DSA

Preparation

Sorting Algorithm

With Time complexity

Just simplified my experience here…

Hope it goona help you all…

Save this pdf and thanks me later

@himanshu_shekhar16

@himanshushekar

Sorting-Algorithms

Sorting means to arrange a following set of numbers in ascending/increasing/non decreasing or descending/decreasing/non increasing order, and we need certain algorithms in programming to implement the same.

Various Sorting Algorithms are as follows:

Bubble Sort

Bubble sort, sometimes referred to as sinking sort, is a simple sorting algorithm that repeatedly steps through the list to be sorted, compares each pair of adjacent items and swaps them if they are in the wrong order. The pass through the list is repeated until no swaps are needed, which indicates that the list is sorted. The algorithm, which is a comparison sort, is named for the way smaller or larger elements "bubble" to the top of the list. Although the algorithm is simple, it is too slow and impractical for most problems even when compared to insertion sort. Bubble sort can be practical if the input is in mostly sorted order with some out-of-order elements nearly in position.

Time complexity analysis:

Worst Case	**Average Case**	**Best Case**
$O(n^2)$	$\Theta(n^2)$	$\Omega(n)$

In-place?	**Stable?**
Yes	Yes

Selection Sort

Selection sort is a sorting algorithm, specifically an in-place comparison sort. It has O(n2) time complexity, making it inefficient on large lists, and generally performs worse than the similar insertion sort. Selection sort is noted for its simplicity, and it has performance advantages over more complicated algorithms in certain situations, particularly where auxiliary memory is limited.

The algorithm divides the input list into two parts: the sublist of items already sorted, which is built up from left to right at the front (left) of the list, and the sublist of items remaining to be sorted that occupy the rest of the list. Initially, the sorted sublist is empty and the unsorted sublist is the entire input list. The algorithm proceeds by finding the smallest (or largest, depending on sorting order) element in the unsorted sublist, exchanging (swapping) it with the leftmost unsorted element (putting it in sorted order), and moving the sublist boundaries one element to the right.

Time complexity analysis:

Worst Case	Average Case	Best Case
$O(n^2)$	$\Theta(n^2)$	$\Omega(n^2)$

In-place?	Stable?
Yes	No

Insertion Sort

Insertion sort is a simple sorting algorithm that builds the final sorted array (or list) one item at a time. It is much less efficient on large lists than more advanced algorithms such as quicksort, heapsort, or merge sort. Insertion sort iterates, consuming one input element each repetition, and growing a sorted output list. At each iteration, insertion sort removes one element from the input data, finds the location it belongs within the sorted list, and inserts it there. It repeats until no input elements remain.

Time complexity analysis:

Worst Case	Average Case	Best Case
$O(n^2)$	$\Theta(n^2)$	$\Omega(n)$

In-place?	Stable?
Yes	Yes

Counting Sort

Counting sort is an algorithm for sorting a collection of objects according to keys that are small integers; that is, it is an integer sorting algorithm. It operates by counting the number of objects that have each distinct key value, and using arithmetic on those counts to determine the positions of each key value in the output sequence. Its running time is linear in the number of items and the difference between the maximum and minimum key values, so it is only suitable for direct use in situations where the variation in keys is not significantly greater than the number of items. However, it is often used as a subroutine in another sorting algorithm, radix sort, that can handle larger keys more efficiently.

Time complexity analysis:

Worst Case	Average Case	Best Case
O(n+k)	Θ(n+k>)	Ω(n+k)

In-place?	Stable?
No	Yes

Cycle Sort

Cycle sort is an in-place, unstable sorting algorithm, a comparison sort that is theoretically optimal in terms of the total number of writes to the original array, unlike any other in-place sorting algorithm. It is based on the idea that the permutation to be sorted can be factored into cycles, which can individually be rotated to give a sorted result.

Time complexity analysis:

Worst Case	Average Case	Best Case
$O(n^2)$	$\Theta(n^2)$	$\Omega(n^2)$

In-place?	Stable?
Yes	No

Heap Sort

Heapsort is a comparison-based sorting algorithm. Heapsort can be thought of as an improved selection sort: like that algorithm, it divides its input into a sorted and an unsorted region, and it iteratively shrinks the unsorted region by extracting the largest element and moving that to the sorted region. The improvement consists of the use of a heap data structure rather than a linear-time search to find the maximum.

The heapsort algorithm involves preparing the list by first turning it into a max heap. The algorithm then repeatedly swaps the first value of the list with the last value, decreasing the range of values considered in the heap operation by one, and sifting the new first value into its position in the heap. This repeats until the range of considered values is one value in length.

Time complexity analysis:

Worst Case	Average Case	Best Case
O(n log(n))	Θ(n log(n))	Ω(n) (bottom up)

In-place?	Stable?
Yes	No

Merge Sort

Merge sort is an efficient, general-purpose, comparison-based sorting algorithm. Most implementations produce a stable sort, which means that the implementation preserves the input order of equal elements in the sorted output. Merge sort is a divide and conquer algorithm. Conceptually, a merge sort works as follows:

1. Divide the unsorted list into n sublists, each containing 1 element (a list of 1 element is considered sorted).
2. Repeatedly merge sublists to produce new sorted sublists until there is only 1 sublist remaining. This will be the sorted list.

