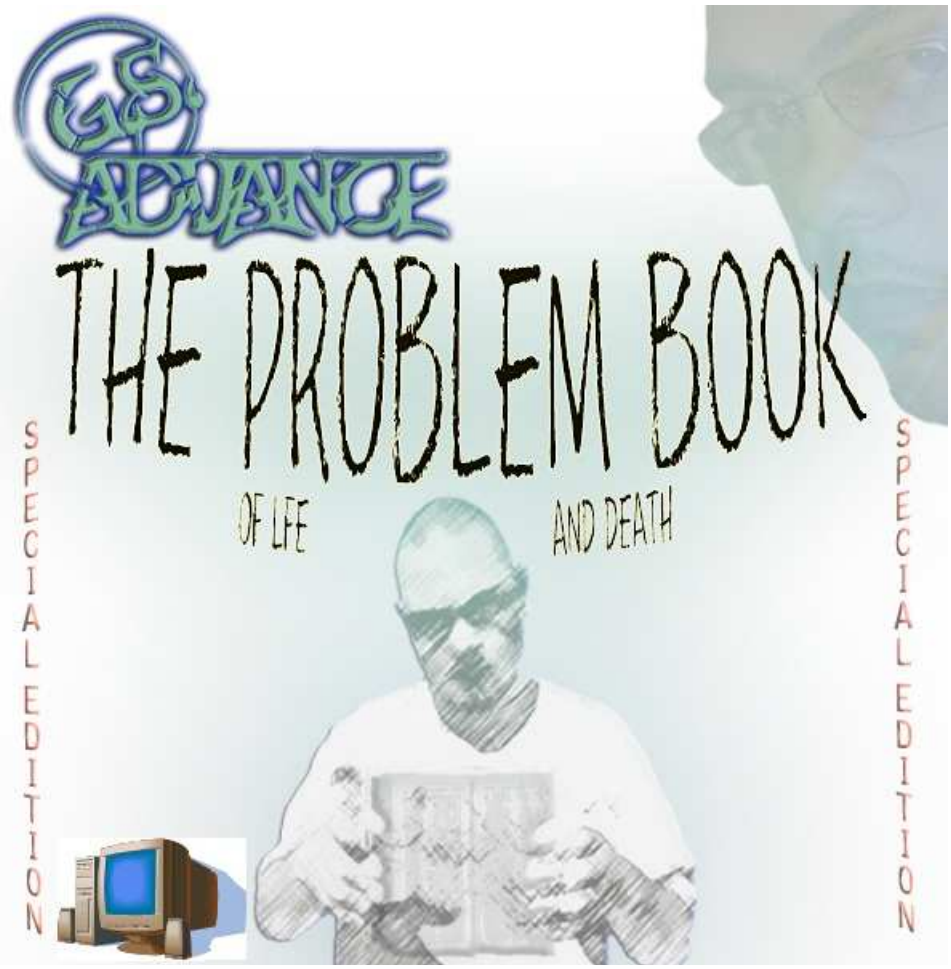


DATA STRUCTURES

By Raghav Sir

Compiled By
Uphar Goyal

Preface

Folks these questions are meant for your practice. The solutions provided may not be best to the problem but are just provided so to guide you along and have a correct vision to approach the same. For getting the best outcome out of these problems you are required to thoroughly check the correctness of the solution because some of the solutions provided are not correct instead they are mentioned just because they were originally presented by the writer of the notes. Also to mention this set of problem book is a collective effort of Alumni and was generated to help junior batches to guide and have a proper visionary for approaching problems. So in the interest of the college you are requested to use the same in the interest of college only and not to share on public forums, friends/girlfriends/boyfriends or either relatives too, outside college. To mention following is the list of doubtable questions/solutions so found by us while compiling, you are required to go through and also may seek corrections:

1. Chapter 1 Problem 28 Method 1
2. Chapter 1 Problem 29 Method 3
3. Chapter 7 Problem 44

Contents

Preface…… i
1. Arrays…….. 1
2. Linked Lists…… 19
3. Sorting…….. 37
4. String…… 39
5. Stacks and Queues…… 44
6. Trees…… 52
7. Miscellaneous Questions…….. 74

Chapter 1: ARRAYS

Q.1 Find the Kth smallest element from unsorted array of n numbers where n>>K (can even say $n \to \infty$ or numbers are being processed and one by one coming to you). Basically this question is to reduce infinite input problem to finite/constant memory/storage.

Solution:

Step 1:- From first K numbers build a max heap so that the top element is largest in those K numbers. Say that element is X.

Step 2:- Now as numbers are coming for processing, say coming number be A

If A>X then A cant be Kth smallest.

If A=X then no change is needed.

If A<X then X cant be Kth smallest element so replace X with A and again call heapify.

e.g. 5 6 1 8 3 2 4 and you have to find 3^{rd} smallest element.

Now make max heap of 3 elements 5 6 1. 6 being the largest in these 3 will be on top of the heap. Now as 8 will come 8>6 so we will neglect 8. When 3 will come since 3<6 we will replace 6 with 3 in the heap and again we will call heapify. After heapify 5 will be on the top. When 2 will come, 2<5 then 5 will be replaced by 2 and after heapification 3 will be on the top. When 4 will come since 4>3 then we will neglect the 4. Now at the end 3 is on the top of the heap so 3 is the3rd smallest element of the given numbers or array.

NOTE - For Kth largest build min heap.

Q.2 Right rotate an array by k units in place.

Solution:-

Steps for this are as follows:

1. First reverse the array.
2. Then reverse 0 to k-1.
3. Then reverse k to end.

NOTE – left rotating an array by k units = right rotating an array by (n-k) units.

Q.3 An array A of an element (n>=3) with distinct +ve integers. Array B with (n*(n-1))/2 elements which represent sum of all possible pairs of two numbers of array A. You are just given array B and you have to find array A.

Solution:-

Array A: $a_1,a_2,a_3\ldots\ldots\ldots\ldots\ldots\ldots\ldots,a_n$.

Array B: $a_1+a_2,a_1+a_3\ldots.,a_2+a_3,a_2+a_4\ldots\ldots\ldots a_3+a_4,a_3+a_5\ldots\ldots\ldots,a_{n-1}+a_n$.

Let the notation of $a_1+a_2=a_{12}$, then $a_i+a_j=a_{ij}$ where i<j.

Assumption: B contains elements in a particular order. In which first all pairs of a_1 is present then pairs of a_2 and so on.

a_{12}, $a_{13},\ldots\ldots\ldots a_{1n},a_{23},a_{24}\ldots\ldots\ldots a_{2n},a_{34},a_{35},\ldots\ldots\ldots a_{3n},a_{45},a_{46},\ldots\ldots\ldots a_{4n}\ldots\ldots\ldots a_{(n-2)(n-1)},a_{(n-2)n},a_{(n-1)n}$

Method 1:

```
void algo(int A[],int B[],int n)
```

```
{
        int i,j,k;
        int sum;
        j=n-1,k=0,sum=n-1;
        for(i=0;i<n-2;i++)
        {
                A[i]=(B[k]+B[k+1]-B[sum])/2;
                k=sum;
                j--;
                sum=sum+j;
        }
        A[n-2]=B[n-3]-A[0];
        A[n-1]=B[n-2]-A[0];
}
```

Method 2:

```
void algo(int A[],int B[],int n)
{
        int i;
        A[0]=(B[0]+B[1]-B[n-1])/2;
        for(i=1;i<n;i++)
                A[i]=B[i-1]-A[0];
}
```

Q.4 You are given an array containing +ve and –ve integers. Find the sub array with the largest sum.

Solution:-

```
void subarray(int arr[],int n)
{
    int sum=0,start=0,end=0,i;
    int tsum=0,tstart=0,tend=0;
    for(i=0; i<n; i++)
    {
            tsum+=arr[i];
            tend++;
            if(tsum>sum)
            {
                    sum=tsum;
                    start=tstart;
                    end=tend;
            }
            else if(tsum<0)
            {
                    tstart=tend=i+1;
                    tsum=0;
            }
    }
    for(i=start; i<end; i++)
            printf("%d \t",arr[i]);
```

```
}
```

Q.5 Let X[1……n] & Y[1……n] be two sorted arrays. Find the median of the two merged arrays.

Solution:-

```
int median(int a[], int b[], int n)
{
        int med1,med2;
        if(a[n-1] < b[0])
                return a[n-1];
        if(b[n-1] < a[0])
                return b[n-1];
        if(n==2)
        {
                int c[4];
                merge(a,b,c,2);
                return c[1];
        }
        med1 = a[(n-1)/2];
        med2 = b[(n-1)/2];
        if(med1 == med2)
                return med1;
        else if(med1 < med 2)
                return median(&a[(n-1)/2],b,n/2+1);
        else
                return median(a,&b[(n-1)/2],n/2+1);
}
```

Q.6 Write a program to implement binary search.

Solution:-

```
int binsearch(int a[],int lb,int ub, int x)
{
        int mid;
        while(lb<=ub)
        {
                mid=(lb+ub)/2;
                if(a[mid]==x)
                        return mid;
                else if(a[mid] >x)
                        ub=mid-1;
                else
                        lb=mid+1;
        }
        return -1;
}
```

Q.7 Make a change in above binary search so that only one comparison should be made inside the loop.

Solution:-

```
int binsearch(int a[],int lb,int ub, int x)
{
        int mid;
        mid=(lb+ub)/2;
        while(lb<=ub && x!=a[mid])
        {
                if(a[mid] >x)
                        ub=mid-1;
                else
                        lb=mid+1;
                mid=(lb+ub)/2;
        }
        if(x==a[mid])
                return mid;
        else
                return -1;
}
```

Q.8 Write a program to implement a binary search in an array which is sorted but rotated by some k places which is not known. Time complexity O(logn).

Solution:-

```
int binsearch(int a[],int lb,int ub,int x)
{
        int mid;
        while(lb<=ub)
        {
                mid=(lb+ub)/2;
                if(a[mid]==x)
                        return mid;
                if(a[mid]>x)
                        if(a[mid]<=a[ub] || x>a[ub])
                                ub=mid-1;
                        else
                                lb=mid+1;
                else
                        if(a[mid]>=a[lb] || x<a[lb])
                                lb=mid+1;
                        else
                                ub=mid-1;
        }
        return -1;
}
```

NOTE- Another method is to find pivot(k) in O(logn) and then search.
Array is rotated by k locations. This k can be find out in O(logn) then array is divided into 0 to i and i+1 to n-1. In these 2 arrays apply binary search which can find the element in O(logn).
O(logn)+O(logn)=O(logn)

Q.9 You are given an array of n+2 elements. All elements of the array occur once between 1 and n except 2 elements which occurs twice. Find the repeating elements in O(n).

Solution:-

Method 1:

1. Find sum of first n natural numbers and sum of all numbers in the array. Subtract them. Then we will get a+b = x.
2. Find sum of squares of first n natural numbers and sum of squares of all numbers present in the array. Subtract them. Then we will get $a^2 + b^2 = y$.
3. Now solve equation of step 1 and step 2.
   $z = a*b = (x^2 – y)/2$.
   $w = a-b = sqrt(x^2 – 4*z)$.
   $a=(x+w)/2$.
   $b=(x-w)/2$.

Method 2:

1. Find sum : a+b = x.
2. Find XOR : a^b = y.
3. Now for each pair which will form the sum x, check a^b whichever will match y.
4. Take that a & b as repeated numbers.

Method 3: here **value** is equal to arr[index].

1. count = 0. This variable "count" represents number of repeated elements found.
2. Start traversing the array.
3. If value == index then go ahead to next element.
4. If value > index then
   If arr[value] == arr[index] then this is repeated value. Store this at the end of the array.
   count++;
   temp = arr[index];
   arr[index]=arr[n+count];
   arr[n+count]=temp;
   else swap arr[index] and arr[value].
   Don't update the index.
5. If index > value then this value is being repeated. Store this at the end of the array.
   count++;
   temp = arr[index];
   arr[index]=arr[n+count];
   arr[n+count]=temp;
   again don't change the index and start comparing again from the same point.

Q.10 Find the unique element from an unsorted array.

Solution:-

Method 1: Check every element with every other element. $O(n^2)$

Method 2: Use hashing to find duplicate. O(n)

Method 3: Sort the array, then find the unique element in the sorted array. O(n)+O(nlogn)

Method 4: Prepare a BST and does not insert duplicate element in the BST.

Average : O(nlogn)
Worst : $O(n^2)$ when array is sorted.

Q.11 We are given a sorted array of n integers. The array has been shifted by X places(X is unknown). The numbers wrap around. Given a number as input we have to find pairs of number that add up to given number.

Solution:-

Method 1: Older method. This method only prints one pair if exists which add upto the given sum and leads to infinite loop if there exists no such pair which add upto given sum.

```
void sum_of_pairs(int a[], int n, int sum)
{
        int i,start,end,t,found=0;
        for(i=0; i<n-1; i++)
                if(a[i] > a[i+1])
                        break;
        start=(i+1)%n;
        end=i;
        while(!found)
        {
                t=a[start]+a[end];
                if(sum == t)
                {
                        found=1;
                        printf("%d %d", a[start], a[end]);
                }
                else if(t < sum)
                        start=(start+1)%n;
                else
                        end=(end==0)?n-1:end-1;
        }
}
```

Method 2: This method prints all pairs which add upto the given sum and print "no pair exist" if there is no such pair which add upto given sum.

```
void sum_of_pairs(int a[], int n, int sum)
{
        int i,start,end,t,found=0;
        for(i=0; i<n-1; i++)
                if(a[i] > a[i+1])
                        break;
        start=(i+1)%n;
        end=i;
        while(start != end)
        {
                t=a[start]+a[end];
                if(sum == t)
                {
                        found=1;
```

```
                        printf("%d %d\n", a[start], a[end]);
                        start=(start+1)%n;
                }
                else if(t < sum)
                        start=(start+1)%n;
                else
                        end=(end==0)?n-1:end-1;
        }
        if(!found)
                printf("No such pair exists\n");
}
```

Time Complexity : O(n)
Space complexity : O(1)

Q.12 There is an infinite length array. The first n elements are sorted and the remaining elements are filled with the element -1. Devise an algorithm that searches for a value X within this array, with O(logn) time. Value of n is not given.

Solution:-

```
n=1;
while(a[n]!=-1)
        n=n<<1;
```

In this way we will get n which satisfies following constraint :

2^k <= actual value of n < $2^{(k+1)}$

So now we have to find the actual value of n between 2^k and $2^{(k+1)}$ which can be find out with binary search (as we know lower bound and upper bound).

After finding the actual value of n we can find out the value X with binary search.

Time complexity: $O(\log k)+O(\log(2^k))+O(\log n)$

$O(\log k)<O(\log n)$ since $k<n$

$O(\log 2^k) < O(\log n)$ since $O(\log 2^k) < O(k) <O(\log n)$

Finally Time Complexity is O(logn).

Q.13 Write a program to print a matrix helically or spirally.

Solution:-

```
void print(int a[][MAX], int r,int c)
{
        int rows,cols,i,j,m,n;
        m=r;
        n=c;
        for(rows=0,cols=0; count<(r*c); rows++,cols++)
        {
                i=rows;
                j=cols;
                while(j<n && count < (r*c))
                {
                        printf("%d ",a[i][j]);
                        j++;
                        count++;
                }
                i++;
```

```
                j--;
                for( ; i<m && count < (r*c); count++,i++)
                        printf("%d ",a[i][j]);
                i--;
                j--;
                for( ; j > cols-1 && count < (r*c); j--,count++)
                        printf("%d ",a[i][j]);
                j++;
                i--;
                for( ; i > rows && count < (r*c); i--,count++)
                        printf("%d ",a[i][j]);
                m--;
                n--;
        }
}
```

Q.14 Given a picture of H*W. Write a program to rotate the picture by 90 degrees. Picture is of 32 bit represented by an int array.

Solution:-

Anticlockwise Rotation

```
void anticlockwise(int pic[][WIDTH], int newpic[][HEIGHT])
{
        int i,j;
        for(i=0; i<WIDTH; i++)
                for(j=0; j<HEIGHT; j++)
                        newpic[i][j]=pic[j][WIDTH-i-1];
}
```

Clockwise Rotation

```
void clockwise(int pic[][WIDTH], int newpic[][HEIGHT])
{
        int i,j;
        for(i=0; i<WIDTH; i++)
                for(j=0; j<HEIGHT; j++)
                        newpic[i][j]=pic[HEIGHT-j-1][i];
}
```

Q.15 We have an array which contains integer numbers (not any fixed range). Only two numbers are repeated odd number of times and remaining even number of times. Find the 2 numbers.

