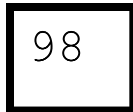


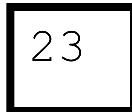


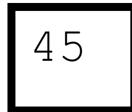


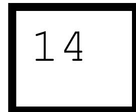


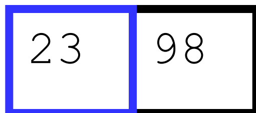


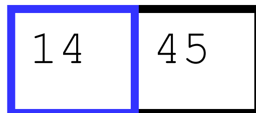




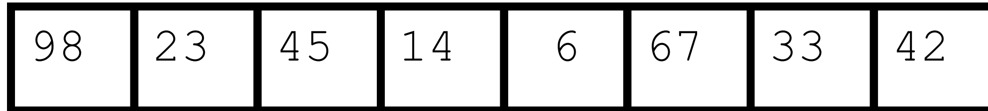


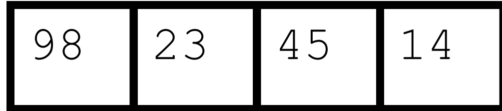


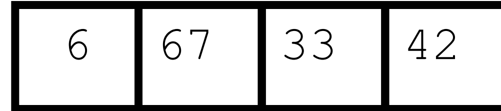


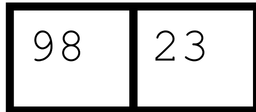


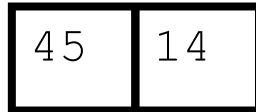
Merge

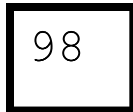

98
23
45
14
6
67
33
42

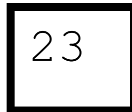

98
23
45
14

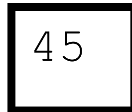

6
67
33
42

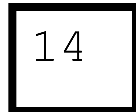

98
23

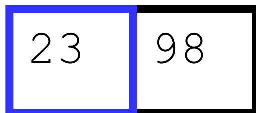

45
14

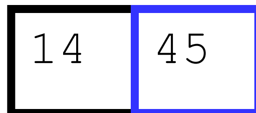

98

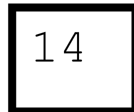

23


45

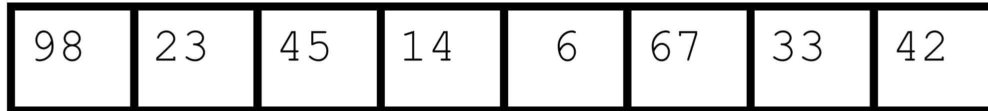
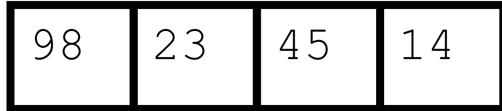
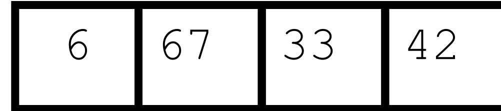
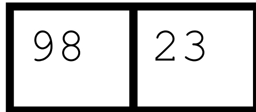
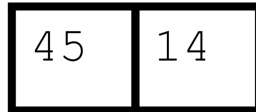
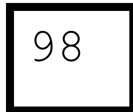
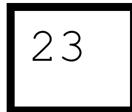
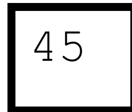
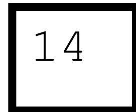
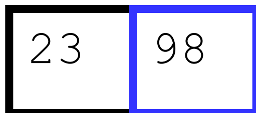
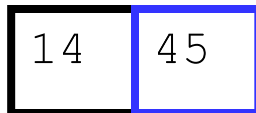
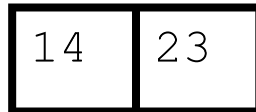

14


23
98

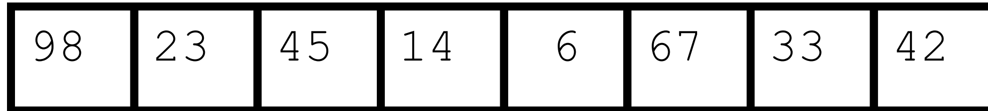
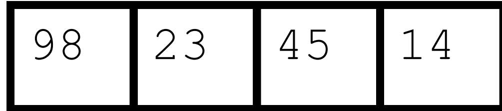
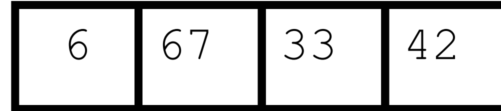
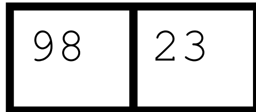
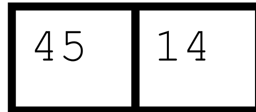
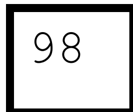
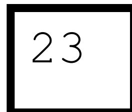

14
45

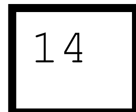
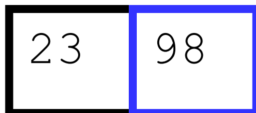
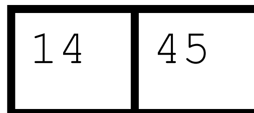
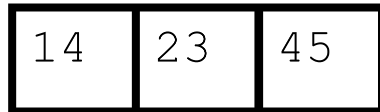

14

Merge

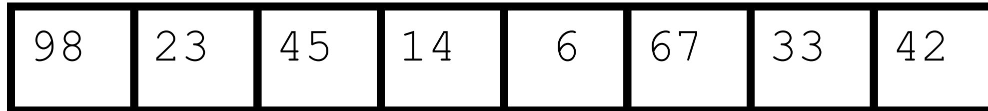
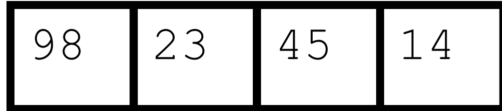
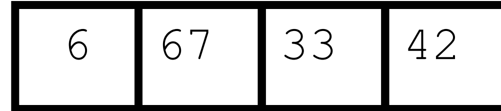
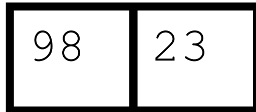
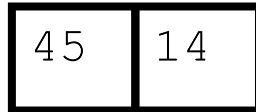
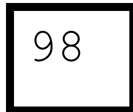
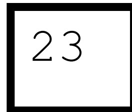
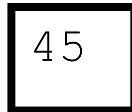
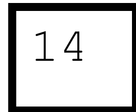
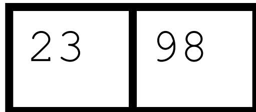
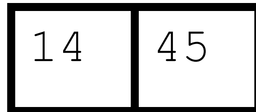
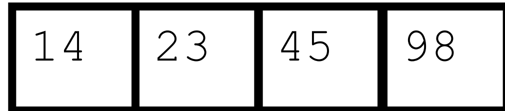

98
23
45
14
6
67
33
42

98
23
45
14

6
67
33
42

98
23

45
14

98

23

45

14

23
98

14
45

14
23

Merge


98
23
45
14
6
67
33
42

98
23
45
14

6
67
33
42

98
23

45
14

98

23

45

14

23
98

14
45

14
23
45

Merge


98 23 45 14 6 67 33 42

98 23 45 14

6 67 33 42

98 23

45 14

98

23

45

14

23 98

14 45

14 23 45 98

Merge

98	23	45	14	6	67	33	42

98	23	45	14

6	67	33	42

98	23

45	14

6	67

33	42

98

23

45

14

23	98

14	45

14	23	45	98

98 23 45 14 6 67 33 42

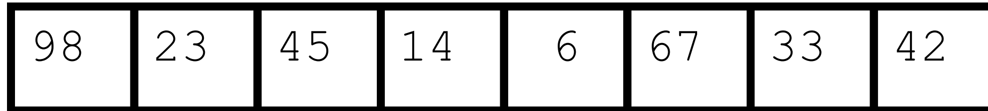
98 23 45 14 | 6 67 33 42

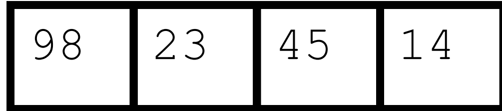
98 23 | 45 14 | 6 67 | 33 42

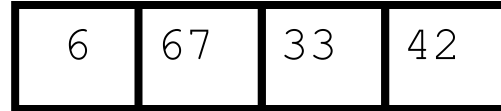
98 | 23 | 45 | 14 | 6 | 67

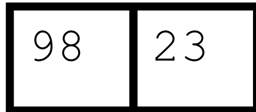
23 98 | 14 45

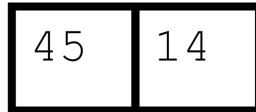
14 23 45 98

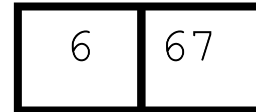

98 23 45 14 6 67 33 42

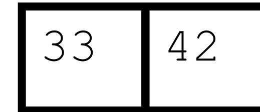

98 23 45 14

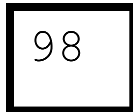

6 67 33 42

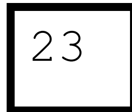

98 23

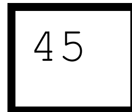

45 14

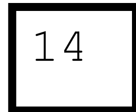

6 67

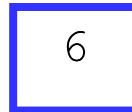

33 42

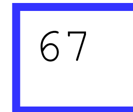

98

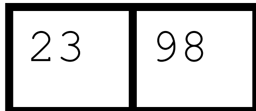

23

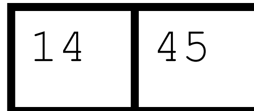

45

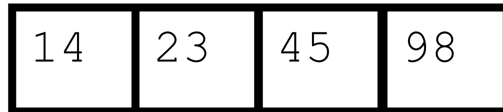

14

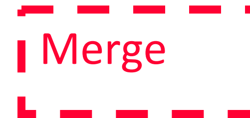

6

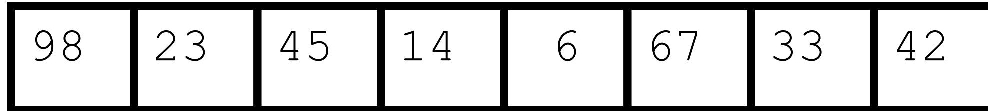
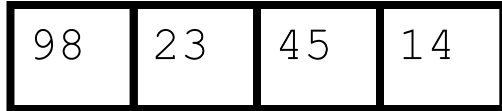
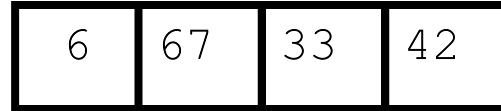
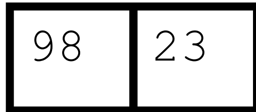
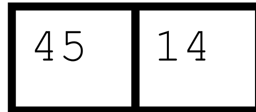
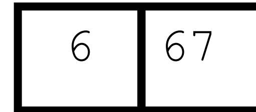
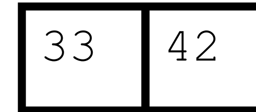
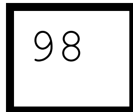
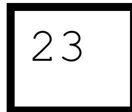
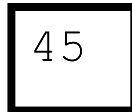
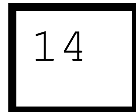
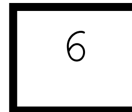
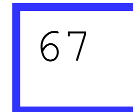
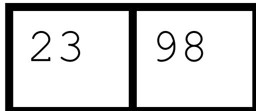
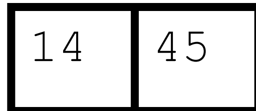
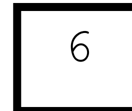
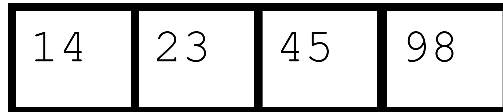
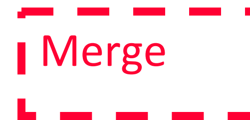

67

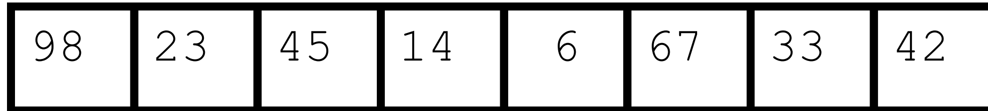
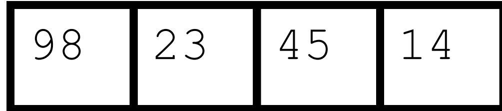
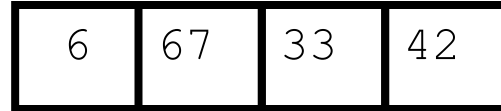
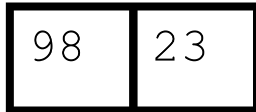
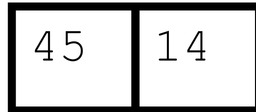
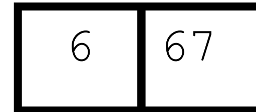
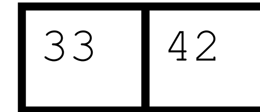
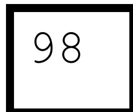
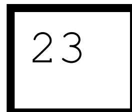

23 98

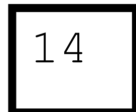
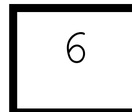
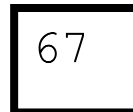
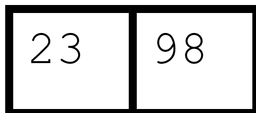
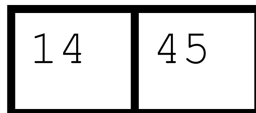
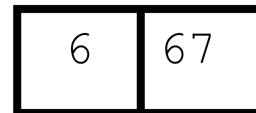
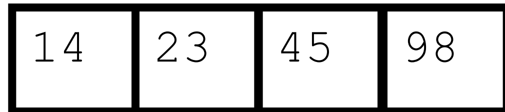
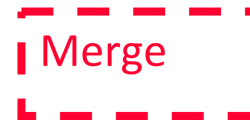