Solution:-

Method 1:

1. Hash the numbers using multiplication method.
2. At the end check the hash table for odd number of counts.

Method 2:

1. Take XOR of all the elements in the array. This will give a^b (a and b are the numbers that are repeated odd number of times).
2. Now find the least significant bit set in a^b. Let it be the kth bit.

3. Now divide the integers in the array in two groups. One , group1 , containing all those integers that are present in array and their kth bit is set and other , group2, containing all those integers that are present in the array and their kth bit is not set.
4. Take XOR of all the elements in group1 this will give one number that is repeated odd number of times. Similarly take XOR of all the elements in group2 this will give other number.

Q.16 Write a program to find the position p of the first occurrence of element t in the array. Array is sorted. Your code should make logarithmic comparisons of array elements.

Solution:-

```
int binarysearch(int a[],int n, int t)
{
        int m,lb,ub;
        lb=-1;
        ub=n;
        while(lb+1 != ub)
        {
                m=(lb+ub)/2;
                if(a[m]<t)
                        lb=m;
                else
                        ub=m;
        }
        if(ub >= n || x[ub]!=t)
                ub=-1;
        return ub;
}
```

This method is better than the original binary search because it will always make only one comparison inside the loop whereas original binary search makes two comparisons.

Q.17 Array of size N is given, N is even. In this array one entry is repeated n/2 times and the remaining n/2 entries are unique. Write an algo to find the repeated value.

Solution:- N is even so two arrangements are possible in which two duplicates numbers are never adjacent to each other.

1. X_X_X_....._X_ (here X is repeated value).
2. _X_X_X……..X_X (here X is repeated value).

In all other arrangements 2 repeating elements must be present adjacent to each other atleast once.

```
int repeat(int a[],int n)
{
        int i,flag=0;
        for(i=0; i<(n-1); i++)
                if(a[i] == a[i+1] || a[i] == a[i+2])
                {
                        flag=1;
                        break;
                }
        if(flag)
                return a[i];
```

```
        else
                return -1;
}
```

Q.18 Write a program to rotate a 1-D array of n elements by k positions.
Solution:-
Method 1:

1. Copy k elements to a temporary array.
2. Move the remaining n-k elements to left k places.
3. Then copy the first k elements from the temporary array to the last k positions in original array.

Method 2:

1. Reverse array from 0 to k-1.
2. Reverse array from k to n-1.
3. Now reverse the whole array.

Q.19 How can you find ith max element from an array.
Solution:- By using quicksort algo, we will see only one side of partition where ith element is present.

```
int randomized_select(int arr[], int lb, int ub, int i)
{
        int q;
        if(lb == ub)
                return arr[lb];
        q=randomized_partition(arr, lb, ub);
        k=q-lb+1;
        if(i == k)
                return arr[q];
        else if(i>k)
                return randomized_select(arr, q+1, ub, i-k);
        else
                return randomized_select(arr, lb, q-1, k);
}
```

Q.20 Find unique elements from sorted array.
Solution:-
Method 1:

```
int unique(int a[],int res[], int n)
{
        int i,j,k;
        res[0]=a[0];
        k=1;
        for(i=1,j=0; i<n; i++)
        {
                if(a[j]!=a[i])
                {
                        res[k++]=a[i];
```

```
                        j=i;
                }
        }
        return k;
}
```

Method 2:

```
int res[n];
int i=0;
res[0]=a[0];
int unique(int a[], int n)
{
        int start = n-1;
        if(a[0] == a[start])
        {
                if(res[i]!=a[0])
                        res[++i]=a[0];
                return i+1;
        }
        do
        {
                start=start/2;
        }while(a[0] != a[start]);
        if(a[0] != res[i])
                res[++i] = a[0];
        return unique(&a[start+1], n-start-1);
}
```

Q.21 Write a function which taken an array and finds the majority element if It exists otherwise prints none.

Solution:- A majority element in an array A[] of size n is an element that appears more than n/2 times (and hence there is at most one such element).

Method 1: The basic solution is to have two loops and keep track of maximum count for all different elements. If maximum count becomes greater than n/2 then break the loops and return the element having maximum count. If maximum count doesn't become more than n/2 then majority element doesn't exist.

Time Complexity : $O(n^2)$.

Auxiliary Space : O(1).

Method 2 : Using Binary Search Tree

Node of the Binary Search Tree (used in this approach) will be as follows:

```
struct node
{
        int element,count;
        struct node *left, *right;
}BST;
```

Insert elements in BST one by one and if an element is already present then increment the count of the node. At any stage, if count of a node becomes more than n/2 then return.
The method works well for the cases where n/2+1 occurrences of the majority element is present in the starting of the array, for example {1, 1, 1, 1, 1, 2, 3, 4}.
Time Complexity: If a binary search tree is used then time complexity will be O(n^2). If a self balancing binary search tree is used then O(nlogn)
Auxiliary Space: O(n)

Method 3:

```
void majority(int *arr, int size)
{
        int i, head, count;
        head=arr[0];
        count=1;
        for(i=1; i<size; i++)
        {
                if(head == arr[i])
                        count++;
                else if(count == 1)
                        head=arr[i];
                else
                        count--;
        }
        count=0;
        for(i=0; i<size; i++)
        {
                if(arr[i] == head)
                        count++;
        }
        if(count >  size/2)
                printf("Majority element is %d\n", head);
        else
                printf("None\n");
}
```

Time Complexity: O(n)
Auxiliary Space : O(1)

Q.22 Given an array with some repeating numbers like 3,1,5,2,10,2,5. The output should be the array with numbers like 3,1,5,5,2,2,10. They should appear as many times as they appear in the array but in the same order.

Solution:-

Method 1: If range is given, do hashing and traverse the original array and then print each number that much no. of times which is present in hash array. Then make hash entry 0.

Method 2: Make a BST including a frequency field. If no. is repeated then increase the frequency count. Now traverse the array and find that element in BST and display it that much no. of times and make frequency field 0, so that for next repeated element it will not display that element again.
Average Time complexity : O(nlogn)

Worst Time complexity: $O(n^2)$
Space complexity: O(n)

Q.23 We have an array. Now we have to print the element in the order of decreasing frequency and the same no. of times as it is coming in the input array. If frequency of 2 elements is same, they should come in same order as they are coming in the input array.

Solution:-

Method 1: If the range of numbers is known then we can do modified counting sort.

1. First find the frequencies.
2. Then apply counting sort on frequencies.

Method 2: Make a max heap based on frequency value and a BST where node points to the nodes of max heap. Whenever a new element is inserted in BST it is also inserted in heap accordingly and its index or position is stored in BST. Whenever the element is present in BST, frequency of max heap node is increased and heapify procedure is called.
Time Complexity : O(nlogn)
But this will not assure that the same frequency element will come in order of appearance.

Method 3: Let index array contains the index of elements.

1. Sort the input array & change its index array accordingly.
2. Now make a frequency array and change index array such that index array have min index for that value.
3. Now sort the frequency array and appropriately make changes in index array.
4. Now partition the frequency array such that in each partition all the values are same. Accordingly partition the index array. Sort each partition of index array.

Time complexity : O(nlogn) Space complexity : O(n)

Example:

Array given is {2, 2, 3, 2, 1, 7, 4, 3, 2, 6, 4, 7}

Index array {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11}

1. After sorting input array {1, 2, 2, 2, 2, 3, 3, 4, 4, 6, 7, 7}
   Index array is {4, 0, 1, 3, 8, 2, 7, 6, 10, 9, 5, 11}
2. Frequency array is {1, 4, 4, 4, ,4, 2, 2, 2, 2, 1, 2, 2}
   Index array is {4, 0, 0, 0, 0, 2, 2, 6, 6, 9, 5, 5}
3. Frequency array after sorting {1, 1, 2, 2, 2, 2, 2, 2, 4, 4, 4, 4}
   Index array is {4, 9, 2, 2, 6, 6, 5, 5, 0, 0, 0, 0}
4. After partitioning frequency array {1, 1}{2, 2, 2, 2, 2, 2}{4, 4, 4, 4}
   Index array is {4, 9}{2, 2, 6, 6, 5, 5}{0, 0, 0, 0}
   Sorting each partition of index array {4, 9}{2, 2, 5, 5, 6, 6}{0, 0, 0, 0}

Now output is {1, 9, 3, 3, 7, 7, 4, 4, 2, 2, 2, 2}

NOTE- There are two ways to print in decreasing order of frequency such that element of same frequency should come in the same order as they are present in the input array:

i. After sorting each partition of index array, print partitions in decreasing order of frequency (i.e. frequency associated with the partition). For above example:
   After sorting each partition of index array {4, 9}{2, 2, 5, 5, 6, 6}{0, 0, 0, 0}
   Print partitions in the order {0, 0, 0, 0}{2, 2, 5, 5, 6, 6}{4, 9}
   Now output is {2, 2, 2, 2, 3, 3, 7, 7, 4, 4, 1, 6}
ii. Sort each partition of index array in decreasing order. Then print the array from the end. For

above example:

After sorting each partition of index array	{9, 4}{6, 6, 5, 5, 2, 2}{0, 0, 0, 0}
Print partitions from the end	{0, 0, 0, 0}{2, 2, 5, 5, 6, 6}{4, 9}
Now output is	{2, 2, 2, 2, 3, 3, 7, 7, 4, 4, 1, 6}

Q.24 Write a program to find longest monotonically increasing subsequence.

Solution:-

Method 1: Let given sequence be A.

1. Sort the copy of given sequence A. let that copy be B.
2. Now perform LCS on A and B.
3. Result is LIS.

Time Complexity : $O(n^2)$

Space Complexity : $O(n^2)$

Space complexity can be improved to O(n) by making slight variation in LCS (for this refer to problem of cormen dynamic programming LCS exercise).

Method 2:

Time complexity : $O(n^2)$

Space Complexity : O(n)

We take two arrays best and prev.

best:- This will note down the length of increasing subsequence. best[i] denotes the length of increasing subsequence till i^{th} element.

prev[i] will tell us the previous element of the subsequence.

```
void lis(int a[], int n)
{
    int i, j, max=0, end, *best, *prev;
    best=(int *)malloc(sizeof(int)*n);
    prev=(int *)malloc(sizeof(int)*n);
    for(i=0; i<n; i++)
    {
            prev[i]=0;
            best[i]=1;
    }
    for(i=1; i<n; i++)
            for(j=0; j<i; j++)
                    if(a[j] < a[i] && best[i] <= best[j]+1)
                    {
                            best[i]=best[j]+1;
                            prev[i]=j;
                    }
    for(i=0; i<n; i++)
            if(max <best[i])
            {
                    end=i;
                    max=best[i];
            }
    /*printing in reverse order*/
```

```
        while(end!=0)
        {
                printf("%d\t", a[end]);
                end=prev[end];
        }
        free(best);
        free(prev);
    }
```

Method 3: Best method
Time Complexity : O(nlogn)
Space Complexity : O(n)

```
    void lis(int a[], int n)
    {
        int  i, j, u, v, m, *best, *prev;
        best=(int *)malloc(sizeof(int)*n);
        prev=(int *)malloc(sizeof(int)*n);
        best[0]=0;
        prev[0]=0;
        j=0;
        for(i=1; i<n; i++)
        {
                if(a[i] > a[best[j]])
                {
                        best[++j]=i;
                        prev[j]=best[j-1];
                        continue;
                }
                for(u=0, v=j; u<v;)
                {
                        m=(u+v)/2;
                        if(a[best[m]] < a[i])
                                u=m+1;
                        else
                                v=m;
                }
                if(u <= v && a[i] < a[best[u]])
                {
                        if(u > 0)
                                prev[u]=best[u-1];
                        best[u]=i;
                }
        }
        for(u=j, v=best[j]; u>=0; u--)
        {
                best[u]=v;
                v=prev[u];
```

```
        }
        for(i=0; i<=j; i++)
                printf("%d\t", a[best[i]]);
}
```

Q.25 Write a program to sort an array arr[n] where arrays elements are in the range of 0 to n^2-1.

$0<=arr[k]<=n^2-1$

$0<=k<=n-1$

Time Complexity : O(n)

Space Complexity : O(n)

Solution:- Apply counting sort in two steps :

1. First on arr[k]%n.
2. Then on arr[k]/n.

First step will arrange the elements on the basis of LSD (least significant digit).

Second step will arrange on the basis of most significant digit.

Q.26 Given an unsorted array of n integers. Find that pair of integers which gives the maximum sum.

Solution:- Find the largest and second largest in array. These two numbers will give maximum sum.

Time complexity : O(n)

Number of comparisons : There are three ways.

i. 2n comparisons (trivial method) : Compare maximum value and second largest value with each element of array.
ii. 3n/2 comparisons : For every two elements there will be only 3 comparisons. Total 3n/2 comparisons.
iii. n+log n comparisons : This method required space complexity of O(n).

Q.27 An array of size N has distinct values 1…….N in random order. You have only operation called rev(x) where x is any value from 0 to N-1, which reverses all values from 0 to x.

a) Objective is to sort the array using minimum number of rev(x) functions.

b) How many permutations of N size array require exactly N numbers of rev(x) operations to get sorted.

e.g. array is 1,4,2,3. rev(2) will reverse the array elements from 0 to 2. So array becomes 2,4,1,3.

Solution:-

At max N-1 rev() operations are needed.

When array is sorted, no rev() operation is needed.

When array is sorted in reverse order, one rev() operation is needed.

1. Find the smallest element.
2. Apply rev(x) where x is the index of the smallest element.
3. Apply this recursively on remaining array.

```
void sort(int a[], int n)
{
    int i, pos, min=MAX_INT;
    if(n == 1)
            return;
    for(i=0; i<n; i++)
            if(min > a[i])
            {
                    min=a[i];
                    pos=i;
```

```
            }
        if(pos != 0)
            rev(pos);
        sort(&a[1],n-1);
    }
```

Q.28 Given an array having placement of integers or characters in the given form….
a1 a2 a3 a4 a5 a6 b1 b2 b3 b4 b5 b6
Write an inplace algorithm to rearrange the elements in the array like…….
a1 b1 a2 b2 a3 b3 a4 b4 a5 b5 a6 b6.

Solution:-

Method 1:

```
void inplace(int a[], int n)
{
    int i, temp, curr, next,d;
    temp=a[1];
    curr=1;
    d=(curr+n+1)/2;
    for(i=curr+1; i<n; i++)
    {
        next=(curr%2)?(curr+n-d)%n:curr/2;
        a[curr]=i<n-1?a[next]:temp;
        curr=next;
        d=(curr+n+1)/2;
    }
}
```

Method 2: If inplace algo is not required then we can make use of another array.

Method 3: By swapping the values.
e.g. $a_1\ a_2\ a_3\ a_4\ b_1\ b_2\ b_3\ b_4$
swap(a_4, b_1)
$a_1\ a_2\ a_3\ b_1\ a_4\ b_2\ b_3\ b_4$
swap(a_3, b_1) swap(a_4, b_2)
$a_1\ a_2\ b_1\ a_3\ b_2\ a_4\ b_3\ b_4$
swap(a_2, b_1) swap(a_3, b_2) swap(a_4, b_3)
$a_1\ b_1\ a_2\ b_2\ a_3\ b_3\ a_4\ b_4$
But complexity is not O(n).

Method 4:
$|a_1\ a_2|\ |a_3\ a_4|\ |b_1\ b_2|\ |b_3\ b_4|$
Swap the last half of a's with first half of b's.
$a_1\ a_2\ b_1\ b_2\ a_3\ a_4\ b_3\ b_4$
Now recursively apply this on subarrays.
$a_1\ b_1\ a_2\ b_2\ a_3\ b_3\ a_4\ b_4$
Time Complexity : O(nlogn)
T(n)=2T(n/2)+O(n)

Q.29 We have set of n elements, divide it into two sets A and B such that sum of elements in set A = sum of elements in set B.

Solution:-

Method 1:

1. Traverse the array and add each element (find sum of all elements in the array).
2. If sum is odd then no such partition exists.
3. If it is even, start adding elements till we get the sum. If whole array get exhausted do backtracking.