14 45


14 23 45 98


Merge


98 23 45 14 6 67 33 42

98 23 45 14

6 67 33 42

98 23

45 14

6 67

33 42

98

23

45

14

6

67

23 98

14 45

6

14 23 45 98

Merge


98 23 45 14 6 67 33 42

98 23 45 14

6 67 33 42

98 23

45 14

6 67

33 42

98

23

45

14

6

67

23 98

14 45

6 67

14 23 45 98

Merge

98	23	45	14	6	67	33	42

98	23	45	14

6	67	33	42

98	23

45	14

6	67

33	42

98

23

45

14

6

67

33

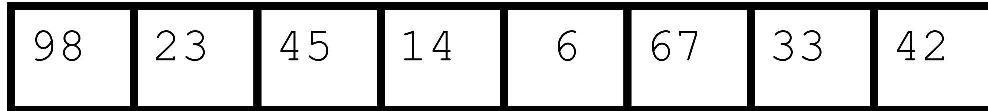
42

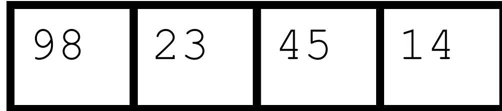
23	98

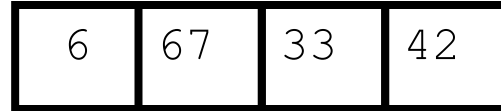
14	45

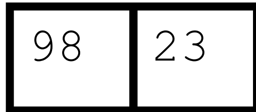
6	67

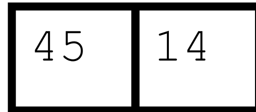
14	23	45	98

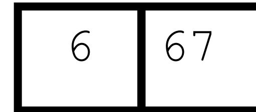


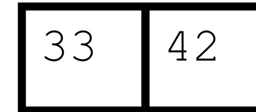


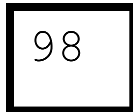


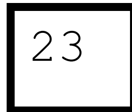


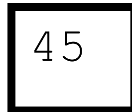


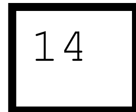


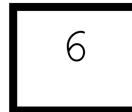


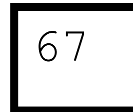


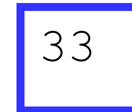


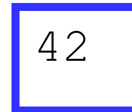


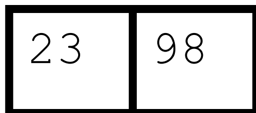


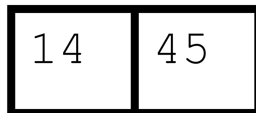


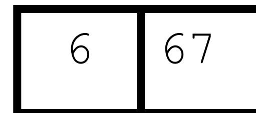


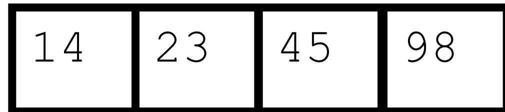




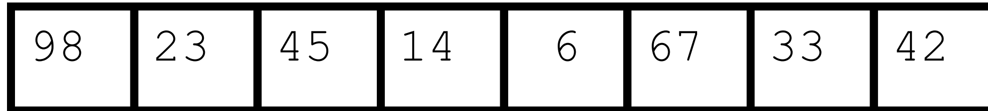


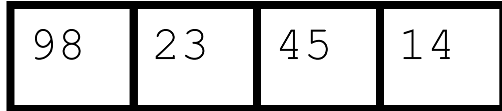


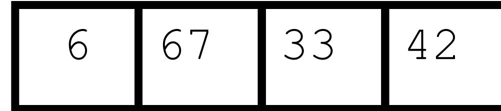


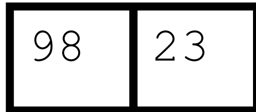


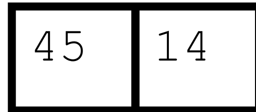
Merge

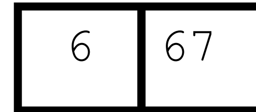


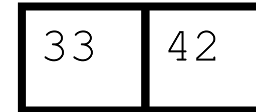


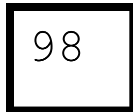


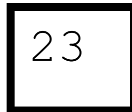


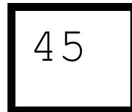


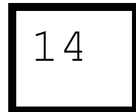


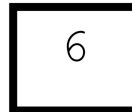


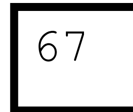


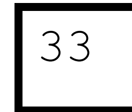


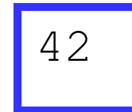


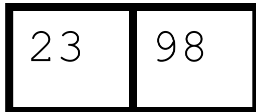


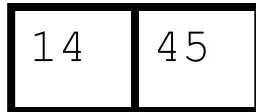


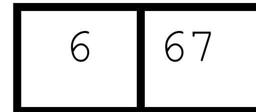


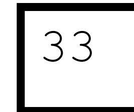


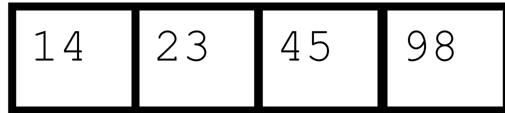




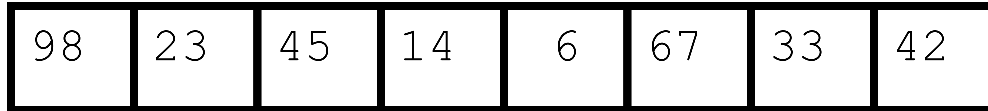


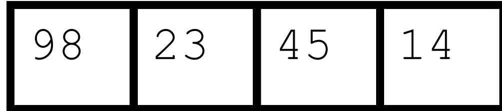


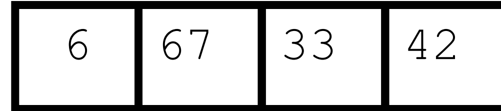


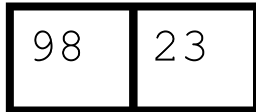


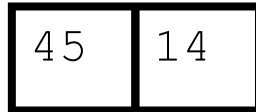
Merge

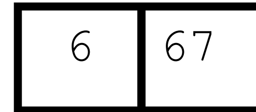


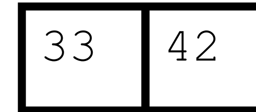


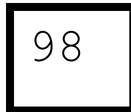


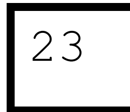


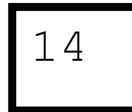


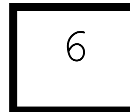


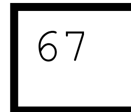


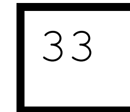


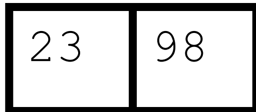


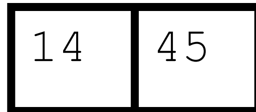


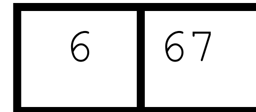


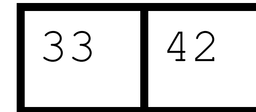


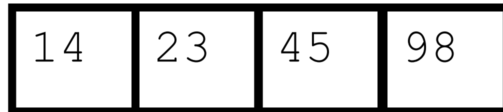




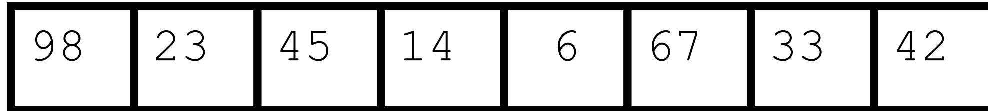


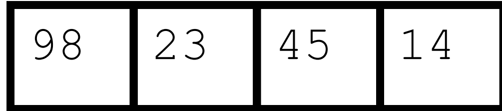


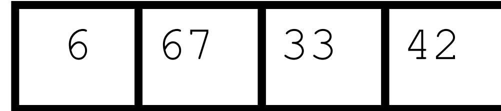


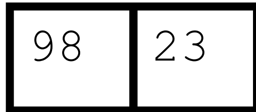


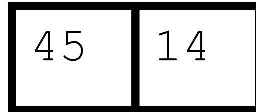
Merge

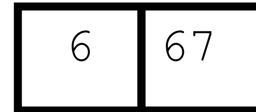


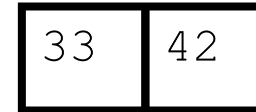


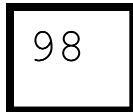


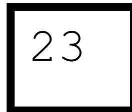


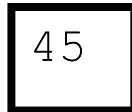


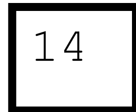


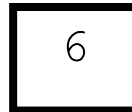


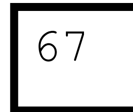


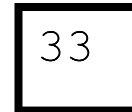


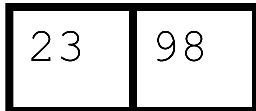


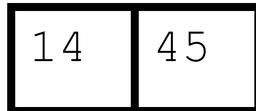


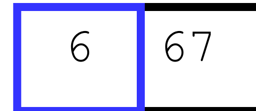


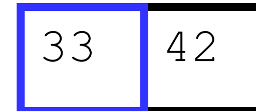


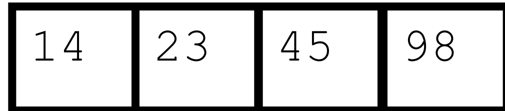




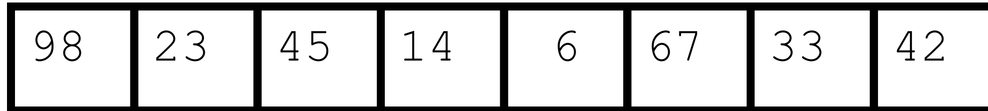


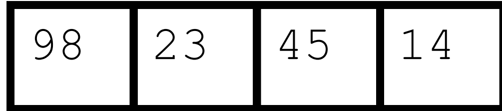


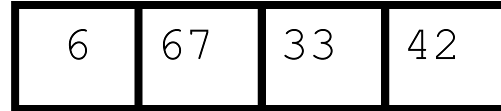


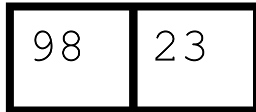


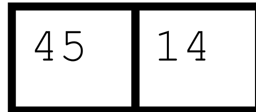
Merge

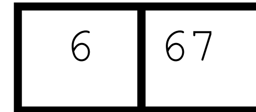


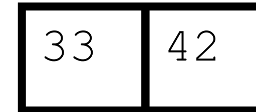


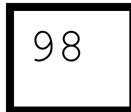


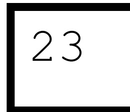


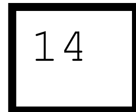


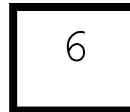


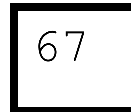


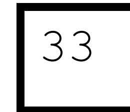


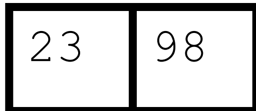


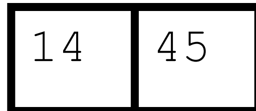


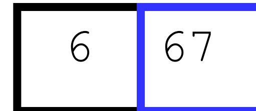


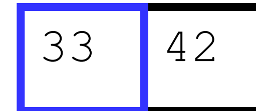


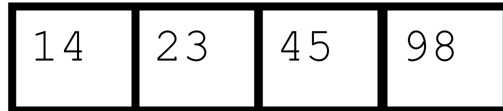


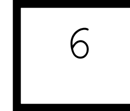




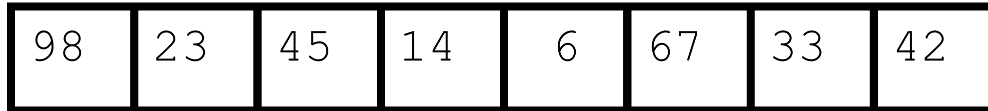


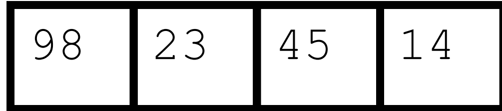


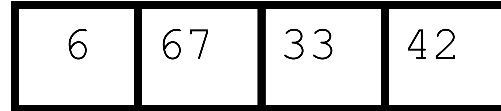


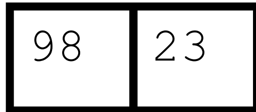


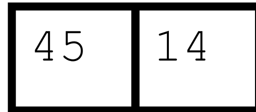
Merge

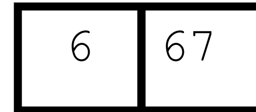


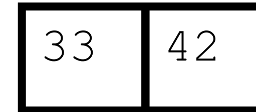


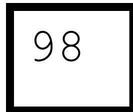


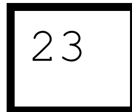


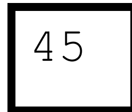


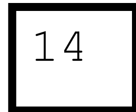


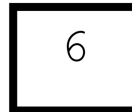


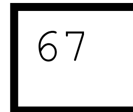


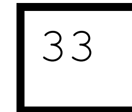


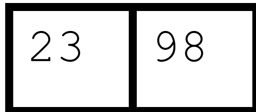


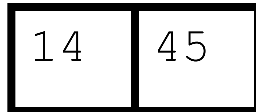


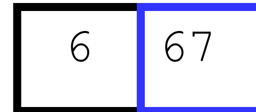


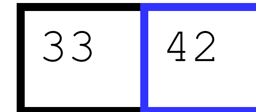


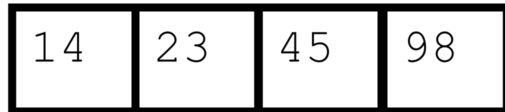


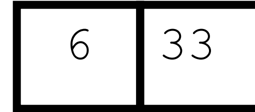




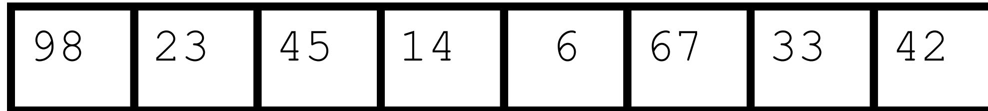
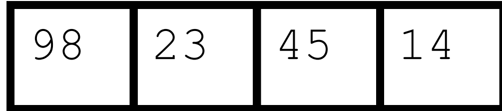
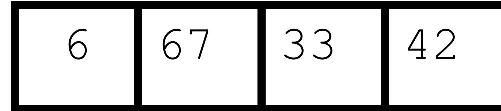
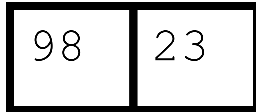
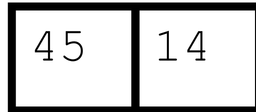
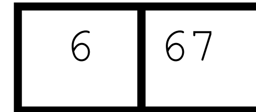
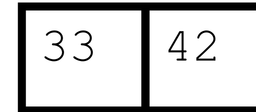
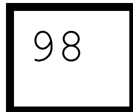
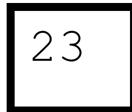
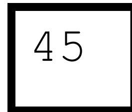
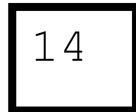
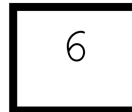
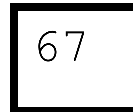
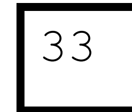


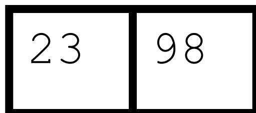
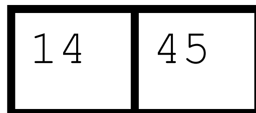
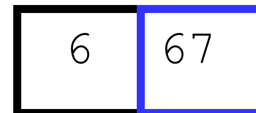
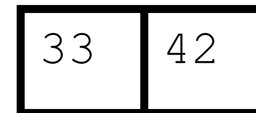
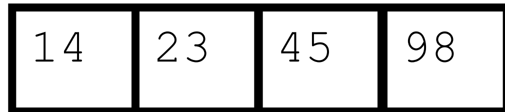
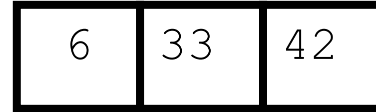