Method 2:

1. Traverse the array **arr** and add each element (find sum of all elements in the array).
2. If sum is odd then no such partition exists.
3. If it is even, then take an array **SUM** of size sum/2 and another Boolean array **bool** of size sum/2.
4. Initialize all elements in Boolean array to 0.
5. For i=0 to n-1
    a. For j=1 to sum/2
        i. If(bool[j] && (j+arr[i])<=sum/2)
            1. SUM[j+arr[i]]=i
            2. bool[j+arr[i]]=1
    b. bool[arr[i]]=1
    c. SUM[arr[i]]=i
6. If(bool[sum/2])
7. Print the elements in both partitions.
8. Else
9. No such partition exists.

Space Complexity : O(sum)

Time Complexity: O(n*sum)

Method 3:

1. Traverse the array and find sum.
2. If sum is odd, no partition exists.
3. If it is even, sort the array and then take one pointer at starting and one at end
4. If last!=sum/2

    Add first and last and check for sum/2. If it is equal then we are done otherwise move both pointers in their respective directions. Now three cases arise:
    a. Add last only then check
    b. Add first only then check.
    c. Add first and last both then check.

Time Complexit : O(nlogn)

Chapter 2: LINKED LISTS

Q.1 Write a function sorted insert which inserts the node into the correct sorted position in the given list.

Solution:-

Method 1:

```
void sortedinsert(struct node **headref, struct node *newnode)
{
        struct node *current;
        if(*headref==NULL || (*headref)->data >= newnode->data)
        {
                newnode-next=*headref;
                *headref=newnode;
        }
        else
        {
                current=*headref;
                while(current->next!=NULL && current->next->data < newnode->data)
                        current=current->next;
                newnode->next=current->next;
                current->next=newnode;
        }
}
```

Method 2:

```
void sortedinsert(struct node **headref, struct node *newnode)
{
        while(*headref!=NULL && (*headref)->data < newnode->data)
                headref=&((*headref)->next);
        newnode->next=*headref;
        *headref=newnode;
}
```

Method 3:

```
void sortedinsert(struct node **headref,struct node *newnode)
{
        struct node dummy;
        struct node *current=&dummy;
        dummy.next=*headref;
        while(current->next!=NULL && current->next->data < newnode->data)
                current=current->next;
        newnode->next=current->next;
        current->next=newnode;
```

```
        *headref=dummy.next;
}
```

Q.2 Write a function "insertnth" which can insert a node at any index within a list.

Solution:-

```
void insertnth(struct node **headref, int index, int val)
{
        int count;
        for(count=0;*headref!=NULL && count<index;count++)
                headref=&((*headref)->next);
        if(count==index)
        {
                struct node *temp=(struct node *)malloc(sizeof(struct node));
                temp->data=val;
                temp->next=*headref;
                *headref=temp;
        }
        else
                printf("index out of range");
}
```

Q.3 Write a function "insertsort" which given a list, rearrange its node so they are sorted in increasing order. (it should use sorted insert).

Solution:-

```
void insertsort(struct node **headref)
{
        struct node *current=*headref;
        struct node *result=NULL;
        struct node *lnext;
        while(current!=NULL)
        {
                lnext=current->next;
                sortedinsert(&result,current);
                current=lnext;
        }
        *headref=result;
}
```

Q.4 Write a function frontbacksplit which splits the list into two sublists one for the front half and one for the back half, if the number is odd the extra element goes into the front list.

Solution:-

```
void frontbacksplit(struct node*source, struct node **frontref,struct node **backref)
{
        if(source==NULL || source->next==NULL)
```

```
        {
                *frontref=source;
                *backref=NULL;
        }
        else
        {
                struct node *slow=source;
                struct node *fast=slow->next;
                while(fast!=NULL)
                {
                        fast=fast->next;
                        if(fast!=NULL)
                        {
                                slow=slow->next;
                                fast=fast->next;
                        }
                }
                *backref=slow->next;
                slow->next=NULL;
                *frontref=source;
        }
}
```

Q.5 Write a function to remove duplicate which takes a list sorted in increasing order and delete any duplicate from the list.

Solution:-

```
void removeduplicate(struct node *head)
{
        if(head==NULL)
                return;
        else
        {
                while(head->next!=NULL)
                {
                        if(head->data==head->next->data)
                        {
                                struct node *t;
                                t=head->next;
                                head->next=t->next;
                                free(t);
                        }
                        else
                                head=head->next;
                }
```

```
        }
    }
```

Q.6 Write a function alternatesplit that takes one list and divide up its node to make two smaller lists. Element in the lists may be in any order.

e.g. List- {1,2,3,4,5,6,7}
List1-{1,3,5,7} List2-{2,4,6}

Solution:-

```
void alternatesplit(struct node *source, struct node **aref, struct node **bref)
{
    if(source==NULL)
        *aref=*bref=NULL;
    else
    {
        int flag=0;
        while(source!=NULL)
        {
            if(flag==0)
            {
                *aref=source;
                source=source->next;
                (*aref)->next=NULL;
                aref=&((*aref)->next);
                flag=1;
            }
            else
            {
                *bref=source;
                source=source->next;
                (*bref)->next=NULL;
                bref=&((*bref)->next);
                flag=0;
            }
        }
    }
}
```

Q.7 Write a function shufflemerge, given two list merges their nodes to make one list, taking node alternatively between the lists.

e.g. List1-{1,3,5,7} List2-{2,4,6}
New List-{1,2,3,4,5,6,7}

Solution:-

Method 1:

```
struct node* shufflemerge(struct node *a,struct node *b)
```

```
{
        if(a==NULL)
                return b;
        if(b==NULL)
                return a;
        struct node *head=a;
        while(a->next!=NULL && b!=NULL)
        {
                struct node *t=b;
                b=b->next;
                t->next=a->next;
                a->next=t;
                a=a->next->next;
        }
        if(a->next==NULL)
                a->next=b;
        return head;
}
```

Method 2:

```
void movenode(struct node **a, struct node **b)
{
        *a=*b;
        *b=(*b)->next;
        (*a)->next=NULL;
}
struct node* shufflemerge(struct node *a, struct node *b)
{
        struct node *result=NULL;
        struct node **lastref=&result;
        while(1)
        {
                if(a==NULL)
                {
                        *lastref=b;
                        break;
                }
                else if(b==NULL)
                {
                        *lastref=a;
                        break;
                }
                else
                {
```

```
                    movenode(lastref,&a);
                    lastref=&((*lastref)->next);
                    movenode(lastref,&b);
                    lastref=&((*lastref)->next);
                }
        }
        return result;
}
```

Method 3: Use dummy node instead of double pointer.

```
struct node dummy;
struct node *tail=&dummy;
```

replace all lastref assignment statements with tail->next and incrementing statement with tail=tail->next.
return dummy.next

Method 4: Recursion

```
struct node* shufflemerge(struct node *a,struct node *b)
{
        struct node *result,*rest;
        if(a==NULL)
                return b;
        else if(b==NULL)
                return a;
        else
        {
                rest=shufflemerge(a->next,b->next);
                result=a;
                a->next=b;
                b->next=rest;
                return result;
        }
}
```

Q.8 Write a function "sorted merge" which takes two lists, each of which is in increasing order, merge these two lists into one lists which is in increasing order.

Solution:-

Method 1:

```
void movenode(struct node **a, struct node **b)
{
        *a=*b;
        *b=(*b)->next;
        (*a)->next=NULL;
}
```

```
struct node* sortedmerge(struct node *a,struct node *b)
{
        struct node *result=NULL;
        struct node **lastref=&result;
        while(1)
        {
                if(a==NULL)
                {
                        *lastref=b;
                        break;
                }
                if(b==NULL)
                {
                        *lastref=a;
                        break;
                }
                else if(a->data <= b->data)
                        movenode(lastref,&a);
                else
                        movenode(lastref,&b);
                lastref=&((*lastref)->next);
        }
        return result;
}
```

Method 2: Use dummy node instead of double pointer.

```
struct node dummy;
struct node *tail=&dummy;
```

replace all lastref assignment statements with tail->next and incrementing statement with tail=tail->next.

return dummy.next

Method 3:

```
void movenode(struct node **a, struct node **b)
{
        struct node *temp;
        temp=*a;
        *a=*b;
        *b=(*b)->next;
        (*a)->next=temp;
}

struct node* sortedmerge(struct node *a,struct node *b)
{
```

```
        struct node *head=a;
        struct node **start=&a;
        int flag=0;
        if(a==NULL)
                return b;
        if(b==NULL)
                return a;
        while(*start!=NULL && b!=NULL)
        {
                if((*start)->data>=b->data)
                {
                        movenode(start,&b);
                        if(flag==0)
                                head=*start;
                }
                start=&((*start)->next);
                flag=1;
        }
        if(*start==NULL)
                *start=b;
        return head;
}
```

Method 4: Recursion

```
struct node* sortedmerge(struct node *a,struct node *b)
{
        struct node *result=NULL;
        if(a==NULL)
                return b;
        if(b==NULL)
                return a;
        if(a->data <= b->data)
        {
                result=a;
                result->next=sortedmerge(a->next,b);
        }
        else
        {
                result=b;
                result->next=sortedmerge(a,b->next);
        }
        return result;
}
```

Q.9 Write a function reverse which reverse the linked list.

Solution:-

Method 1:

```
void reverse(struct node **headref)
{
        struct node *r=NULL;
        struct node *s=*headref;
        struct node *t=s->next;
        while(1)
        {
                s->next=r;
                if(t==NULL)
                        break;
                r=s;
                s=t;
                t=t->next;
        }
        *headref=s;
}
```

Method 2:

```
void reverse(struct node **headref)
{
        struct node *r=NULL,*s=*headref,*t;
        while(s!=NULL)
        {
                t=s->next;
                s->next=r;
                r=s;
                s=t;
        }
        *headref=r;
}
```

Method 3:

```
void movenode(struct node **a, struct node **b)
{
        struct node *temp;
        temp=*a;
        *a=*b;
        *b=(*b)->next;
        (*a)->next=temp;
}
```

```
void reverse(struct node **headref)
{
        struct node *result=NULL;
        struct node *current=*headref;
        while(current!=NULL)
                movenode(&result,&current);
        *headref=result;
}
```

Method 4: Recursion

```
struct node* reverse(struct node *root)
{
        struct node *temp;
        if(root==NULL || root->next==NULL)
                return root;
        temp=reverse(root->next);
        root->next->next=root;
        root->next=NULL;
        return temp;
}
```

Method 5: When one pointer is used(means no temporary pointer is used). Take head pointer as global.

```
void reverse(struct node *t)
{
        if(t==NULL || t->next==NULL)
        {
                head=t;
                return;
        }
        reverse(t->next);
        t->next->next=t;
        t->next=NULL;
}
```

NOTE-To reverse doubly linked list, just change next and prev pointers & at last change the head pointer.

Q.10 Write a function for mergesort.

Solution:-

```
void mergesort(struct node **headref)
{
        if((*headref)->next!=NULL)
        {
                struct node *a,*b;
```

```
                frontbacksplit(*headref,&a,&b);
                mergesort(&a);
                mergesort(&b);
                *headref=sortedmerge(a,b);
        }
        else
                return;
}
```

Q.11 Write a function for sortedintersect in which two sorted lists are available and task is to find the intersection of two lists. Lists must not be changed, a list has to be created.

e.g. List 1-{1,2,4,7,8}
List 2-{2,5,6,7,8,10}
List {2,8}

Solution:-

Method 1:

```
struct node* sortedintersect(struct node *a,struct node *b)
{
        struct node *current=NULL;
        struct node **headref=&current;
        while(a!=NULL && b!=NULL)
        {
                if(a->data==b->data)
                {
                        (*headref)=(struct node*)malloc(sizeof(struct node));
                        (*headref)->next=NULL;
                        (*headref)->data=a->data;
                        headref=&((*headref)->next);
                        a=a->next;
                        b=b->next;
                }
                else if(a->data > b->data)
                        b=b->next;
                else
                        a=a->next;
        }
        return current;
}
```

Method 2: Recursion

```
struct node* sortedintersect(struct node *a,struct node *b)
{
        struct node *t;
        if(a==NULL || b==NULL)
                return NULL;
        if(a->data==b->data)
        {
                t=(struct node *)malloc(sizeof(struct node));
```

```
                t->data=a->data;
                t->next=sortedintersect(a->next,b->next);
                return t;
        }
        else if(a->data > b->data)
                return sortedintersect(a,b->next);
        else
                return sortedintersect(a->next,b);
}
```

Q.12 Given two huge numbers represented as link lists, write a routine to add them and return a no. in the same format.

Solution:-

Reverse the link lists.
Take carry = 0.
Now add as follows:

L3->data=(L1->data+L2->data+carry)%10.
carry=(L1->data+L2->data+carry)/10.

Do this until one of the list get empty.
Now add carry to remaining list and copy contents to new node and then finally reverse the third newly created link list.

NOTE- If link lists can be changed then we will make use of the larger link list to store the result and make program space efficient.

Q.13 How do you implement a generic link list.

Solution:-

```
struct node
{
        void *data;
        struct node *next;
};

void insert(struct node **head, void *data, unsigned int n)
{
        struct node *t;
        int i;
        t=(struct node*)malloc(sizeof(struct node));
        t->data=malloc(n);
        for(i=0; i<n; i++)
                *(char *)(t->data+i)=*(char *)(data+i);
        t->next=*head;
        *head=t;
}

void printint(void *p)
{
        printf("%d\n",(int *)p);
```

```
}

void printstr(void *str)
{
        printf("%s\n",(char *)str);
}

void print(struct node *head, void (*f)(void *))
{
        while(head)
        {
                (*f)(head->data);
                head=head->next;
        }
}
```

The print function will be called like print(head, printint) if data is integer.

Q.14 In a sorted doubly linked list, how do you optimize binary search. Given a head pointer, want to search a constant.
Solution:- I don't think so there will be any improvement in binary search (whatever be the type of link list given).

Q.15 You will be given the address of the head of a link list and a random number generator (generates between 0 & 1). You have to return a node from the list randomly using the random no. such that list should be traversed only once.
Solution:-
Method 1:

1. Start traversing the link list.
2. At each node call random function.
3. If the value is greater than 0.5 then choose this node otherwise move forward.

Method 2:

Keep a counter and increment it with each visited node.
if(node->next!=NULL)
This will assume that length of the link list is n+1 and returns the current node with n/(n+1) probability.
1st node with 1/2.
2nd node with 1/3 and so on.
rand() > n/(n+1) then output the value of the node.

Method 3:

Make a slight change in method 2. Instead of using n/(n+1) use n/(n+rand()).

Q.16 You are given a doubly linked list with one pointer of each node pointing to the next node just as in single linked list. The second pointer can point to any of the node in the linked list. Write a program to copy the above list in O(n) time.
Solution:-

```
struct node
 {
        int data;
        struct node *next,*any;
}

struct node* copy(struct node *head)
{
        if(head == NULL)
                return NULL;
        struct node *new=NULL, *temp=head, *temp1;
        while(temp!=NULL)
        {
                new=(struct node *)malloc(sizeof(struct node));
                new->data=temp->data;
                new->next=temp->next;
                temp->next=new;
                temp=new->next;
        }
        temp=head;
        while(temp!=NULL)
        {
                if(temp->any!=NULL)
                        temp->next->any=temp->any->next;
                else
                        temp->next->any=NULL;
                temp=temp->next->next;
        }
        new=head->next;
        temp=head;
        temp1=new;
        while(1)
        {
                temp->next=temp1->next;
                temp=temp->next;
                if(temp!=NULL)
                {
                        temp1->next=temp->next;
                        temp1=temp1->next;
                }
                else
                        break;
        }
        return new;
}
```

Q.17 Write a program to implement XOR linked list.

Solution:- In Xor linked list, we always have to point to two consecutive nodes to get address of next node(either in forward direction or in backward direction).