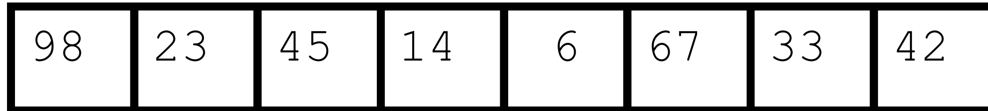


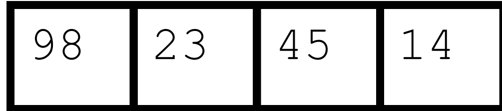
Merge

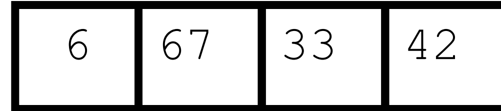

98 23 45 14 6 67 33 42

98 23 45 14

6 67 33 42

98 23

45 14

6 67

33 42

98

23

45

14

6

67

33

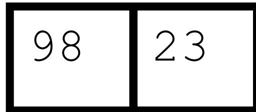
42

23 98

14 45

6 67

33 42

14 23 45 98

6 33 42

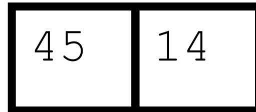
Merge

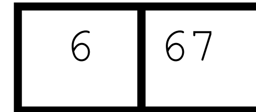

98 23 45 14 6 67 33 42

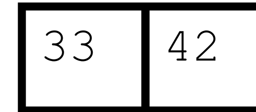

98 23 45 14

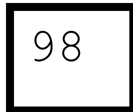

6 67 33 42

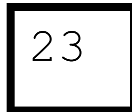

98 23

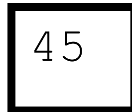

45 14

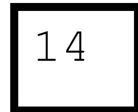

6 67

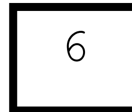

33 42

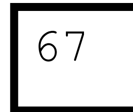

98

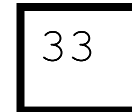

23


45

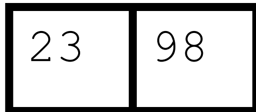

14

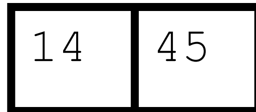

6

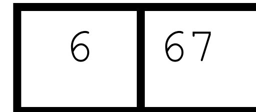

67

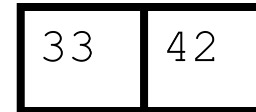

33

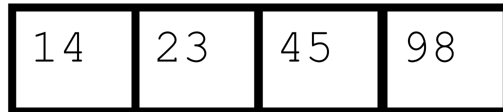
42

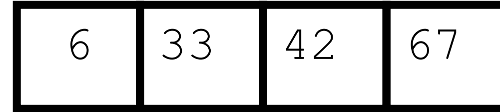

23 98


14 45

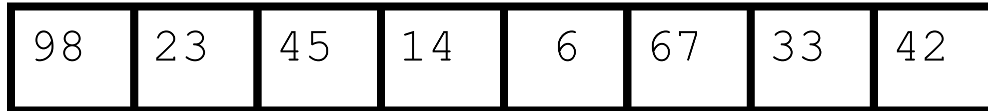

6 67

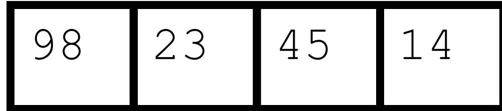

33 42

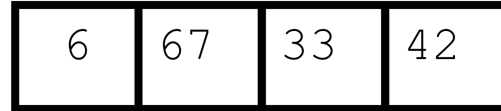

14 23 45 98

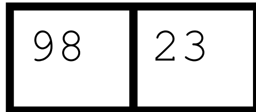

6 33 42 67

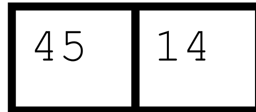
Merge

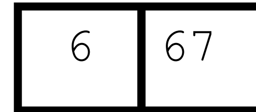


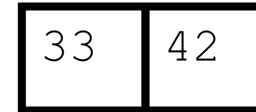


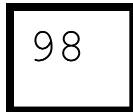


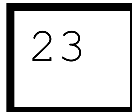


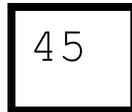


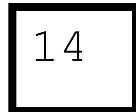


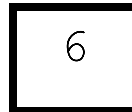


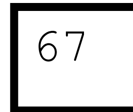


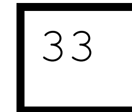


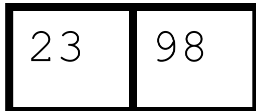


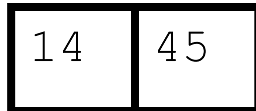


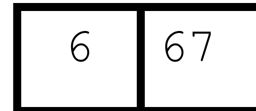


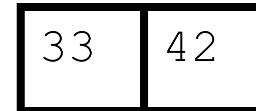


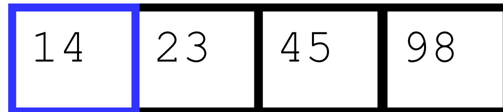


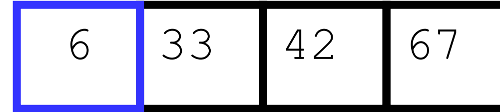




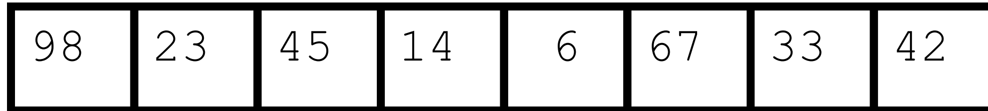


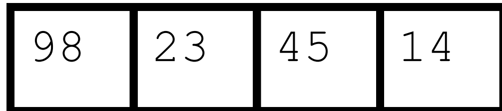


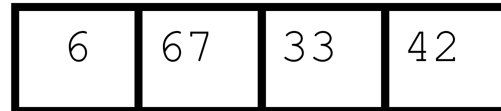


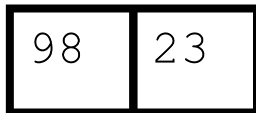


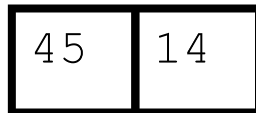
Merge

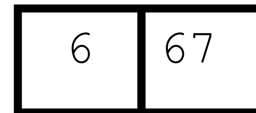


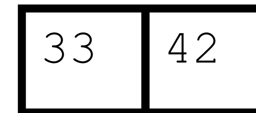


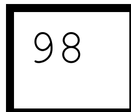


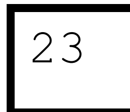


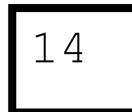


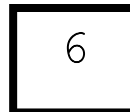


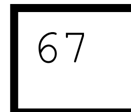


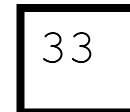


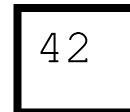


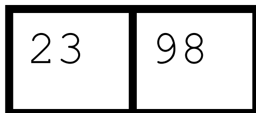


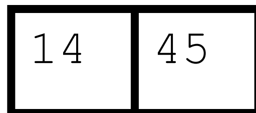


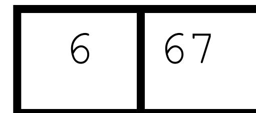


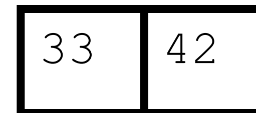


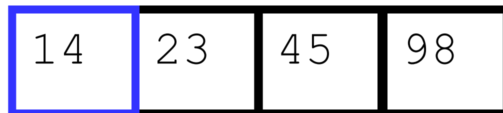


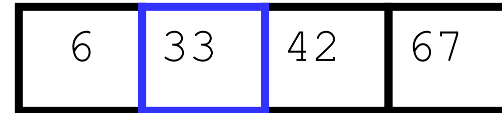


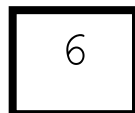




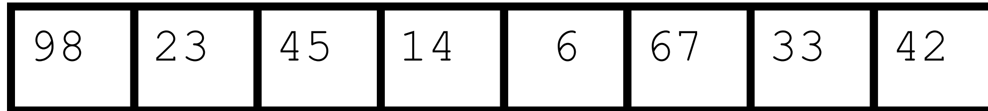


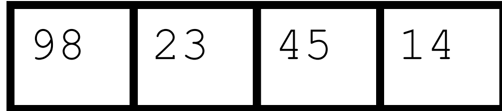


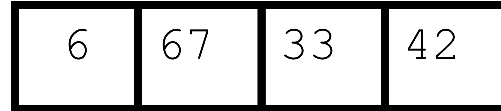


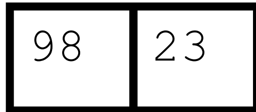


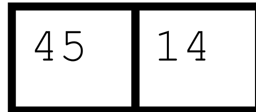
Merge

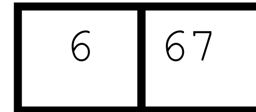


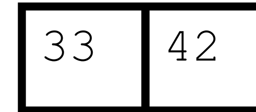


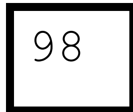


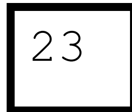


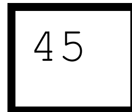


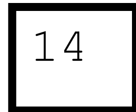


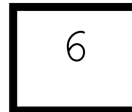


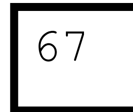


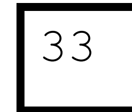


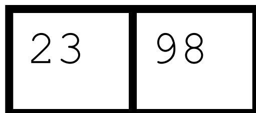


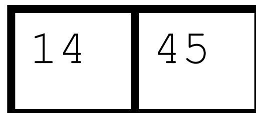


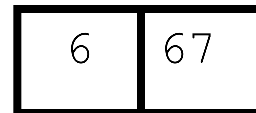


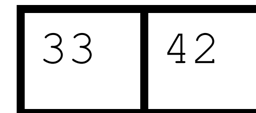


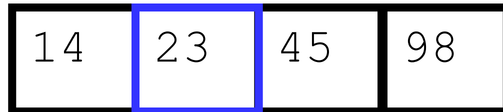


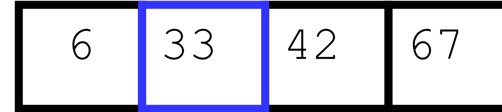


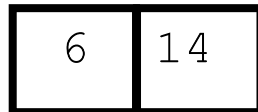




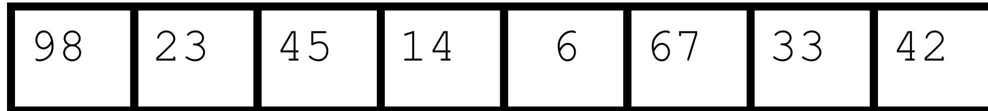


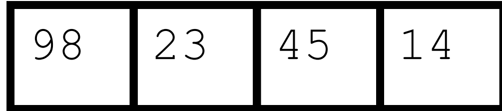


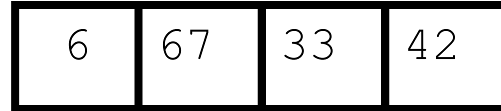


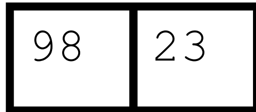


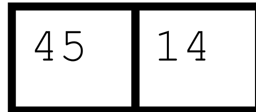
Merge

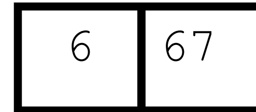


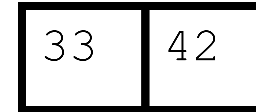


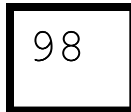


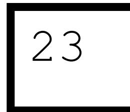


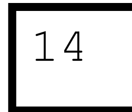


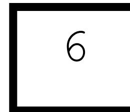


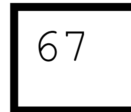


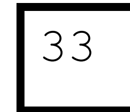


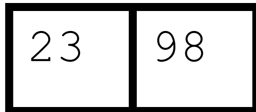


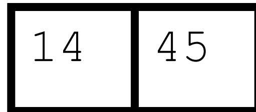


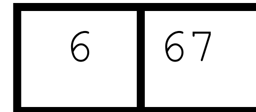


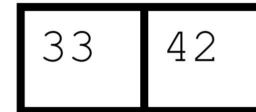


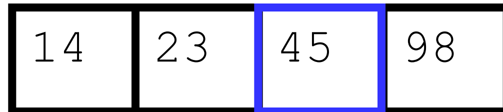


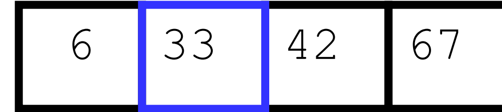


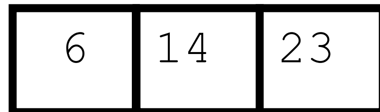




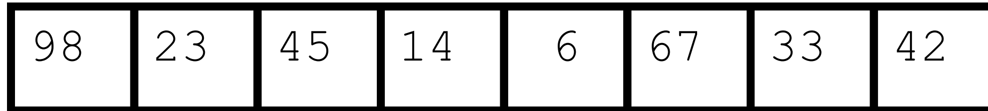


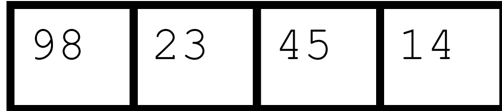


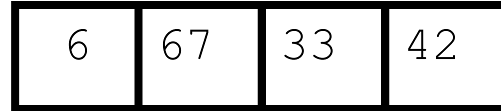


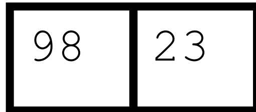


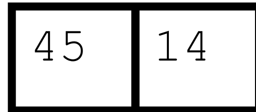
Merge

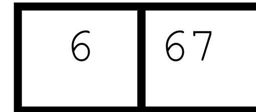


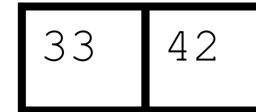


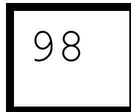


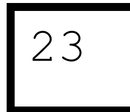


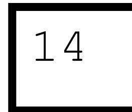


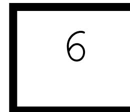


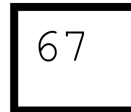


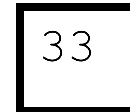


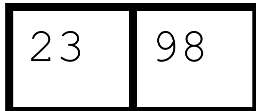


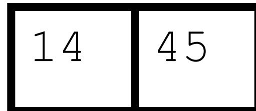


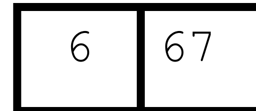


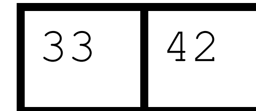


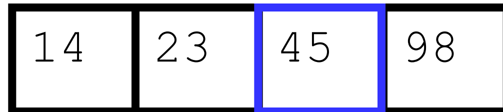


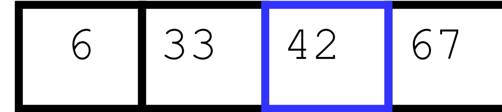


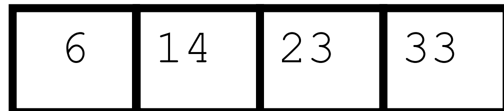




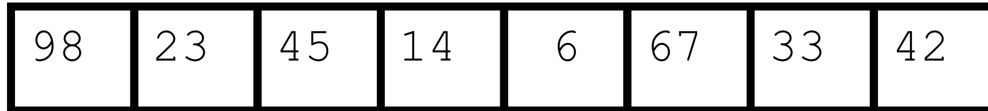


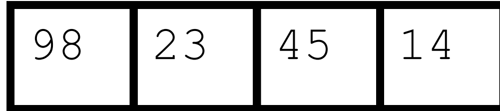


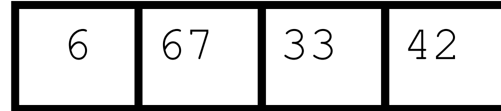


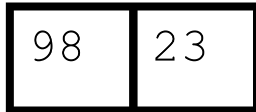


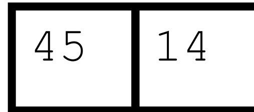
Merge

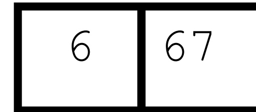


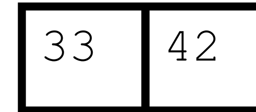


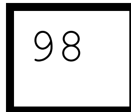


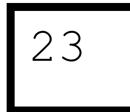


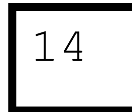


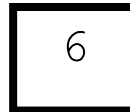


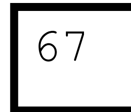


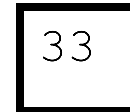


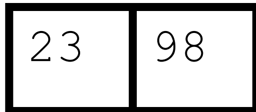


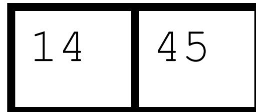


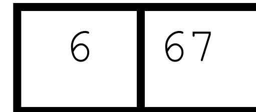


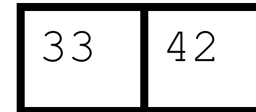


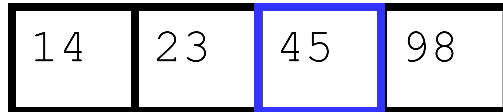


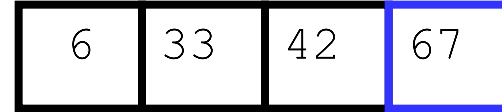