A	B	C	D
NULL^B	A^C	B^D	C^NULL

```
struct node
{
        int data;
        struct node *diff;
}

#define XOR(x,y) (struct node*)((int)x ^ (int)y)
#define XOR3(x,y,z) (struct node*)((int)x ^ (int)y ^ (int)z)

void insert(int data, struct node **head,struct node **tail)
{
        struct node *prev, *curr, *temp, *t;
        prev = NULL;
        curr = *head;
        while( curr && curr->data <data)
        {
                temp=XOR(prev,curr->diff);
                prev=curr;
                curr=temp;
        }
        t=(struct node *)malloc(sizeof(struct node));
        t->data=data;
        t->diff = XOR(prev,curr);
        if(curr!=NULL)
                curr->diff=XOR3(curr->diff,prev,t);
        else
                *tail=t;
        if(prev != NULL)
                prev->diff=XOR3(prev->diff,curr,t);
        else
                *head=t;
}
```

Deletion

```
void delete(struct node **head, struct node **tail, int i)
{
        struct node *prev, *temp, *curr;
        prev=NULL;
        curr=*head;
        while(curr && i--)
        {
                temp=XOR(prev, curr->diff);
                prev=curr;
                curr=temp;
```

```
        }
        if(curr==NULL)
                return;
        temp=curr;
        curr=XOR(prev, curr->diff);
        if(prev!=NULL)
                prev->diff=XOR3(prev->diff,curr,temp);
        else
                *head=curr;
        if(curr!=NULL)
                curr->diff=XOR3(curr->diff,temp,prev);
        else
                *tail=prev;
        free(temp);
}
```

Q.18 Write a program to reverse a link list.

Solution:-

```
#define XOR(x,y)  (int) (x) ^ (int)(y)
struct node* reverse(struct node *x, struct node *y)
{
        while(x!=NULL)
        {
                x=(struct nde *)XOR(x->next, y);
                y=(struct nde *)XOR(x->next, y);
                x=(struct nde *)XOR(x->next, y);
                x=(struct nde *)XOR(x, y);
                y=(struct nde *)XOR(x, y);
                x=(struct nde *)XOR(x, y);
        }
        return y;
}
Initial call is head = reverse(head, NULL)
```

Q.19 There is a sorted circular linked list of integers. You are given a node pointer at somewhere in between but which way the list is sorted is not given i.e. it could be that pointer towards right contains element > or < than current. Now we have to insert a node in this circular list.

Solution:-

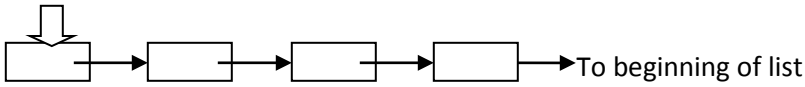
Method 1:

```
void insert(struct node *p, int val)
{
        struct node *t,*temp,**ref;
        int flag=0;
        t=p;
        while(t->data < t->next->data)
                t=t->next;
        p=t->next;
        t->next=NULL;
```

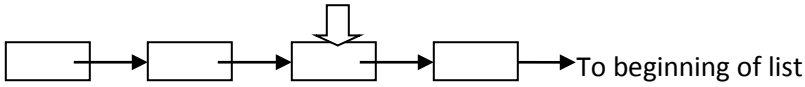
```
        ref=&p;
        while((*ref) != NULL && (*ref)->data < val)
        {
                ref=&((*ref)->next);
                flag=1;
        }
        temp=(struct node*)malloc(sizeof(struct node));
        temp->data=val;
        temp->next=*ref;
        *ref=temp;
        while(*ref)
                ref=&((*ref)->next);
        if(flag == 1)
                *ref=p;
        else
                *ref=temp;
}
```

Method 2:

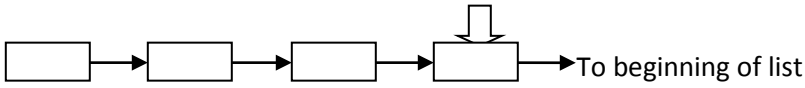
Case 1:



Case 2:



Case 3:



In all the above three cases there are two possibilities:-

a) Val to be inserted is greater than current.
b) Val to be inserted is smaller than current.

Assumption: distinct values are present in the list.

```
        void insert(struct node *t, int val)
        {
                int flag=1;
                if(val < t->data)
                        while(t->data < t->next->data)
                                t=t->next;
                else
                {
                        while(t->data < t->next->data && val > t->next->data)
                                t=t->next;
                        flag=0;
                }
                if(t->data > t->next->data && val > t->next->data && flag)
                {
                        t=t->next;
                        while(t->next->data < val)
```

```
                    t=t->next;
        }
        struct node *temp=(struct node *) malloc(sizeof(struct node));
        temp->next=t->next;
        t->next=temp;
        temp->data=val;
}
```

Q.20 Delete the nth node from the end of a link list in one pass.

Solution:-

```
void deletenthfromend(struct node **root, int n)
{
        if(! *root)
                return;
        struct node *temp=*root, *prev=*root;
        int flag=0;
        while(n > 1 && temp!=NULL)
        {
                temp=temp->next;
                n--;
        }
        if(temp!=NULL)
        {
                while(temp->next != NULL)
                {
                        prev=*root;
                        root=&((*root)->next);
                        temp=temp->next;
                        flag=1;
                }
                if(flag)
                {
                        prev->next=(*root)->next;
                        free(*root);
                }
                else
                {
                        *root=(*root)->next;
                        free(prev);
                }
        }
}
```

Chapter 3 : SORTING

Q.1 Write a function of quicksort.
Solution:-

```
void quicksort(int arr[],int lb,int ub)
{
        if(lb<ub)
        {
                int q;
                q=partition(arr,lb,ub);
                quicksort(arr,lb,q-1);
                quicksort(arr,q+1,ub);
        }
}
void partition(int arr[],int lb,int ub)
{
    int i,j,x;
    i=lb-1;
    j=lb;
    x=arr[ub];
    for(;j<ub;j++)
    {
        if(arr[j]<=x)
        {
            i++;
            swap(arr[i],arr[j]);
        }
    }
    swap(arr[i+1],arr[ub]);
    return i+1;
}
```

We can use randomized quicksort by taking pivot as any random element in the array which reduces the chances of worst case. The above method will perform least number of swaps.

Q.2 Write a program to sort an array containing only 0, 1 and 2 randomly.
Solution:-

```
    int partition(int a[], int lb, int ub, int x)
    {
        int up, down,t;
        up=lb;
        down=ub;
        while(up < down)
        {
            while( a[up] == x)
                up++;
            while(a[down] != x)
                down--;
```

```
                    if(up < down)
                    {
                        t=a[up];
                        a[up]=a[down];
                        a[down]=t;
                        up++;
                        down--;
                    }
                }
                return down+1;
        }
```

In main function the function partition() will be used as:

```
pivot=partition(a,0,n-1,0);
pivot=partition(a,pivot,n-1,1);
```

Q.3 Write a function for iterative mergesort.

Solution:-

```
        void mergesort(int arr[],int n)
        {
                int l1,l2,u1,u2,i,j,k,size,*aux;
                aux=(int *)malloc(sizeof(int)*n);
                size=1;
                while(size<n)
                {
                        l1=0,k=0;
                        while(l1+size < n)
                        {
                                l2=l1+size;
                                u1=l2-1;
                                u2=(l2+size-1<n)?l2+size-1:n-1;
                                for(i=l1,j=l2;i<=u1&&j<=u2;k++)
                                if(arr[i]<=arr[j])
                                        aux[k]=arr[i++];
                                else
                                        aux[k]=arr[j++];
                                for(;i<=u1;k++)
                                        aux[k]=arr[i++];
                                for(;j<=u2;k++)
                                        aux[k]=arr[j++];
                                l1=u2+1;
                        }
                        for(i=l1;k<n;i++)
                                aux[k++]=arr[i];
                        for(i=0;i<n;i++)
                                arr[i]=aux[i];
                        size=size*2;
                }
        }
```

Chapter 4 : STRING

Q.1 Write strlen function that returns length in less complexity provided we have char str[100],str can contain any string of length 100 maximum.

Solution:-

Assumption:- string can contain only 'a'-'z' or 'A'-'Z' or'0'-'9'.

```
int strlen(char str[])
{
        int lb,ub,mid;
        lb=0,ub=99,mid=(lb+ub)/2;
        while(1)
        {
                if(str[mid]=='\0')
                        return mid;
                else if(str[mid]==printable character)
                        lb=mid+1;
                else
                        ub=mid-1;
                mid=(lb+ub)/2;
        }
}
```

Q.2 In inplace string reversal what are sources of error to routine.

Solution:-

1. Null pointer is being passed as the string pointer.
2. String is not terminated by '\0'.

Q.3 Given a string s1 and a string s2. Write a snippet to say whether s2 is a rotation of s1 using one call to strstr routine.

```
e.g :  s1=abcd          s2=cdab
          return true.
       s1=abcd          s2=acbd
          return false.
```

Solution:-

```
int check(char str1[],char str2[])
{
        char *t;
        if(strlen(str1) == strlen(str2))
        {
                strcat(s1,s1);
                t=strstr(s1,s2);
                if(t!=NULL)
                return 1;
        }
        return 0;
}
```

Q.4 Replace a char with a string. Input is a string and we have to replace all occurrences of a character with a string.
Input = bbbbaccdee
Replace a->xyz and d->uv
Ouput = bbbbxyzccuvee
Solution:-

1. In first pass count no. of characters. If a character is to be replaced add string length of its replacement. Now we have length of the output string.
2. Start from the back of the input string and place character by character at the calculated length and if the character is to be replaced then place their replacement string.

Q.5 Write a program to find number of a palindrome in a given string.
Solution:-

```
int palindrome(char a[])
{
        int i,j,k,l,count,len;
        len=strlen(a);
        i=j=k=l=count=0;
        while(i<len && j<len)
        {
                k=i;
                l=j;
                /*calculating number of odd length palindrome*/
                while(k>=0 && l<len && a[k]==a[l])
                {
                        count++;
                        k--;
                        l++;
                }
                i++;
                j++;
        }
        i=0;
        j=1;
        while(i<len && j<len)
        {
                k=i;
                l=j;
                /*calculating number of even length palindrome*/
                while(k>=0 && l<len && a[k]==a[l])
                {
                        count++;
                        k--;
                        l++;
                }
                i++;
                j++;
        }
```

```
        return count;
}
```

Or we can do is take a for loop which iterate till the end of string and take both loops for odd and even length palindrome in the for loop.

Variation in question: Write a program to find the longest palindrome in the given string.

For this apply same approach, instead of incrementing count check the length of new founded string with the length of max string.

Q.6 Check whether a given string consists the substring of type "ab*c".

Solution:-

```
char *str=INPUT STRING
int i=0,flag=0;
while(str[i])
{
        if(str[i++]=='a')
        {
                while(str[i]=='b')
                        i++;
                if(str[i++]=='d')
                {
                        printf("substring found");
                        flag=1;
                        break;
                }
        }
}
if(!flag)
        printf("substring not found");
```

Q.7 Given two strings. Find out whether they are anagrams or not?

Solution:-

Method 1:

1. Hash the values of string1(increment by 1).
2. Now hash the value of string2 (decrement by 1).
3. Now recheck all the indexes corresponding to string1 in hash table if anyone is not equal to zero return false.
4. And check all the indexes corresponding to string2 if anyone is not equal to zero return false.

Method 2:

1. Sort both the strings using
   a) Counting sort b) any comparison sort
2. Now compare both the strings

This method Is inefficient as counting sort requires O(n) space complexity and comparison sort requires O(nlogn) time complexity.

Method 3:

```
for( i=0; str1[i]!='\0'; i++)
        sigma1+=str1[i]*str1[i];
```

```
for( i=0; str2[i]!='\0'; i++)
        sigma2+=str2[i]*str2[i];
if(sigma1 ==  sigma2)
        return true;
return false;
```

NOTE- Not sure whether this will work or not (not tested for all test cases).

Q.8 Write all the test cases for anagram checking program.

Solution:-

1. Str1 is NULL.
2. Str2 is NULL.
3. Both str1 and str2 is NULL.
4. Both are anagram of each other.
5. Non-anagram strings.
6. Non-anagrams with unequal length.
7. Anagram strings but one in upper case.
8. Non-anagram but only difference is length (one string having only one character extra).

Q.9 Given two strings A and B, how would you find out if the characters in B are a subset of characters in A.

Solution:-

```
int subset(char a[], char b[])
{
        static int hash[8];
        int i;
        for(i=0; b[i]!='\0'; i++)
                hash[b[i]/32] |= (1 << (b[i]%32));
        for(i=0; a[i]!='\0';  i++)
                hash[a[i]/32] &= ~(1 << (a[i]%32));
        for(i=0; b[i]!='\0'; i++)
                if(((hash[b[i]/32] & (1 << (a[i]%32))) >> (b[i]%32)) ==0)
                        continue;
                else
                        break;
        if(a[i] == '\0')
                return 1;
        return 0;
}
```

Q.10 Given two strings S1 and S2.Delete from S2 all those characters which occur in S1 and finally create a clean S2 with relevant characters deleted.

Solution:-

Hashing: Proceed like above question.

Take an hash array hash[8].

For all the characters present in S1 set their corresponding bit.

Now check that character is present in S2. If present then delete it.

Q.11 Write an efficient C code for tr. tr abc xyz replace a with x, b with y and c with z in the input file.

Solution:-

Create an character array of length 256.

```
int n = sizeof(argv[1]);
int m=sizeof(argv[2]);
if(m == n)
{
        int i=0;
        for(i=0; i<n; i++)
                array[argv[1][i]] = argv[2][i];
        while(!eof)
        {
                c=getc();
                putc(array[c]);
        }
}
```

Chapter 5: STACKS AND QUEUES

Q.1 Write function for the push, pop and find minimum in a stack in constant time.
Solution:

```
#define MAX 1000
struct stack
{
        int arr[Max];
        int top;
}st,min;

st.top=min.top=0;

void push(int data)
{
        if(st.top==0)
        {
                st.arr[st.top++]=data;
                min.arr[min.top++]=data;
        }
        else if(st.top<MAX)
        {
                st.arr[st.top++]=data;
                if(min.top<MAX && data<=min.arr[min.top-1])
                        min.arr[min.top++]=data;
        }
        else
                printf("stack overflow\n");
}

int pop()
{
        if(st.top==0)
                printf("stack is empty");
        else
        {
                int temp=st.arr[--st.top];
                if(temp==min.arr[min.top-1])
                        min.top--;
                return temp;
        }
}
```

```
int minimum()
{
        if(min.top>0)
                return min.arr[min.top-1];
        return -1;
}
```

Q.2 Write a program to implement stack using two queues.

Solution:

```
#define MAX 1000
struct queue
{
        int front,rear;
        int arr[MAX];
}queue1,queue2;

void add(struct queue *q, int data)
{
        if(q->rear==MAX)
                printf("stack is full");
        else if(q->rear<MAX)
                q->arr[q->rear++]=data;
}

int del(struct queue *q)
{
        if(q->front==q->rear)
        {
                printf("stack is empty");
                return -1;
        }
        else
        {
                int temp=q->arr[q->front];
                q->front=q->front+1;
                if(q->front==q->rear)
                        q->front=q->rear=0;
                return temp;
        }
}

queue1.rear=queue1.front=queue2.rear=queue2.front=0;
```

Version A:- Push efficient

```
void push(int data)
{
        if(queue1.rear!=0)
                add(&queue1,data);
        else
                add(&queue2,data);
}

int pop()
{
        if(queue1.rear!=0)
        {
                while(queue1.front!=queue1.rear-1)
                        add(&queue2,del(&queue1));
                return del(&queue1);
        }
        else if(queue2.rear!=0)
        {
                while(queue2.front!=queue2.rear-1)
                        add(&queue1,del(&queue2));
                return del(&queue2);
        }
        else
        {
                printf("stack is empty");
                return -1;
        }
}
```

Version B:- Pop efficient

```
int pop()
{
        if(queue1.front!=queue1.rear)
        {
                return del(&queue1);
        }
        else if(queue2.front!=queue2.rear)
        {
                return del(&queue2);
        }
        else
        {
                printf("stack is empty");
                return -1;
```

```
        }
}

void push(int data)
{
        if(queue1.rear==0)
        {
                add(&queue1,data);
                while(queue2.front!=queue2.rear)
                        add(&queue1,del(&queue2);
        }
        else
        {
                add(&queue2,data);
                while(queue1.front!=queue1.rear)
                        add(&queue2,del(&queue1);
        }
}
```

Q.3 Write a program to sort a stack in ascending order. Use push, pop, isempty and isfull functions of stack.