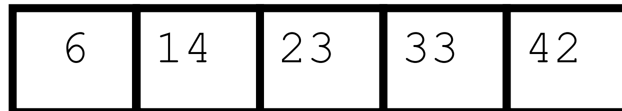




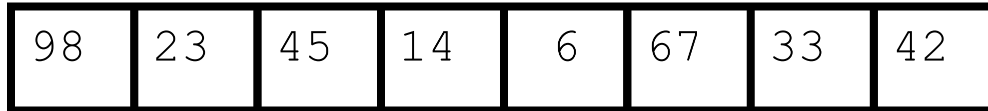


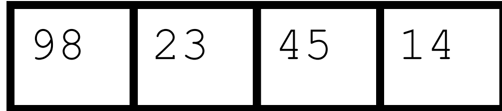


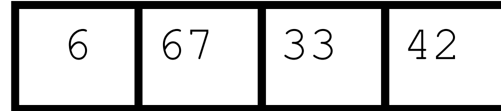


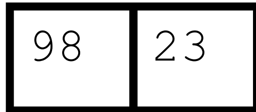


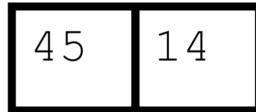
Merge

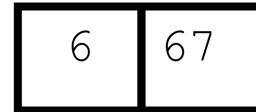


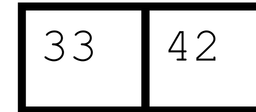


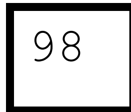


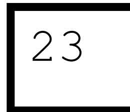


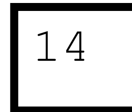


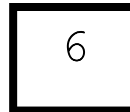


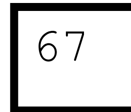


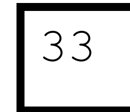


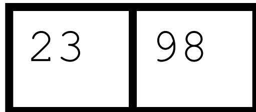


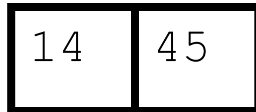


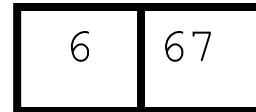


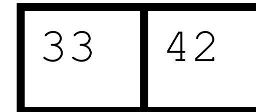


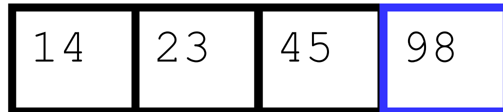


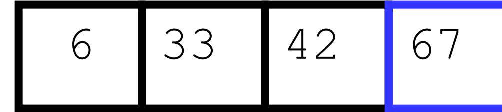


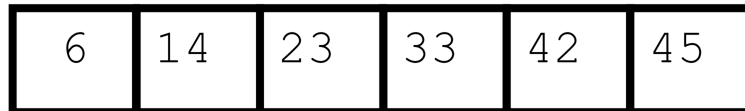




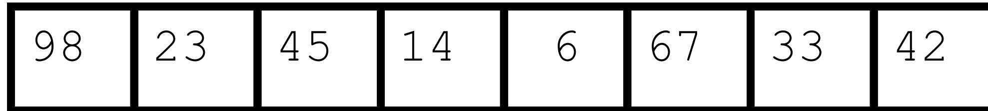


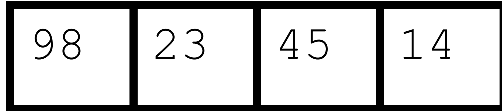


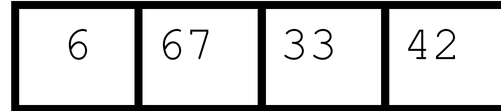


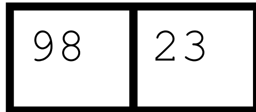


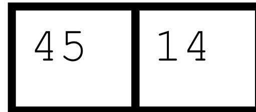
Merge

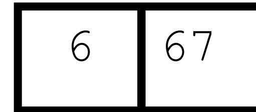


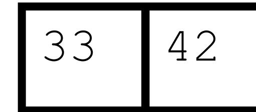


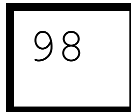


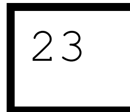


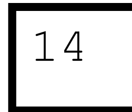


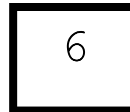


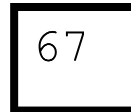


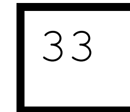


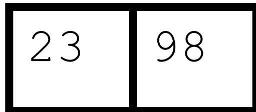


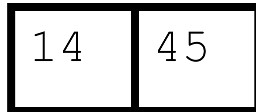


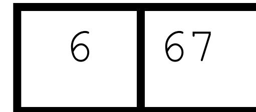


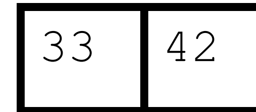


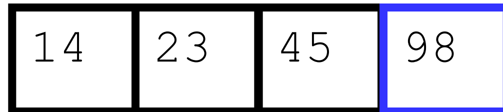


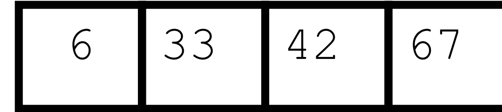


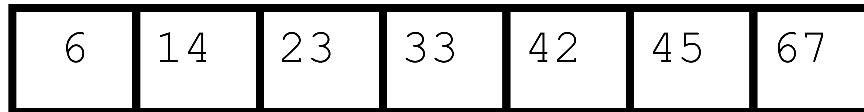




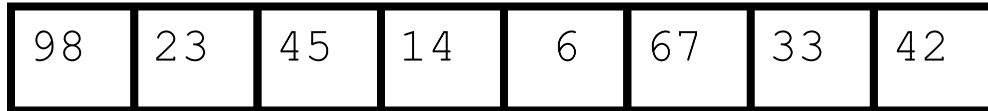


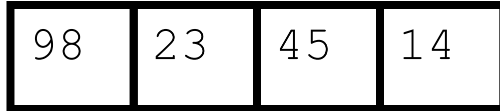


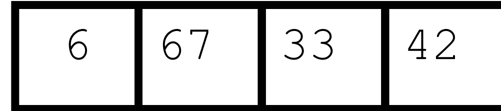


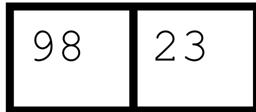


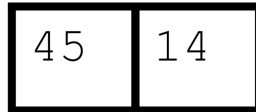
Merge

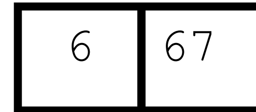


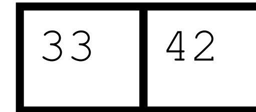


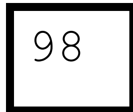


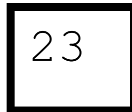


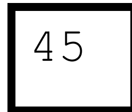


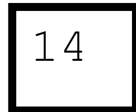


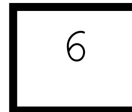


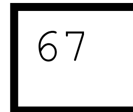


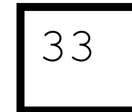


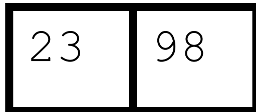


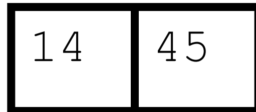


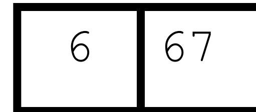


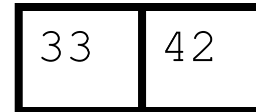


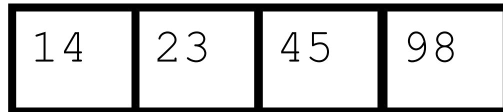


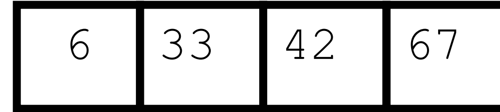


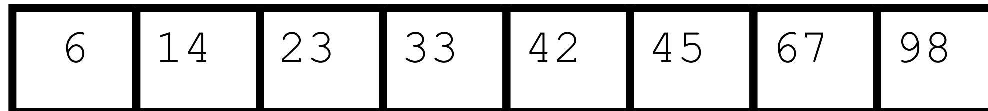












Merge

98	23	45	14	6	67	33	42

98	23	45	14

6	67	33	42

98	23

45	14

6	67

33	42

98   23   45   14   6   67   33   42

23	98

14	45

6	67

33	42

14	23	45	98

6	33	42	67

6	14	23	33	42	45	67	98

98	23	45	14	6	67	33	42

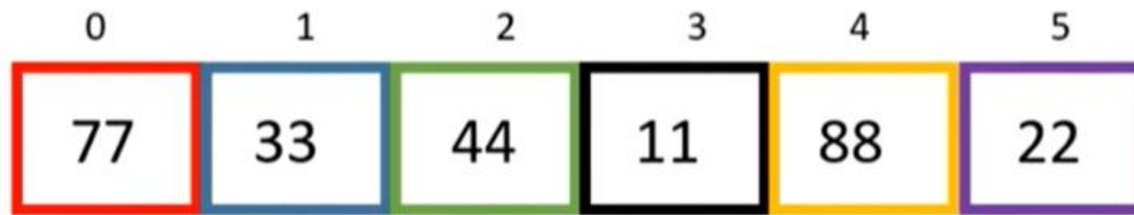
6	14	23	33	42	45	67	98

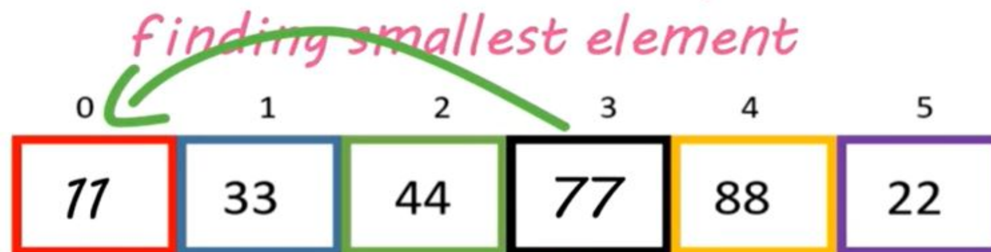
Summary

- **Divide the unsorted collection into two**
- **Until the sub-arrays only contain one element**
- **Then merge the sub-problem solutions together**