Solution:-

Assumption: All elements in the stack are distinct.

```
#define MAX 1000
struct  stack
{
int arr[MAX];
int top;
}s;
```

Method 1: Recursion

```
void sort(struct stack s)
{
        int t;
        if(isempty())
                return;
        t=s.pop();
        sort(s);
        res_push(s,t);
}

void res_push(struct stack s,int d)
{
        int t;
        if(s.isempty() || s.top()>d)
        {
```

```
                s.push(d);
                return;
        }
        t=s.pop();
        res_push(s,d);
        s.push(t);
}
```

Method 2: Non recursive. Use another Stack.

```
void sort(struct stack s)
{
        struct stack t;
        int d;
        while(!s.isempty())
                t.push(s.pop());
        while(!t.isepmty())
        {
                d=t.pop();
                if(s.isempty() || s.top()>d)
                        s.push(d);
                else
                {
                        while(!s.isempty() && s.top()<d)
                                t.push(s.pop());
                        s.push(d);
                }
        }
}
```

Q.4 Write a program for implementing priority queue using arrays. Function you should define are add(), mindelete() and maxdelete().

Solution:-

Priority queue is a data structure in which intrinsic ordering of elements does not determine the results of its basic operations. It is of two types : ascending priority queue and descending priority queue.

add() : inserts data element in the ascending order , Complexity O(n).

mindelete(): O(1).

maxdelete(): O(1).

```
#define MAX 1000
struct queue
{
        int front,rear;
        int arr[MAX];
}pqueue;
```

```
void add(int data)
{
	if(pqueue.front==(pqueue.rear+1)%MAX)
		printf("queue is full");
	else
	{
		int  i,j,k,temp;
		k=(pqueue.rear-pqueue.front+MAX)%MAX;
		/*calculating appropriate position of data*/
		for(i=0; i<k; i++)
			if(pqueue.arr[(i+pqueue.front)%MAX]>data)
				break;
	/*stroing data at calculated position and shifting the array towards right*/
		temp=pqueue.arr[(i+pqueue.front)%MAX];
		pqueue.arr[(i+pqueue.front)%MAX]=data;
		for(j=i; j<k; j++)
		{
			int t= pqueue.arr[(j+1+pqueue.front)%MAX];
			pqueue.arr[(j+1+pqueue.front)%MAX]=temp;
			temp=t;
		}
		pqueue.rear=(pqueue.rear+1)%MAX;
	}
}

int mindelete()
{
	if(pqueue.front==pqueue.rear)
	{
		printf("queue is empty");
		return -1;
	}
	else
	{
		int temp=pqueue.arr[pqueue.front];
		pqueue.front=(pqueue.front+1)%MAX;
		return temp;
	}
}

int maxdelete()
{
	if(pqueue.front==pqueue.rear)
	{
```

```
                printf("queue is empty");
                return -1;
        }
        else
        {
                pqueue.rear=(pqueue.rear-1)%MAX;
                return pqueue.arr[pqueue.rear];
        }
}
```

Q.5 Write add and delete functions of circular queue.

Solution:-

Method 1: In this method, one element of array is not used i.e. if size of array is 5 then you can store only 4 elements.

```
#define MAX 1000
int arr[MAX],front=0,rear=0;

void add(int data)
{
        if(front==(rear+1)%MAX)
                printf("queue is full");
        else
        {
                arr[rear]=data;
                rear=(rear+1)%MAX;
        }
}

int delete()
{
        if(front==rear)
        {
                printf("queue is empty")
                return -1;
        }
        else
        {
                int  t=arr[front];
                front=(front+1)%MAX;
                return t;
        }
}
```

Method 2: In this complete array is utilized.

```
#define MAX 1000
int arr[MAX],front = -1,rear=0;

void add(int data)
{
        if(front==(rear+1)%MAX)
                printf("queue is full");
        else
        {
                if(front==-1)
                        front=rear=0;
                else
                        rear=(rear+1)%MAX;
                arr[rear]=data;
        }
}

int delete()
{
        if(front == -1)
        {
                printf("queue is empty")
                return -1;
        }
        else
        {
                int  t=arr[front];
                if(front == rear)
                        front=rear=-1;
                else
                        front=(front+1)%MAX;
                return t;
        }
}
```

Chapter 6 : TREES

Q.1 Write a program to find inorder successor of a node in binary tree.
Solution:-

```
struct tree* inorder(struct tree *root)
{
        struct tree *q;
        if(root->right)
        {
                q=root->right;
                while(q->left)
                        q=q->left;
                return q;
        }
        q=root->parent;
        while(q && q->right==root)
        {
                root=q;
                q=root->parent;
        }
        return q;
}
```

Q.2 How can we traverse a binary tree without using more than constant memory.
Solution:- Structure of binary tree must contain a parent field.

```
void inorder(struct tree *root)
{
        struct tree *p;
        if(!root)
                return;
        while(1)
        {
                while(p->left)
                        p=p->left;
                printf("%d\n",p->data);
                while(1)
                {
                        if(p->right)
                        {
                                p=p->right;
                                break;
                        }
                        else
                        {
                                while(p==p->parent->right)
                                        p=p->parent;
```

```
                                        p=p->parent;
                                        if(!p)
                                                return;
                                        printf("%d\n",p->data);
                                }
                        }
                }
        }
```

Q.3 Give an algorithm to find out if the sum of elements lying in any root to leaf path of the tree has given sum or not.

Solution:-

```
int hassum(struct tree *root, int sum)
{
        if(root==NULL && sum==0)
                return 1;
        if(root==NULL)
                return 0;
        return hassum(root->left, sum - root->data)||hassum(root->right, sum - root->data);
}
```

It will return 1 if the sum of elements in any path from root to leaf in the tree has given sum else it will return 0.

Q.4 Check whether given binary tree is BST or not.

Solution:-

Method 1:

```
int isbst(struct tree *root)
{
        static int min=INT_MIN;
        if(!root)
                return 1;
        if(isbst(root->left) && min<root->data)
                min=root->data;
        else
                return 0;
        return isbst(root->right);
}
```

Method 2:

```
int isbst(struct tree *root)
{
        if(!root)
                return 1;
        if(root->data > max(root->left) && root->data < min(root->right))
                return isbst(root->left)&&isbst(root->right)
        return 0;
}
```

Q.5 Given two nodes p &q in a BST find their closest ancestor.
Solution:-

```
struct tree* closestances(struct tree *root, struct tree *p,struct tree *q)
{
        struct tree *t;
        if(root==NULL || p==NULL || q==NULL || root==p || root==q)
                return NULL;
while(root)
{
if(p->data<root->data&&q->data>root->data||p->data>root->data&&q->data<root->data)
                return root;
else if(p->data > root->data && q->data > root->data)
{
                t=root;
                root=root->right;
}
else if(p->data < root->data && q->data < root->data)
{
                t=root;
                root=root->left;
}
else if(p->data == root->data || q->data == root->data)
                return t;
}
}
```

Q.6 Write a function to check if two binary tree are isomorphic.
Solution:- Two trees are isomorphic if they are same or can be obtained by a series of flips. A flip across a node is swapping its left and right subtree.

```
int isomorphic(struct tree *p,struct tree *q)
{
        if(p==NULL && q==NULL)
                return 1;
        if(p==NULL || q==NULL)
                return 0;
        if(p->data == q->data)
        return  ((isomorphic(p->left,q->left)&&isomorphic(p->right,q->right))  ||  (isomorphic(p-
>left,q->right) && isomorphic(p->right,q->left)));
        return 0;
}
```

Q.7 How do you convert a binary tree into a BST.
Solution:-
Method 1:

i. Take inorder or preorder of binary tree and store that in an array.
ii. Now sort the array.
iii. Now place the sorted element from the array in the binary tree in inorder form (Find the

leftmost element, place first element of sorted array there and then second in its inorder successor and so on.)

This method doesn't changes the binary tree.
Time complexity: O(n)+ O(nlogn)+O(n)=O(nlogn)
Space complexity: O(n)

Method 2: This method involves changing of binary tree. Let previous tree root be oldroot and BST root be BST. Call function as oldroot=covert2bst(oldroot, NULL)

```
struct tree* convert2bst(struct tree *oldroot, struct tree *bst)
{
        struct tree *left,*right,*temp1,*temp2;
        if(oldroot==NULL)
                return bst;
        left=oldroot->left;
        right=oldroot->right;
        oldroot->left=oldroot->right=NULL;
        temp1=bst;
        if(bst)
        {
                while(temp1)
                {
                        temp2=temp1;
                        if(temp1->data >=oldroot->data)
                                temp1=temp1->left;
                        else
                                temp1=temp1->right;
                }
                if(temp2->data<oldroot->data)
                        temp2->right=oldroot;
                else
                        temp2->left=oldroot;
        }
        else
                bst=oldroot;
        if(left)
                bst=convert2bst(left,bst);
        if(right)
                bst=convert2bst(right,bst);
        return bst;
}
```

Q.8 Count the number of nodes in a binary tree.

Solution:-

```
int maxsize(struct tree *root)
{
    int l,r;
    if(!root)
            return 0;
```

```
    l=maxsize(root->left);
    r=maxsize(root->right);
    return l+r+1;
}
```

Q.9 Double tree():- for each node in a binary search tree, create a new duplicate node and insert the duplicate as the left child of the original node. The resulting tree should still be binary search tree.

Solution:-

```
void doubletree(struct tree *root)
{
    struct tree *t,*temp;
    if(!root)
            return;
    temp=(struct tree*)malloc(sizeof(struct tree));
    temp->data=root->data;
    temp->left=temp->right=NULL;
    t=root->left;
    root->left=temp;
    temp->left=t;
    doubletree(temp->left);
    doubletree(root->right);
}
```

Q.10 Given a binary tree , print all of its root to leaf paths.

Solution:- Take arr[HEIGHT] as global array and call function printpaths(root,0).

```
void  printpaths(struct tree *root,int pathlen)
{
    if(root==NULL)
            return;
    arr[pathlen++]=root->data;
    if(root->left == NULL && root->righ t== NULL)
    {
            int i;
            for(i=0;i<pathlen;i++)
                    printf("%d\t",arr[i]);
            printf("\n");
    }
    printpaths(root->left,pathlen);
    printpaths(root->right,pathlen);
}
```

Q.11 Write a program to count trees. Suppose you are building n node binary search tree with values 1 to n. How many structurally different binary search trees are there that store those values.

Solution:-

```
    int counttrees(int numkeys)
    {
            if(numkeys<=1)
                    return 1;
```

```
        else
        {
                int sum=0;
                int left,right,root;
                for(root=1; root<=numkeys; root++)
                {
                        left=counttrees(root-1);
                        right=counttrees(numkeys-root);
                        sum+=left*right;
                }
        }
        return sum;
}
```

Q.12 You are given a binary tree (not a BST). You have to find is there any element which occurs twice, and if so what is the value.

Solution:-

Method 1: Convert binary tree into BST(by changing the binary tree). Take each node from binary tree and insert it into BST. Then find the duplicated element.

Time Complexity: O(nlogn) average

$O(n^2)$ worst

Space Complexity: O(1)

Method 2: Convert the binary tree into BST(without destroying the original tree) and then find the duplicate element.

Time Complexity: O(nlogn)

Method 3: Find range i.e. find min and max value in the binary tree. Allot that much space and then count the element (Hashing).

Time Complexity: O(n)

Space Complexity: O(range)

Q.13 Construct a tree from inorder & preorder traversal.

Solution:-

Method 1:

```
struct tree* construct(int pre[], int in[], int pstart, int istart, int iend)
{
        struct tree *root;
        int k,i;
        if(istart>iend)
                return NULL;
        root=(struct tree*)malloc(sizeof(struct tree));
        root->data=pre[pstart];
        for(i=istart,k=0; i<=iend; i++,k++)
                if(in[i] == pre[pstart];
                        break;
        root->left=construct(pre,in,pstart+1,istart,k-1);
        root->right=construct(pre,in,pstart+k+1,i+1,iend);
```

```
        return root;
}
```

Method 2:

```
void construct(struct tree **p,int pre[], int in[], int n)
{
        int i;
        if(!n)
                return;
        (*p)=(struct tree*)malloc(sizeof(struct tree));
        (*p)->data=pre[0];
        (*p)->left=(*p)->right=NULL;
        for(i=0; i<n; i++)
                if(in[i] == pre[0])
                        break;
        construct(&((*p)->left),&pre[1],in,i);
        construct(&((*p)->right),&pre[i+1],&in[i+1],n-i-1);
}
```

Q.14 Write a program to find the postorder if preorder and inorder are given.

Solution:-

Method 1: Use one of the techniques of above question to construct the tree and then find the postorder.

Method 2: Use one of the techniques of above question but instead of creating a tree insert the element into an array. Start filling the postorder array from backward with preorder[pstart] and swap the recursive calls i.e. first call right subtree then left subtree.

Q.15 Write a program to find whether the given tree is balanced or not.

Solution:-

```
int balanced(struct tree *root)
{
        int left,right;
        if(!root)
                return 0;
        left=balanced(root->left);
        right=balanced(root->right);
        if(left == FAILURE || right == FAILURE)
                return FAILURE;
        if(abs(left – right) > 1)
                return FAILURE;
        if(left >right)
                return left+1;
        else
                return right+1;
}
```

Q.16 Find nth node of a tree according to inorder traversal of the tree.

Solution:- In this we have assumed tree indexing start from 0.

```
struct  tree* findnth(struct tree *root, int *n)
{
        struct tree *ptr;
        if(!root)
                return NULL;
        ptr=findnth(root->left,n);
        if(ptr)
                return ptr;
        if(!(*n))
                return root;
        (*n)--;
        return findnth(root->right,n);
}
```

Q.17 Write a program to find nearest common ancestor of two given nodes in the binary tree.

Solution:-

Method 1:

```
struct tree* ancestor(struct tree *root, struct tree *p, struct tree *q)
{
        struct tree *left,*right;
        if(!root || !p || !q || root==p || root==q)
                return NULL;
        if(root->left==p || root->left==q || root->right==p || root->right==q)
                return root;
        left=ancestor(root->left,p,q);
        right=ancestor(root->right,p,q);
        if(left && right)
                return root;
        if(left)
                return left;
        return right;
}
```

Method 2:

1. Take level order traversal of tree.
2. Then move the pointers p and q appropriately towards their parents, where both of them meet that is their nearest common ancestor.
   Level oder traversal can be taken in O(n).
   Finding p and q in the queue takes O(n).
   Now finding NCA…

```
parent_p=parent(p);
parent_q=parent(q);
while(parent_p != parent_q)
{
    if(parent_p>parent_q)
            p=parent_p;
```

```
            else
                    q=parent_q;
            parent_p=parent(p);
            parent_q=parent(q);
        }
        return queue[parent_p];
```

Here p,q parent_p and parent_q are indexes.
Finding NCA also takes < O(n)
Time Complexity :- O(n)
But it takes large amount of space and wastes large space if tree us sparse.

Q.18 How do you convert the inorder traversal of a binary tree into a circularly doubly link list.
Solution:-

```
struct tree* convert(struct tree *root)
{
        struct tree *alist,*blist;
        if(!root)
                return NULL;
        alist=convert(root->left);
        blist=convert(root->right);
        root->left=root->right=root;
        alist=append(alist,root);
        alist=append(alist,blist);
        return alist;
}

struct tree* append(struct tree *a, struct tree *b)
{
        struct tree *alast,*blast;
        if(a==NULL)
                return b;
        if(b==NULL)
                return a;
        alast=a->left;
        blast=b->left;
        alast->right=b;
        b->left=alast;
        a->left=blast;
        blast->right=a;
        return a;
}
```

Q.19 How do you convert binary tree inorder to a singly link list.
Solution:- Apply same approach as in above question but with some changes that are as follows:
In convert() function make following changes:

```
root->left=NULL;
 root->right=NULL;
```

Method 1: In append() function there is no alast and blast. Change it with following code:

```
tmp=a;
while(tmp->right)
        tmp=tmp->right;
tmp->right=b;
return a;
```

Method 2: Take a global or static variable which always points to either head or tail of the list. If it points at head then we will always return tail pointer & vice versa.

Method 3: Without recursion

```
struct tree* reversepathleft(struct tree *root, struct root *parent)
{
        struct tree *t, *tmp;
        while(root)
        {
                t=root;
                tmp=root->left;
                root->left=parent;
                parent=root;
                root=tmp;
        }
        return t;
}
/* part of main function where reversepathleft() is used to convert binary tree into singly linked list*/
        ptr=NULL;
        start=reversepathleft(root, NULL)
        end =0;
        while(1)
        {
                if(ptr==NULL)
                        ptr=start;
                while(ptr->right==NULL)
                {
                        if(ptr->left==NULL)
                        {
                                end=1;
                                break;
                        }
                        ptr=ptr->left;
                }
                if(end)
                        break;
                ptr->left=reversepathleft(ptr->right, ptr->left);
                ptr->right=NULL;
                ptr=ptr->left;
        }
```

Q.20 How can you remove duplicate from a binary tree(not BST).

Solution:-

Method 1:

1. Take any traversal in an array.
2. Sort the array.
3. Remove duplicate.
   Time Complexity : O(nlogn)
   Space Complexity : O(n)

Method 2:

1. Convert binary tree into BST.
2. Now duplicates can be removed easily.
   Time Complexity : O(nlogn)

Method 3:

Hashing O(n).
Traverse the tree and set the value corresponding index if value is already not present.

Q.21 Write a program to merge two binary search tree.

Solution:-

Method 1:

1. Take inorder traversal of bst1 in array1.
2. Take inorder traversal of bst2 in array2.
3. Now merge array1 and array2 into a single array.
4. Apply the following procedure :

```
struct tree* createtree(int *arr, int low, int high)
{
        if(low <= high)
        {
                mid=(low+high)/2;
                struct tree *root=(struct tree*)malloc(sizeof(struct tree));
                root->data=arr[mid];
                root->left=root->right=NULL;
                root->left=createtree(arr,low,mid-1);
                root->right=createtree(arr,mid+1,high);
                return root;
        }
        return NULL;
}
```

Time complexity:- O(n) Space Complexity:- O(n)

Method 2:

Take each node from bst1 and insert in into bst2.
Time Complexity:- O(nlogn) Space Complexity:- O(1)

```
void mergebst(struct tree **root1, struct tree *root2)
{
        struct tree *left,*right;
```

```
        if(!root2)
                return;
        left=root2->left;
        right=root2->right;
        root->left=root->right=NULL;
        insert(&root1,root2);
        mergebst(root1,left);
        mergebst(root1,right);
}
```

Q.22 How will you rearrange the nodes of a binary tree (not bst) to a max heap.

Solution:-

Method 1: For complete binary tree

Just apply heapify algorithm.

Method 2:

1. Convert binary tree into doubly linked list.
2. Then convert doubly linked list to binary tree to make complete binary tree.
3. Then apply heapify procedure.

Step 1 and Step 3 are known. Function for converting doubly linked list to binary tree is as follows:-

```
struct tree* convert(struct tree *root)
{
    struct tree *ptr, *r;
    if(root == NULL || root->right == NULL)
            return root;
    r=root;
    ptr=root->right;
    add(root);
    while(!queueempty())
    {
            root=del();
            root->left=ptr;
            if(ptr)
            {
                    add(ptr);
                    ptr=ptr->right;
                    root->right=ptr;
            }
            else
                    root->right=NULL;
            if(ptr)
            {
                    add(ptr);
                    ptr=ptr->right;
            }
    }
```

```
        return r;
    }
```

Q.23 Write a program to find preorder of a binary tree without recursion.
Solution:-

```
    void preorder(struct tree *root)
    {
        while(root)
        {
            if(root->right)
                push(root->right);
            printf("%d ",root->data);
            if(root->left)
                root=root->left;
            else
                root=pop();
        }
    }
```

Q.24 Write a program to find inorder of a binary tree without recursion.
Solution:-

```
    void inorder(struct tree *root)
    {
        while(root)
        {
            if(root->left)
            {
                push(root);
                root=root->left;
            }
            else
            {
                while(root && root->right==NULL)
                {
                    printf("%d ",root->data);
                    root=pop();
                }
                if(root)
                {
                    printf("%d ",root->data);
                    root=root->right;
                }
            }
        }
    }
```

Q.25 Write a program to find postorder of the binary tree without recursion.
Solution:-

```
void postorder(struct tree *root)
{
        struct tree *q;
        while(root)
        {
                if(root->left)
                {
                        push(root);
                        root=root->left;
                }
                else if(root->right)
                {
                        push(root);
                        root=root->right;
                }
                else
                {
                        printf("%d  ", root->data);
                        q=pop();
                        while( q &&(q->right == NULL || q->right == root))
                        {
                                root=q;
                                printf("%d ", root->data);
                                q=pop();
                        }
                        if(q)
                        {
                                root=q->right;
                                push(q);
                        }
                        else
                                root=NULL;
                }
        }
}
```

Q.26 Write a program to find the maximum depth of binary tree without recursion.

Solution:- int depth(struct tree *root)

There is a little variation in the above code of postorder for implementing depth function.

Use variables i =0 and max =0.

Whenever traverse to the left and right increment i and check it against max and whenever pop an element decrement i.

In if(root->left) put i++;

In if(root->right) put i++;

Just before while loop put if(i> max) max=i;

And in while loop put i--;

Return max.

Q.27 Write a program to delete an element from BST.
Solution:-

```
struct tree* del(struct tree *root, int data)
{
        if(root->data > data)
                root->left=del(root->left,data);
        else  if(root->data < data)
                root->right=del(root->right,data);
        else
        {
                struct tree *t=NULL;
                if(root->left && root->right)
                {
                        int m;
                        m=min(root->right);
                        root->data=m;
                        root->right=del(root->right,m);
                        return root;
                }
                if(root->left)
                        t=root->left;
                if(root->right)
                        t=root->right;
                free(root);
                return t;
        }
        return root;
}
```

Q.28 Write a program to find the median of a BST.
Solution:-
Method 1:

1. Count the number of nodes.
2. Now find the (n/2th) node according to inorder traversal.

Method 2:

1. Convert BST inorder into doubly linked list.
2. Now, find the middle of the link list.

Q.29 Write a program to find the median of a binary tree.
Solution:-
Method 1:

1. Take any traversal of binary tree into an array.
2. Sort the array.
3. Now take the n/2th element.

Method 2:

1. Convert the binary tree into BST.

2. Then find the median according to above question.

Method 3:

1. Take any traversal of binary tree in an array.
2. Now find the n/2th smallest or largest element.
3. Return this as median.
   Time Complexity : O(n) Space Compexity : O(n)

Method 4:

1. Convert the binary tree into doubly linked list.
2. Now find the n/2th smallest or largest element(as we find it in arrays).
3. Return this element as median.

Space Complexity : O(1)
Time Complexity: O(n) (not sure)

Q.30 How will you efficiently store a binary tree to a persistent file for later retrieval.
Solution:-
Method 1: Store the level order of the binary tree.
This approach is quiet inefficient because if tree is sparse then it will take a large number of NULL and this approach will store those NULL also. If n is the depth of the tree then in case of sparse tree number of NULL's stored is of the order $O(2^n)$.
e.g. Say depth of binary tree is 10 but only 10 nodes are in the tree, then this storage will store 1000's of NULL in file.

Method 2: For sparse trees, we can take 2 values corresponding to each node. First to store the array index and second to store the array value. (think it again)

Method 3: Store inorder and preorder or inorder and postorder.
In case of complete binary tree method 1 is best.

Q.31 You are having a binary tree. Initially all the nodes ar unlocked. You will have a set of input representing the nodes given and you have to check those nodes can be locked or node. A node can be locked only if all the following three conditions hold:
a) If it is already unlocked.
b) If all its parents are unlocked.
c) If all its children are unlocked.
What will be the suitable data structure to represent the tree? How efficiently can we check these three operations.
Solution:-

```
struct tree
{
        int data;
        struct tree *left, *right;
        int lock;
}
```

Value of lock in struct field is 0 when unlocked and 1 when locked.
We can include parent field in structure to make it more efficicent.

Q.32 How can you merge two min or max heaps?
Solution:-
Method 1: Take an array that can accommodate both the arrays (both heaps). Now apply heapify procedure.

Method 2: This method is applicable only if both the heaps depth differ by 1 and smaller one heap is a full heap (full binary tree). Take a dummy node and make it root and bigger heap is left child and smaller heap is right child.

Q.33 Write a program to implement AVL tree.
Solution:-

```
struct node* insert(struct node *root, int data)
{
        if(root==NULL)
        {
                root=(struct node*)malloc(sizeof(struct node));
                root->data=data;
                root->right=root->left=NULL;
        }
        else if(root->data > data)
        {
                root->left=insert(root->left, data);
                if(abs(height(root->left) – height(root->right))==2)
                {
                        if(data < root->left->data)
                                root = leftrotation(root);
                        else
                                root=doublelr(root);
                }
        }
        else if(root->data < data)
        {
                root->right=insert(root->right, data);
                if(abs(height(root->left) – height(root->right))==2)
                {
                        if(data > root->right->data)
                                root = rightrotation(root);
                        else
                                root=doublerl(root);
                }
        }
        return root;
}

struct node* leftrotation(struct node *t)
{
        struct node *temp;
```

```
        temp=t->left;
        t->left=temp->right;
        temp->right=t;
        return temp;
}

struct node* rightrotation(struct node *t)
{
        struct node *temp;
        temp=t->right;
        t->right=temp->left;
        temp->left=t;
        return temp;
}

struct node * doublelr(struct node *t)
{
        t->left=rightrotation(t->left);
        return leftrotation(t);
}

struct node * doublerl(struct node *t)
{
        t->right=leftrotation(t->right);
        return rightrotation(t);
}
```

Q.34 Given a binary tree, write a program to balance a tree.

Solution:- Pseudocode is :-

```
balance(struct node *root)
{
        if(!root)
                return;
        balance(root->left);
        balance(root->right);
        if(height(root->left) – height(root->right) >1)
        {
                if(height(root->left->left) – height(root->left->right) < 0)
                        doublelr(root);
                else
                        leftrotation(root);
        }
        if(height(root->right) – height(root->left) >1)
        {
                if(height(root->right->left) – height(root->right->right) > 0)
                        doublerl(root);
                else
```

```
                    rightrotation(root);
            }
        }
```

Insert return statements and check for NULLs for running code.

Q.35 Given a binary tree, write a function which will return level of the tree which has maximum number of nodes. Optimize it for a sparse tree.

Solution:-

1. For full binary tree : return the depth of the tree.
2. For complete binary tree : return either depth or depth-1 after checking number of nodes at depth and depth-1.
3. For sparse tree : Use one of the following methods.

Method 1:

```
static int hash[MAXLVL]; //MAXLVL is height of tree
int max=0;

int max_nodes(struct tree *root, int level)
{
        if(root)
        {
                hash[level]++;
                if(hash[level] > hash[max])
                        max=level;
                max_nodes(root->left, level+1);
                max_nodes(root->right, level+1);
        }
        return max;
}
```

Time complexity: O(n)
Space complexity: O(logn)

Method 2:

```
int maxlevel=0,maxno_of_nodes=0,nodes_innextlvl=0,nodes_incurlvl=0;
struct tree *queue[MAXNODES];

void maxnodes(struct tree *root)
{
        int cnt=0,level=0;
        if(root==NULL)
                return;
        add(root);
        nodes_incurlvl=1;
        maxno_of_nodes=1;
        while(!queue.empty())
        {
                root=pop();
                if(cnt == nodes_incurlvl)
```

```
                {
                        level++;
                        if(nodes_innextlvl > maxno_of_nodes)
                        {
                                maxno_of_nodes = nodes_innextlvl;
                                maxlevel=level;
                        }
                        nodes_incurlvl=nodes_innextlvl;
                        nodes_innextlvl=0;
                        cnt=0;
                }
                cnt++;
                if(root->left)
                {
                        add(root->left);
                        nodes_innextlvl++;
                }
                if(root->right)
                {
                        add(root->right);
                        nodes_innextlvl++;
                }
        }
}
```

Q.36 Write a program to find whether the given tree is complete binary tree or not.
Solution:- For this refer to above question. Apply one of the methods and check the no. of nodes in each level is 2^{level} and in the last level nodes must be present from left to right i.e. in the last level there should be no NULL node between any two adjacent nodes.

Q.37 Write a program to make mirror image of a given binary tree.
Solution:-
Method 1: When new tree is to be created

```
struct tree* mirror(struct tree *root)
{
    struct tree *t;
    if(!root)
            return NULL;
    t=(struct tree*)malloc(sizeof(struct tree));
    t->data=root->data;
    t->left=mirror(root->right);
    t->right=mirror(root->left);
    return t;
}
```

Method 2: When the given tree is to changed.

```
void mirror(struct tree *root)
{
```

```
        struct tree *t;
        if(!root)
                return;
        t=root->left;
        root->left=root->right;
        root->right=t;
        mirror(root->left);
        mirror(root->right);
    }
```

Q.38 Write a program to find loop in a binary tree.

Solution:-

Method 1:

1. Change tree structure i.e. include a field visited.
2. Initially make all visited field false.
3. Now make any traversal. If we reach to any node whose visited field is true then there is a loop.

Method 2: Do address hashing. Whenever there is a hit then there is a loop.

Time Complexity : O(n)

Space Complexity : O(n)

Method 3:

```
        int find_loop(struct tree *ptr1, struct tree *ptr2)
        {
                int i=0, j=0, k=0, l=0, m=0, n=0, o=0, p=0;
                if(ptr1==NULL || ptr2==NULL)
                        return 0;
                if(ptr1==ptr2)
                        return 1;
                if(ptr2->left)
                {
                        if((i=find_loop(ptr1->left,ptr2->left->left)>0)
                                return 1;
                        if((j=find_loop(ptr1->left,ptr2->left->right)>0)
                                return 1;
                }
                if(ptr2->right)
                {
                        if((k=find_loop(ptr1->left,ptr2->right->left)>0)
                                return 1;
                        if((l=find_loop(ptr1->left,ptr2->right->right)>0)
                                return 1;
                }
                if(ptr2->left)
                {
                        if((m=find_loop(ptr1->right,ptr2->left->left)>0)
                                return 1;
```

```
                if((n=find_loop(ptr1->right,ptr2->left->right)>0)
                        return 1;
        }
        if(ptr2->right)
        {
                if((o=find_loop(ptr1->right,ptr2->right->left)>0)
                        return 1;
                if((p=find_loop(ptr1->right,ptr2->right->right)>0)
                        return 1;
        }
        return i || j || k || l || m || n || o || p;
}
```

Chapter 7: MISCELLANEOUS QUESTIONS

Q.1 Find sum of digits of a number till the sum is only a single digit number i.e. if sum of digits of a number is greater than 9 then again find sum of digits of the resultant sum and do it until sum is single digit number. e.g. 4298, sum of its digit is 23 and again sum of digits of 23 is 5. So your answer is 5.

Solution: Suppose number be the **num** whose sum of digits we have to find. For this do the following

```
(num%9)?(num%9):(num?9:0)
```

Explanation:

```
4298=4*1000+2*100+9*10+8
    =4*999+4+2*99+2+9*9+9+8
    =9[111*4+11*2+1*9]+4+2+9+8
    =9*k+23
    =9*k+9*m+5 [on similarly breaking 23 as we have done for 4298]
```

So taking modulous with 9 will give 5.

Special case occurs when number is 9*k. In this case sum of digits is always 9 as you can see from table of 9. But modulous of 9*k with 9 will give 0 as well as when number is 0 modulous with 9 will give 0. So in such case check required is that if number is non zero and multiple of 9 then print 9 and if number is 0 then print 0.

Q.2 Infinite bit stream is coming (one bit at a time). At a given time tell whether number is divisible by 3 or not.

Solution: In method 1 and 2, sum takes memory, processing and bit can go out of bound.

Method 1:

```
sum=0;
while(bit stream is coming)
{
        sum =sum*2+bit;
        if(sum%3==0)
                number divisible;
        else
                not divisible;
}
```

Method 2:

```
sum=0;
while(bit stream is coming)
{
        sum=sum<<1+1*(current bit);
        if(sum%3==0)
                printf("divisible\n");
```

```
        else
                printf("not divisible");
}
```

Method 3: Use automata theory. In this method change state according to the state table and bit coming. So with constant storage able to process.