SELECTION SORT

- Pick the smallest element in each iteration



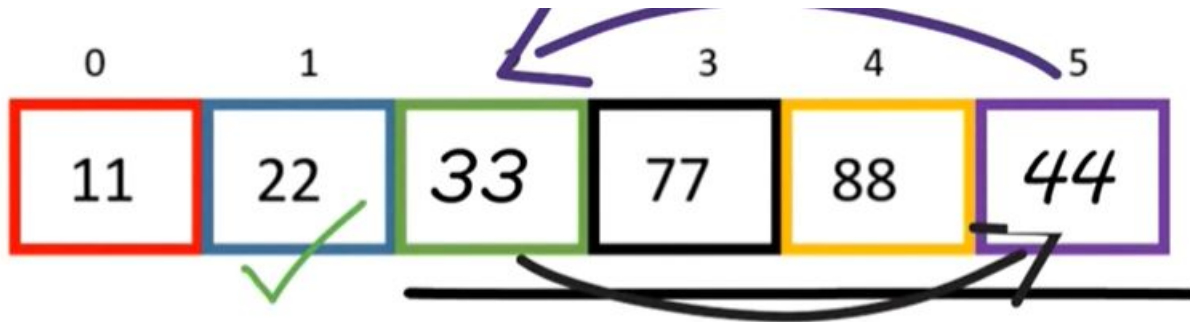


SELECTION SORT

- Pick the smallest element in each iteration
- Now ignore 11 in Pass 2

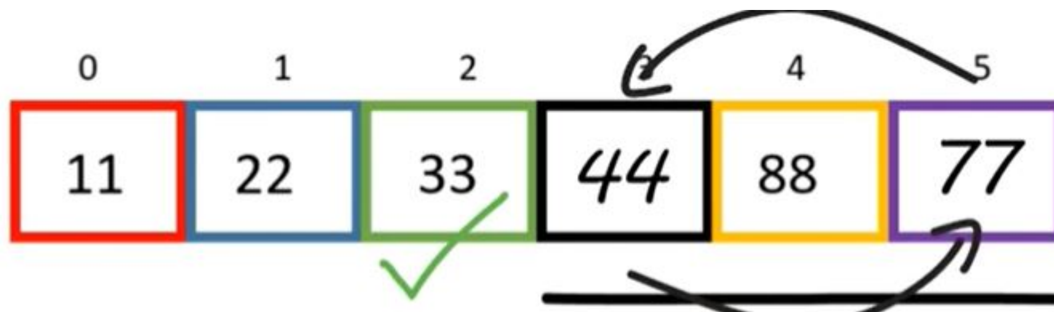
SELECTION SORT

- Pick the smallest element in each iteration
- Now ignore 11,22 in Pass 3



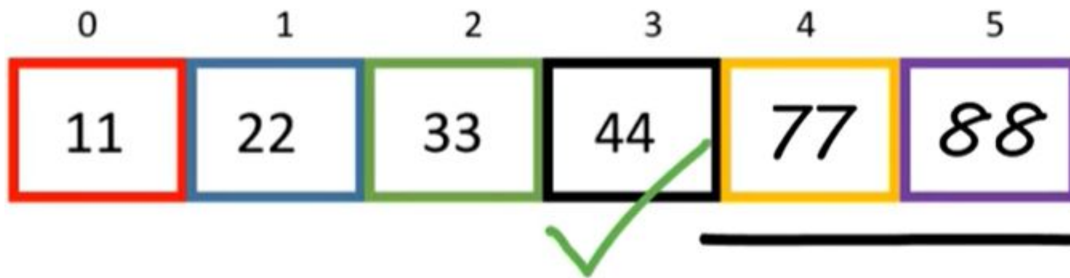
SELECTION SORT

- Pick the smallest element in each iteration
- Now ignore 11,22,33 in Pass 4

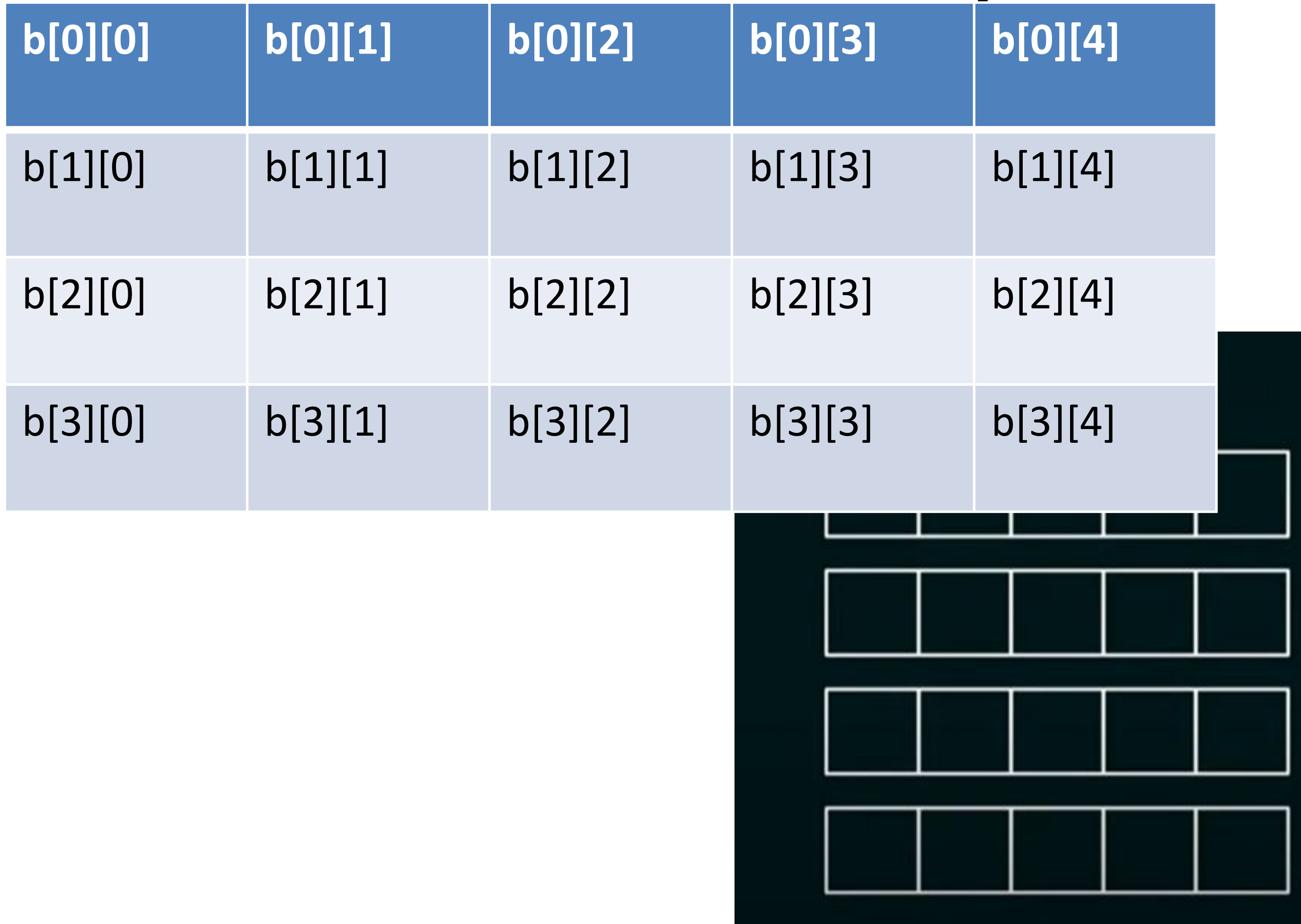


SELECTION SORT

- Pick the smallest element in each iteration
- Now ignore 11,22,33,44 in Pass 5



2D array



b[0][0]	b[0][1]	b[0][2]	b[0][3]	b[0][4]
b[1][0]	b[1][1]	b[1][2]	b[1][3]	b[1][4]
b[2][0]	b[2][1]	b[2][2]	b[2][3]	b[2][4]
b[3][0]	b[3][1]	b[3][2]	b[3][3]	b[3][4]

Sum/Scan/Print

```
for(int i=0;i<2;i++)
{
    for(int j=0;j<3;j++)
    {
        printf("%d",b[i][j]);
    }
    printf("\n");
}

for(int i=0;i<2;i++)
{
    for(int j=0;j<3;j++)
    {
        c[i][j]=a[i][j]+b[i][j];
    }

}

 for(int i=0;i<2;i++)
{
    for(int j=0;j<3;j++)
    {
        printf("%d\t",c[i][j]);
    }
    printf("\n");
}
```

Sum of diagonals

```
 for(int i=0;i<2;i++)
{
    for(int j=0;j<3;j++)
    {
        if(i==j)
        s1+=a[i][j];
        if(i+j==2)
        s2+=a[i][j];
    }
    printf("\n");
}
printf("%d %d",s1,s2);
return 0;
```

Valid?

```c
#include <stdio.h>

int main()
{
    char a[2][10]={"abc","raj"};
    int b[][2] = {{50,60},{34,54},{44,65}};
    for(int i=0;i<2;i++)
    {
        for(int j=0;j<2;j++)
        {
            printf("%d",b[i][j]);
        }
    }


    printf("%d",b[0][0]);
    return 0;
}
```

Memory Representation

Columns

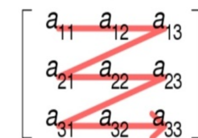
2D ARRAY	0	1	2
0	a[0][0]	a[0][1]	a[0][2]
1	a[1][0]	a[1][1]	a[1][2]
2	a[2][0]	a[2][1]	a[2][2]

Rows

Addressing

- **Example:** Given an array, **arr[1………10][1………15]** with base value **100** and the size of each element is **1 Byte** in memory. Find the address of **arr[8][6]** with the help of row-major order?

Row-major order



Column-major order

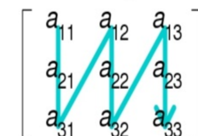


Illustration of difference between row- and column-major ordering

- *Base address B = 100*
  Storage size of one element store in any array W = 1 Bytes
  Row Subset of an element whose address to be found I = 8
  Column Subset of an element whose address to be found J = 6
  Lower Limit of row/start row index of matrix LR = 1
  Lower Limit of column/start column index of matrix = 1
  Number of column given in the matrix N = Upper Bound – Lower Bound + 1
  = 15 – 1 + 1
  = 15
- ***Formula:***
  *Address of A[I][J] = B + W * ((I – LR) * N + (J – LC))*
- ***Solution:***
  *Address of A[8][6] = 100 + 1 * ((8 – 1) * 15 + (6 – 1))*
  *= 100 + 1 * ((7) * 15 + (5))*
  *= 100 + 1 * (110)*
  Address of A[I][J] = 210

Column major

- *Base address B = 100*
  Storage size of one element store in any array W = 1 Bytes
  Row Subset of an element whose address to be found I = 8
  Column Subset of an element whose address to be found J = 6
  Lower Limit of row/start row index of matrix LR = 1
  Lower Limit of column/start column index of matrix = 1
  Number of column given in the matrix M = Upper Bound – Lower Bound + 1
  = 10 – 1 + 1
  = 10
- ***Formula:***
  *Address of A[I][J] = B + W * ((J – LC) * M + (I – LR))*
  *Address of A[8][6] = 100 + 1 * ((6 – 1) * 10 + (8 – 1))*
  *= 100 + 1 * ((5) * 10 + (7))*
  *= 100 + 1 * (57)*
  Address of A[I][J] = 157

Deletion

```
printf("Enter the value of row number :");
scanf("%d", &x);
for(i=0;i<2;i++)
{
if(i==x)
{
for(j=0;j<3;j++)
{

printf("\t%d" , b[i+1][j]);

}
i++;
}
else
{
for(j=0;j<3;j++)
{
printf("\t%d" , b[i][j]);
}
}
printf("\n");
```

Sorting

```
#include<stdio.h>

int main()
{
    int m[2][3]={{2,3,1},{4,3,5}}; int swap;
  // loop for rows of matrix
  for (int i = 0; i < 2; i++)
  {
    // loop for column of matrix
    for (int j = 0; j < 3-i; j++)
    {

        if(m[i][j] > m[i][j+1])
        {
            swap = m[i][j];
            m[i][j] = m[i][j+1];
            m[i][j+1] = swap;
        }
    }
  }
```