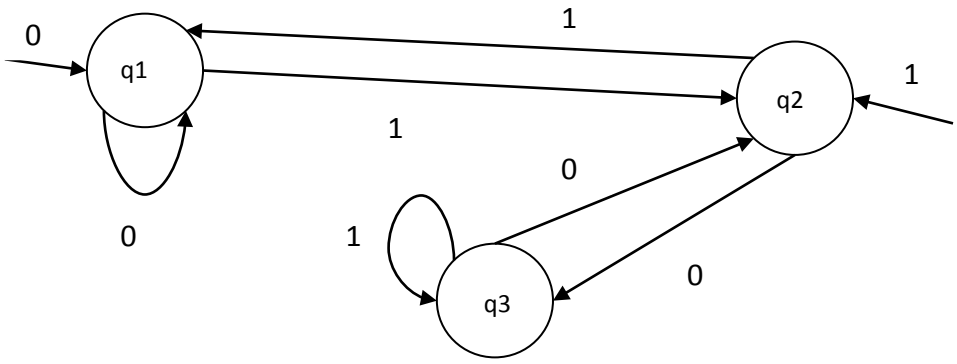


Table for the State transition on the basis of input is as follows:-
q1 is the state when starting bit is 0 or when number is divisible by 3.
q2 is the state when starting bit is 1 or on dividing the number by 3 gives 1 as remainder.
q3 is the state where on dividing the number by 3 gives 2 as remainder

	0	1
q1	q1	q2
q2	q3	q1
q3	q2	q3

So choose the starting state according to first bit of bit stream i.e. q1 when first bit is 0 and q2 when first bit is 1. After that just change the states according to the above transition table and input bit coming.

e.g. input bit stream be 10110100
since first bit is 1 so start state will be q2. Now the transition of states will be as follows
(1)->q2 (0)->q3 (1)->q3 (1)->q3 (0)->q2 (1)->q1 (0)->q1 (0)->q1
so final state is q1 so number is divisible by 3.

Q.3 Give a one line C expression to test whether a number is power of 2 or not.
Solution:-

```
int power_of_2(int n)
{
        return(!(n&(n-1)) && n);
}
```

Q.4 Count number of 1's in the bit representation of an integer.
Solution:-
Method 1:

```
int count_1(int n)
{
        int count=0;
        while(n)
        {
                count+=(n&0X1);
                n=n>>1;
        }
        return count;
}
```

Method 2:

```
int count_1(int n)
{
        int c=0;
        while(n)
        {
                c++;
                n=n&(n-1);
        }
        return c;
}
```

Q.5 Write a program to compute sum of first n terms of the series S.
S=1-3+5-7+9
Solution:-

n	Series	Sum
1	1	+1
2	1-3	-2
3	1-3+5	+3
4	1-3+5-7	-4
5	1-3+5-7+9	+5
6	1-3+5-7+9-11	-6

```
int series(int n)
{
        if(n%2==0)
                return –n;
        else
                return n;
}
```

Q.6 Write a function to count number of bits set in a floating point number.
Solution:-

```
int count(float num)
{
        int j=0,n=sizeof(float)/sizeof(char);
        char *p=(char *)&num;
        for(int i=0; i<n; i++)
        {
                while(*p)
                {
                        j++;
                        *p=(*p) & (*p)-1;
                }
                p++;
        }
        return j;
}
```

Q.7 There are N steps, how many possible ways are there to climb those steps if you can take either 1 or 2 steps at a time.

Solution:-

- When no. of step is 1, No. of ways=1.
- When no. of steps are 2, No. of ways=2 [(1+1),(2)].
- When no. of steps are 3, No. of ways=3 [(1+1+1),(1+2),(2+1)].
- When no. of steps are 4, No. of ways=5 [(1+1+1+1),(1+1+2),(1+2+1),(2+1+1),(2+2)].
- When no. of steps are 5, No. of ways=8 [(1+1+1+1+1),(1+1+1+2),(1+1+2+1),(1+2+1+1),(2+2+1),(2+1+2),(1+2+2),(2+1+1+1)].

For N steps, No. of ways f(n)=f(n-1)+f(n-2).

Q.8 Given a number n. If n is even then n=n/2 else if n is odd then n=3*n+1.
If you keep on doing above said operation n will converge to 1 at some point. You are given a set of numbers (say 65 to 125). Give an efficient solution to find out which number converges to 1 quickest.

Solution:- Any number which is power of 2 will converge quickest. We will try to find out that no:-

1. If given range does not contain any power of 2 number, then we will take the 2 power no. just above the range. In this case 65-125 doesn't contain any 2-power number, so we will take 128.
2. Now from any no. present in range we have to make 128 because we have to reach 128. (128-1)/3=127/3=42 but 42 is not present in range.
3. Multiply 128 by 2
   128*2=256.
   Now do step 2 again (256-1)/3=85.
   Present in range, so 85 is the required number.
   Do this until we get the number.

Q.9 Find whether the n^{th} bit of a given number is on or off.

Solution:- (1<<n)&number

If above expression is true it means bit is on else bit is off.

Q.10 Write a function which simply returns 0X00000000 if input value is 0 else 0XFFFFFFFF. Condition is no branching , no looping and no ternary operators.

Solution:-

Method 1:

```
int bit(int n)
{
        return (!n-1);
}
```

Method 2:

```
int bit(int n)
{
        return (!n + (n|~n));
}
```

Q.11 Find whether the machine is big endian or little endian.

Solution:-

```
int biglittle()
{
        int i=5;
        if(*(char*)&i==5)
                return 1;       //little endian
        return 0;               //big endian
}
```

Q.12 Given n +ve numbers find max power of 2 which divides the GCD of these n numbers.

Solution:- Take the OR of n numbers because we have to find the max power of 2 which divides the GCD.

Let that number be x. If x divides GCD of a_1,a_2,a_3……,a_n, then it must divide each a_1 to a_n. So by taking OR of these n numbers we will get the corresponding bits set. Take least bit set in result of OR of these n numbers. This is the maximum power of 2 which divides the GCD of those numbers.

e.g. Let there are three numbers 12,20,40.

12 - 001100
20 - 010100
40 - 101000

Taking OR of the above 3 numbers. Result is 111100. Least bit set in the result correspond to 4. So 4 is the maximum number which is power of 2 and will divide GCD of 12,20,40.

Q.13 Given only putchar(no sprint, itoa etc). Write a routine print that prints out an unsigned long in decimal.

Solution:-

Method 1:

```
void print(long num)
{
        long s=1,t=num;
        if(!t)
        {
```

```
                putchar('0');
                return;
        }
        if(num<0)
        {
                putchar('-');
                t=-t;
        }
        while(t)
        {
                t=t/10;
                s=s*10;
        }
        s=s/10;
        t=num;
        while(s)
        {
                putchar(t/s+'0');
                t=t%s;
                s=s/10;
        }
}
```

Method 2:

```
void print(long num)
{
        if(num<0)
        {
        putchar('-');
        num=-num;
        }
        if(!num)
                putchar('0');
        if(num/10)
                print(num/10);
        putchar(num%10+'0');
}
```

Q.14 Write a program to generate all the subsets of a set.

Solution:-

```
#define MAX 1000
struct prob
{
        int arr[MAX];
        int start,end;
}s,temp;

temp.end=temp.start=s.start=0;
```

```
void subset(int n)
{
        int i;
        for(i=temp.start; i<temp.end; i++)
                printf("%d\t",temp.arr[i]);
        for(i=n; i<s.end; i++)
        {
                temp.arr[temp.end++]=s.arr[i];
                subset(i+1);
                temp.end--;
        }
}
```

s.end=number of elements in the set and initial call will be subset(0);

Q.15 Write a code to compute sign of an integer.

Solution:-

```
int  sign(int num)
{
        return( +1 | num >> sizeof(int)*8 -1);
}
```

This will return +1 or -1. +1 when the number is positive and -1 when the number is negative. When we right shift, the sign bit is copied instead of 0. Therefore in case of negative number we will get -1 on right shifting.

Q.16 Set the most significant bit of a unsigned int to 0.

Solution:-

```
unsigned int set(unsigned int x)
{
        unsigned int temp=1;
        temp=temp<<(sizeof(int)*8-1);
        temp=~temp;
        return x & temp;
}
```

Q.17 Find minimum and maximum of two numbers:

i. without branching.
ii. without relational operators.
iii.both i & ii.

Solution:-

i. Without branching

```
Let x and y are two numbers.
Min = y+((x-y) & -(x<y));
Max =  x-((x-y) & -(x<y));
```

ii. Without relational operators

```
if((x-y)>>(sizeof(int)*8-1))
printf("maximum %d minimum %d",y,x);
```

```
        else
        printf("maximum %d minimum %d",x,y);
```

iii. Both i & ii

```
Min = y+((x-y) & (x-y)>>(sizeof(int)*8-1));
Max = x-((x-y) & (x-y)>>(sizeof(int)*8-1));
```

Q.18 Find the next highest power of 2 for any given number for 32 bit integer.

Solution:-

Method 1: Let the given number be n. The steps for finding the next highest power of 2 are as follows-

```
n--;
n = n | n>>1;
n = n | n>>2;
n = n | n>>4;
n = n | n>>8;
n = n | n>>16;
n++;
```

Method 2: Let the given number be n. The steps for finding the next highest power of 2 are as follows-

```
if(n>1)
{
        float f = (float)n;
        unsigned int t = 1<<((*(unsigned int*)&f>>23)-0x7f);
        r = t<<(t<n);
}
else
        r=1;
```

Q.19 Given time in HH:MM format, give a formula to determine the angle between the needles.

Solution:-

Angle between needles=(30*HH)-(5.5*MM).

If angle > 180 then angle = angle-180, else angle remains same.

Q.20 Write a program to print numbers from1 to 99 or any number without using loop.

Solution:-

Method 1: Recursion

```
void print(int n)
{
        if(n>99)
                return;
        printf("%d\n",n);
        print(n+1);
}
```

Method 2: using labels

```
void print(int n)
{
        int i=1;
        Loop: printf("%d\n",i);
```

```
        if(i<=n)
        {
                i++;
                goto Loop;
        }
}
```

Method 3:

```
void donothing(int x, int n)
{
        return;
}

typedef void (*funcp) (int x, int n);

void print(int x, int n)
{
        printf("%d\n",x);
        x++;
        (*fptr[x>n])(x,n);
}

funcp fptr[2]={print,donothing};

void main()
{
        print(1,99);
}
```

Q.21 Convert little endian integer to big endian.

Solution:-

Method 1:

```
int convert(int x)
{
        int x1,x2,x3,x4;
        x1=x&0xff;
        x2=x&0xff00;
        x3=x&0xff0000;
        x4=x&0xff000000;
        return (x1<<24 + x2<<8 + x3>>8 + x4>>24);
}
```

Method 2:

```
int convert(int *x)
{
        char *p[4],*t;
        p[0]=(char *)x;
        p[1]=p[0]+1;
```

```
        p[2]=p[1]+1;
        p[3]=p[2]+1;
        *t=*(p[0]);
        *(p[0])=*(p[3]);
        *(p[3])=*t;
        *t=*(p[1]);
        *(p[1])=*(p[2]);
        *(p[2])=*t;
}
```

Q.22 Write a program to find GCD of two numbers.

Solution:-

```
int gcd(int i, int j)
{
        while(i!=j)
        {
                if(i>j)
                        i=i-j;
                else
                        j=j-i;
        }
        return i;
}
```

Q.23 Write a program to count no. of set bits in a 32-bit integer without any loop.

Solution:-

```
int count(int x)
{
        int y;
        y=(x & 0XAAAAAAAA) >> 1 + (x & 0X55555555);
        y=(y & 0XCCCCCCCC) >> 2 + (y & 0X33333333);
        y=(y & 0XF0F0F0F0) >> 4 + (y & 0X0F0F0F0F);
        y=(y & 0XFF00FF00) >> 8 + (y & 0X00FF00FF);
        y=y >> 16 + (y & 0X0000FFFF);
        return y;
}
```

After first statement each pair of two bits of y will contain binary representation of the number of bits as in original pair.
e.g. 01001110 => 01001001
Similarly each statement afterwards will do this thing.

Q.24 Compute modulus division by 1<<s without using division operator.

Solution:-

```
const unsigned int n;                   // numerator
const unsigned int s;
const unsigned int d = 1U << s;    // So d will be one of: 1, 2, 4, 8, 16, 32, ...
unsigned int m;                         // m will be n % d
m = n & (d - 1);
```

Q.25 Compute modulus division by ((1<<s) – 1) without using division operator.

Solution:-

```
unsigned int n;                          // numerator
const unsigned int s;                    // s > 0
const unsigned int d = (1 << s) - 1;     // so d is either 1, 3, 7, 15, 31, ...).
unsigned int m;                          // n % d goes here.
for (m = n; n > d; n = m)
{
        for (m = 0; n; n >>= s)
        {
                m += n & d;
        }
}
// Now m is a value from 0 to d, but since with modulus division
// we want m to be 0 when it is d.
m = m == d ? 0 : m;
```

Q.26 Reverse the bits of an unsigned interger.

Solution:-

```
unsigned int reverse(unsigned int x)
{
        x=(x)>>16 | (x & 0x0000ffff)<<16;
        x=(x & 0xff00ff00)>>8 | (x & 0x00ff00ff)<<8;
        x=(x & 0xf0f0f0f0)>>4 | (x & 0x0f0f0f0f)<<4;
        x=(x & 0xcccccccc)>>2| (x & 0x33333333)<<2;
        x=(x & 0xaaaaaaaa)>>1 | (x & 0x55555555)<<1;
        return x;
}
```

Q.27 Find the absolute value of a number

i. without branching

ii. without branching and relational operator.

Solution:-

(i)

```
int absolute(int a)
{
        int sign;
        sign=(a>0)-(a<0);
        return (sign * a);
}
```

(ii)

```
int absolute(int a)
{
        int sign;
```

```
        sign=(a^(a>>(sizeof(int)*8-1))-(a>>(sizeof(int)*8-1)));
        return (sign > 0 ? a : -a );
}
```

Q.28 Write a routine to draw a circle without using any floating point computation.

Solution:-

```
void circle(int xcenter, int ycenter, int radius)
{
        int p,x,y;
        p=1-radius;
        x=0;
        y=radius;
        drawpt(xcenter, ycenter,x,y);
        while(x < y)
        {
                x++;
                if(p<0)
                        p=p+2*x+1;
                else
                {
                        y- -;
                        p=p+2*(x-y)+1;
                }
                drawpt(xcenter, ycenter, x, y);
        }
}
```

Q.29 Find the $\log_2$num.

Solution:-

Method 1: Brute Force Approach

```
count=0;
while(num=num>>1)
        count++;
```

Method 2:

```
unsigned int b[]= {0x2, 0xc, 0xf0, 0xff00, oxffff0000};
unsigned int s[]= {1,2,4,8,16};
unsigned int r=0;
for(i=4; i>=0; i--)
{
        if( num & b[i])
        {
                num = num>>s[i];
                r = r | s[i];
        }
}
```

Method 3:

```
union
{
 unsigned int u[2];
double d;
} t;
t.u[BYTE_ORDER == LITTLE_ENDIAN] = 0x43300000;
t.u[BYTE_ORDER != LITTLE_ENDIAN] = num;
t.d -=4503599627370496.0  //2^52
r = (t.u[BYTE_ORDER == LITTLE_ENDIAN] >> 20) – 0x3ff;
```

Q.30 Given a file with 10^7 numbers in the range 1 to 10^7. Total RAM available is 1 MB. How will you sort the numbers.

Solution:-

Method 1:

Assumption :- all numbers are unique.
As size of RAM < size of all numbers.
Use bit vector to sort the numbers.
1MB=2^{10} * 2^{10} bytes = 2^{20} bytes.
4 bytes (one integer) can be used to sort first 32 integers.
So, 2^{20} bytes can sort first $8*2^{20}$ numbers.
Since $8*10^6$ is approximately equal to $8*2^{20}$, so numbers onwards ($8*10^6$ – 10^7) are sorted in second pass.

Method 2:

1. Bring all the single digit numbers then sort and write them to file.
2. Now increase the digit by 1 and do step 1 again.
   After 7^{th} iteration whole data is sorted.

Q.31 Write a program to generate fibbonacci number.

Solution:-

Method 1:

```
int fib[MAX];
fib[0]=0;
fib[1]=1;
for(int i=2; i<MAX; i++)
        fib[i]=fib[i-1]+fib[i-2];
```

Method 2: Recursion

```
if( n <= 2)
        return 1;
else
        return fib(n-1)+fib(n-2);
```

Method 3: Iterative

```
a=0;
b=1;
i=2;
while(i<n)
```

```
{
        c=a+b;
        printf("%d ",c);
        a=b;
        b=c;
        i++;
}
```

Method 4: Fast method iterative

```
a=0;
b=1;
i=0;
while(i<n)
{
        a=a+b;
        printf("%d ",a);
        b=a+b;
        printf("%d ",b);
        i=i+2;
}
```

Q.32 Write a program to find whether a given no. is a Fibonacci no. or not.

Solution:-

Method 1:

1. If $5n^2+4$ or $5n^2-4$ is a perfect square then given number is Fibonacci no.
   Function to find out whether $5n^2+4$ or $5n^2-4$ is a perfect square or not is :

```
int perfect_square(int n)
{
    int sub=1;
    while(n > 0)
    {
            n=n-sub;
            sub=sub+2;
    }
    if(n<0)
            return 0; //failure
    return 1; //success
}
```

Method 2: Generate Fibonacci series till the no. or just one no. greater than the given no.

Q.33 Write a program to find nth Fibonacci number.

Solution:-

Method 1 : f(n) = (1/sqrt(5)) * (((1+sqrt(5))/2) ^ n - ((1-sqrt(5))/2) ^ n)

Method 2:

```
int M[2][2]={{1,0},{0,1}};
int A[2][2]={{1,1},{1,0}};
int C[2][2]={{0,0},{0,0}};
// Recursive function with divide and conquer strategy
void matMul(int n)
{
if(n > 1)
{
matMul(n/2);
mulM(0); // M * M
}
if(n%2!=0)
{
mulM(1); // M * {{1,1}{1,0}}
}
}
// Function which does some basic matrix multiplication.
void mulM(int m)
{
int i,j,k;
if(m==0)
{
// C = M * M
for(i=0;i < 2;i++)
for(j=0;j < 2;j++)
{
C[i][j]=0;
for(k=0;k < 2;k++)
C[i][j]+=M[i][k]*M[k][j];
}
}
else
{
// C = M * {{1,1}{1,0}}
for(i=0;i < 2;i++)
for(j=0;j < 2;j++)
{
C[i][j]=0;
for(k=0;k < 2;k++)
C[i][j]+=A[i][k]*M[k][j];
}
}
// Copy back the temporary matrix in the original matrix M
for(i=0;i < 2;i++)
for(j=0;j < 2;j++)
{
M[i][j]=C[i][j];
```

```
}
}
```

Now M[0][0] will have nth Fibonacci number. Initially function will be called as matMul(n-1).
Logic behind this is

$$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n = \begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix}.$$

and power of matrix can be calculated in O(log n) time by matMul() function.

Method 3:
Find the binary equivalent of number.
$f_{2n+1} = (f_{n+1})^2 + 2*f_n*f_{n+1}$
$f_{2n} = (f_n)^2 + (f_{n+1})^2$
If digit of binary equivalent is 1 then
$f_n = f_{2n+1}$
$f_{n+1} = f_{2n+2}$ where $f_{2n+2} = f_{2n+1} + f_{2n}$
If digit of binary equivalent is 0 then
$f_n = f_{2n}$
$f_{n+1} = f_{2n+1}$

```
int fib(int n)
{
int fn=0, fnp1=1, a[32], i=0, f2n, f2np1;
while(n)
{
        a[i++]=n%2;
        n=n/2;
}
i=i-2; /*i is one more than no. of binary digit and we don't need last one.*/
while(i)
{
        f2n=fn*fn + fnp1*fnp1;
        f2np1=fnp1*fnp1 + 2*fn*fnp1;
        if(a[i])
        {
                fn=f2np1;
                fnp1=f2np1+f2n;
        }
        else
        {
                fn=f2n;
                fnp1=f2np1;
        }
        i--;
}
return fn;
}
```

In main(), to find n^{th} Fibonacci number, fib() will be called as fib(2*n);
For e.g. to find 1^{st} Fibonacci number fib(2) will be called.

Q.34 Write a program to implement 8-queen problem.
Solution:-
The 8 queens problem is a classic problem using the chess board. This problem is to place 8 queens on the chess board so that they do not attack each other horizontally, vertically or diagonally. Suppose we have an array t[8] which keeps track of which column is occupied in which row of the chess board. That is, if t[1]==5, then it means that the queen has been placed in the fifth column of the first row. We need to couple the backtracking algorithm with a procedure that checks whether the tuple is completable or not, i.e. to check that the next placed queen 'i' is not menaced by any of the already placed 'j' (j < i):
Two queens are in the same column if t[i]=t[j]
Two queens are in the same major diagonal if (t[i]-t[j])=(i-j)
Two queens are in the same minor diagonal if (t[j]-t[i])=(i-j)
empty() function is used to check whether the given location is empty or not.

```
static int t[10]={-1};
void queen(int i)
{
        for(t[i]=1; t[i]<=8; t[i]++)
        {
                if(empty(i))
                {
                        if(i==8)
                        {
                                printsolution();
                                /* If this exit is commented, it will show ALL possible
                                combinations */
                                exit(0);
                        }
                        else
                                queen(i+1);
                }
        }
}

int empty(int i)
{
        int j=1;
        while(t[i]!=t[j] && abs(t[i]-t[j])!=(i-j) && j<8)
                j++;
        return (i == j) ? 1 : 0;
}

void print_solution()
{
        int i;
        for(i=1; i<=8; i++)
```

```
                printf("\n%d Queen is at %d",i,t[i]);
        }
```

Initial call is queen(1).

Q.35 All numbers are integers. You do not know the size of integers. They can be infinitely long so you can't count on truncating at any point. There are no comparisons allowed. Operations that can be performed are available. No negative numbers also. Operations are :-
a) set a variable to 0.
b) set a variable = another variable.
c) you can increment a variable by 1 (post increment)
d) you can loop. So if you do loop(v) and v=10, loop will execute 10 times. Value of v can be changed inside loop but it will execute without changing the no. of times.

i. Write a function that decrement by 1.
ii. Write a function that subtract one variable from other.
iii. Write a function to multiply two numbers.
iv. Write a function to divide two numbers.

Solution:-

i.
```
int decrement(int n)
{
        int i=0, j=0;
        loop(n)
        {
                j=i;
                i++;
        }
        return j;
}
```

ii.
```
int subtract(int a, int b)
{
        int result =a;
        loop(b)
        {
                result=decrement(result);
        }
        return result;
}
```

iii.
```
int multiply(int a, int b)
{
        int count =0;
        loop(a)
        {
                loop(b)
                {
                        count++;
```

```
                    }
                }
                return count;
        }
```

iv.
```
        int divide(int a, int b)
        {
                int c=subtract(a,b);
                int  d=0;
                loop( c )
                        return 1+ divide(c,b);
                c=subtract(b,a);
                loop( c)
                        return d;
                d++;
                return d;
        }
```

Q.36 Write a function which returns the value of 7n/8 without using * and / operators.
Solution:-

```
int fun(int n)
{
        return n - (n >> 3);
}
```

Q.37 Write a program to find whether a given number is multiple of 7 or not.
Solution:-
Method 1: Using modulous operator.

Method 2:

```
unsigned int calcRemainder(unsigned int n)
{
   unsigned int sum = 0;
   while(1)
   {
        sum = 0;
        while(n)
        {
                sum += n&7;
                n = n >> 3;
        }
        if(sum>7)
                n = sum;
        else if(sum==7)
                return 0;
        else
                return sum;
   }
```

```
}
```

Above function calcRemainder() returns 0 when number is 0 or multiple of 7 otherwise returns non zero value from 1 to 6 i.e. remainder on dividing the number by 7.

Q.38 You are given an infinite number of cookies boxes containing either 6,9, or 400 cookies. You are allowed to use these boxes in any combination as desired. What is the max number of cookies that you can't give out using the above boxes.

Solution:- 803 is the maximum number of cookies that you can't give out. After 803 all of the number of cookies are possible to give out.

Explanation:

1	2	3
4	5	6
7	8	9
10	11	12
13	14	15

All multiples of 3 can be made from 6 & 9 till less than 400 except 3.

1	2	3
4	5	6
7	8	9
		.
		.
397	398	399
400	401	402
		.
		.
796	797	798
799	800	801

After that all combination of 400 + multiple of 3 are possible except 403.

799	800	801
802	803	804
805	806	807
808	809	810
811	812	813

After that all combination of 800 + multiple of 3 are possible except 803.

Q.39 n persons are standing in a circle. The first person is given a gun. He shoots second and handover it to the 3rd person. 3rd shoots the 4th one and hands it to 5th and so on. Write an algorithm to find out which person will be alive at the end.

Solution:-

Method 1: Make a circular link list and delete each alternate node. In the end one node left is the alive person.

Method 2: This is Josephus Problem with m =2. Recurrence equation for this is:

F(2n)=2F(n)-1

F(2n+1)=2F(n)+1

Solution is 1+2*n – 2^(1+floor(log n)).

Method 3: Take circular left shift of n (no. of persons). This will give us answer.
e.g. Let number of persons n=10.
Binary representation of n i.e. $10=(1010)_2$.
After circular left shift of $(1010)_2$, we get $(0101)_2$.
Since $(0101)_2$ is equal to 5 which is the person alive at the end.

Q.40 Write a program to solve the maze problem.
Solution:-
Pseudocode:

```
While(!stackempty() && !found)
{
        t=pop();
        row=t.row, col=t.col, dir=t.dir;
        while(dir<8 && !found)
        {
                nextrow=row+mov[dir].vert;
                nextcol=col+mov[dir].hrz;
                if(nextrow==EXITROW && nextcol==EXITCOL)
                        found=1;
                else if(maze[nextrow][nextcol]==0 && mark[nextrow][nextcol]==0)
                {
                        dir++;
                        push(row,col,dir);
                        mark[nextrow][nextcol]=1;
                        dir=0;
                        row=nextrow;
                        col=nextcol;
                }
                else
                        dir++;
        }
}
If(found)
        Print path
Else
        No path exists
```

C code :

```
#define SIZE 15
void main()
{
int maze[SIZE][SIZE],mark[SIZE][SIZE],stack[SIZE][3];
static int move[8][2]={-1,0,-1,1,0,1,1,1,1,0,1,-1,0,-1,-1,-1};
int i,j,m,n,top,mov,g,h;
printf("enter size");
scanf("%d%d",&m,&n);
for(i=1;i<=m;i++)
{
```

```
for(j=1;j<=n;j++)
{
scanf("%d",&maze[i][j]);
}
}
for(i=0;i<=n+1;i++)
maze[0][i]=1;
for(i=0;i<=n+1;i++)
maze[m+1][i]=1;
for(i=0;i<=m+1;i++)
maze[i][0]=1;
for(i=0;i<=m+1;i++)
maze[i][n+1]=1;
for(i=1;i<=m;i++)
{
for(j=1;j<=n;j++)
{
mark[i][j]=0;
}
}
mark[1][1]=1;
stack[0][0]=1;
stack[0][1]=1;
stack[0][2]=2;
top=1;
while(top!=0)
{
i=stack[0][0];
j=stack[0][1];
mov=stack[0][2];
top=top-1;
while(mov<=7)
{
g=i+move[mov][0];
h=j+move[mov][1];
if(mark[g][h]==0&&maze[g][h]==0)
{
mark[g][h]=1;
top++;
stack[top][0]=i;
stack[top][1]=j;
mov=-1;
i=g;j=h;
}
mov=mov+1;
if(g==m&&h==n)
{
printf("path is");
```

```
for(i=1;i<=top;i++)
printf("%d %d",stack[i][0],stack[i][1]);
printf("%d %d",m,n);
exit(0);
}
}
}
}
```

Q.41 You have two files F1 and F2. F1 is of size X TB and F2 is of size 100X TB. F1 has records of type (id, address) and F2 has records of type (id, paycheck_no). In F1, all records have unique "id" no. while in F2 there may be multiple records with same "id" no. Each of the "id" in F2 appears in F1. Combine the two files, F1 and F2 into F3, with records like (id, address, paycheck_no) where ids can again be similar for two records. Obviously the no. of records in F3 is going to be equal to no. of records in F2.

Q.42 Given 5 numbers x_1, x_2, x_3, x_4 and x_5. In how many comparisons you will find median.

Solution:-

Method 1: Make a min heap. In worst case i.e. when array is sorted in ascending order 8 comparisons are needed.

Method 2: Number of comparisons needed is 6.

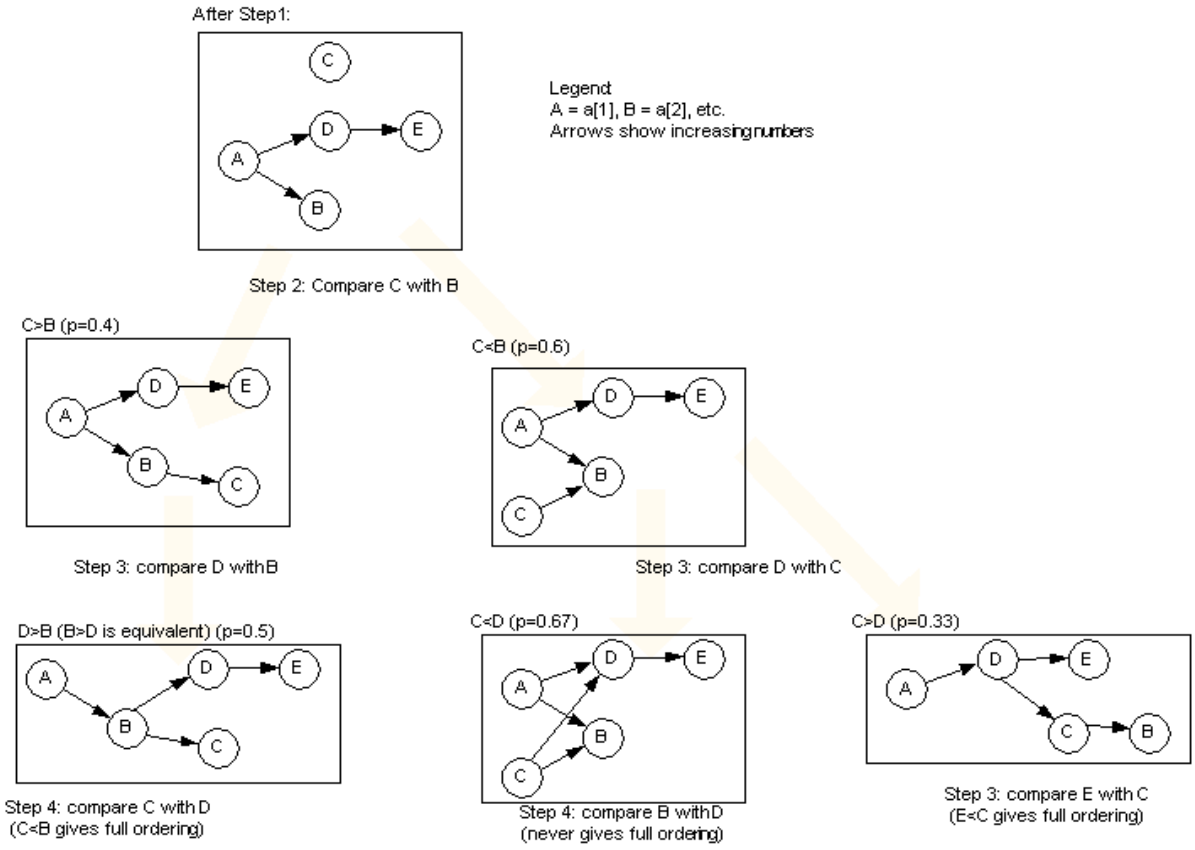
i. Let the 5 given numbers are stored in an array such that a[1]=x_1, a[2]=x_2, a[3]=x_3, a[4]=x_4 and a[5]=x_5.

ii. Use 3 comparisons to arrange elements in array such that a[1]<a[2], a[4]<a[5] and a[1]<a[4].
   a) Compare a[1] and a[2] and swap if necessary.
   b) Compare a[4] and a[5] and swap if necessary.
   c) Compare a[1] and a[4]. If a[4] is smaller then swap a[1] & a[4] and a[2] &a[5].

iii. If a[3]>a[2]. If a[2]<a[4], then median value is smaller of a[3] and a[4] otherwise median value is smaller of a[2] and a[5].

iv. If a[3]<a[2]. If a[3]>a[4] then median is smaller of a[3] and a[5] otherwise median is smaller of a[2] and a[4].

You can refer to the following figure which explains how this method works.



Q.43 Having n machines, each having n integers. Now you have to find medians of these n^2 integers but you can load only 2*n integers at a time in memory.

Q.44 Write a program to find if a number is multiple of 9.

Solution:- x/9= x/8 - (x/8 –x%8)/9

```
int nby9(int n)
{
        if(n == 0 || n == 9)
                return 1;
        if(n<9)
                return 0;
        return nby9(n>>3 – n&-7);
}
```