Time complexity analysis:

Worst Case	Average Case	Best Case
O(n log(n))	Θ(n log(n))	Ω(n log(n))

In-place?	Stable?
No	Yes

Quick Sort

Quicksort (sometimes called partition-exchange sort) is an efficient sorting algorithm, serving as a systematic method for placing the elements of an array in order. When implemented well, it can be about two or three times faster than its main competitors, merge sort and heapsort.

Quicksort is a comparison sort, meaning that it can sort items of any type for which a "less-than" relation (formally, a total order) is defined. In efficient implementations it is not a stable sort, meaning that the relative order of equal sort items is not preserved. Quicksort can operate in-place on an array, requiring small additional amounts of memory to perform the sorting. It is very similar to selection sort, except that it does not always choose worst-case partition.

Time complexity analysis:

Worst Case	Average Case	Best Case
$O(n^2)$	Θ(n log(n))	Ω(n log(n))

In-place?	Stable?
Yes	No

Radix Sort

Radix sort is a non-comparative integer sorting algorithm that sorts data with integer keys by grouping keys by the individual digits which share the same significant position and value. A positional notation is required, but because integers can represent strings of characters (e.g., names or dates) and specially formatted floating point numbers, radix sort is not limited to integers.

Time complexity analysis:

Worst Case	Average Case	Best Case
O(nk)	Θ(nk)	Ω(nk)

In-place?	Stable?
Depends on implementation	Depends on implementation

Cocktail Sort

Cocktail Sort is a variation of Bubble sort. The Bubble sort algorithm always traverses elements from left and moves the largest element to its correct position in first iteration and second largest in second iteration and so on. Cocktail Sort traverses through a given array in both directions alternatively.

Time complexity analysis:

Worst Case	Average Case	Best Case
$O(n^2)$	$\Theta(n^2)$	Ω(n)

In-place?	Stable?
Yes	Yes

Bogo Sort

Also known as Permutation Sort. Stupid Sort, Slowsort, Shotgun Sort, or Monkey Sort, Bogosort randomly generates permutations of the input and checks if it is sorted until it generates one that happens to be sorted by chance. Not meant to be an efficient sort, Bogosort is largely used for educational purposes and for comparison to actual sorting algorithms for contrast. It has a best case of O(n) where the list is already sorted, an average performance of O((n+1)!) and an unbounded worst case performance.

Time complexity analysis:

Worst Case	Average Case	Best Case
Unbounded	O((n+1)!)	Ω(n)

In-place?	Stable?
Yes	No

Bozo Sort

Bozosort is somewhat similar to Bogosort in its random behaviour. It works by choosing two random elements of the list and swapping them, until the list is actually sorted. The running time analysis of a bozosort is difficult, but some estimates are found in H. O(n!) is found to be the expected average case. Considering that every iteration calls a function that checks if the list is ordered or not, that makes the average case actually O((n+1)!)

Time complexity analysis:

Worst Case	Average Case	Best Case
Unbounded	O((n+1)!) (estimated)	Ω(n)

In-place?	Stable?
Yes	No

Shell Sort

Shellsort, also known as Shell sort or Shell's method, is an in-place comparison sort. It can be seen as either a generalization of sorting by exchange (bubble sort) or sorting by insertion (insertion sort).The method starts by sorting pairs of elements far apart from each other, then progressively reducing the gap between elements to be compared. Starting with far apart elements, it can move some out-of-place elements into position faster than a simple nearest neighbor exchange. Donald

Shell published the first version of this sort in 1959.The running time of Shellsort is heavily dependent on the gap sequence it uses. For many practical variants, determining their time complexity remains an open problem.

Time complexity analysis:

Worst Case	Average Case	Best Case
$O(n^2)$	Depends on Implementation	$\Omega(n \log(n))$

In-place?	Stable?
Yes	No

IntroSort

Introsort or introspective sort is a hybrid sorting algorithm that provides both fast average performance and (asymptotically) optimal worst-case performance. It begins with quicksort and switches to heapsort when the recursion depth exceeds a level based on (the logarithm of) the number of elements being sorted. This combines the good parts of both algorithms, with practical performance comparable to quicksort on typical data sets and worst-case O(n log n) runtime due to the heap sort. Since both algorithms it uses are comparison sorts, it too is a comparison sort.

Time complexity analysis:

Worst Case	Average Case	Best Case
O(n log(n))	Θ(n log(n))	O(n log(n))

In-place?	Stable?
Yes	No

Bucket Sort

Bucket sort is a sorting algorithm that involves first putting elements that need to be sorted into "buckets", sorting these buckets and then merging the results to get the sorted list. It is best used on data that is uniformly distributed over a range and can perform quite well, O(n + m) (where n is the number of elements in the list and m is the number of buckets used). The buckets are typically stored as an array of linked lists so there is an additional space requirement for this sorting algorithm. For example, if there is a list of numbers in the range 1 to 1000 we can have 10 buckets. All numbers between 1 and 100 get added to the first bucket, between 101 and 200 get added to the second bucket, etc. After all numbers have been added to the appropriate bucket insertion sort is applied to all the buckets. This step is the bottleneck for the algorithm. If the data is not

clustered tightly than the sorts can be completed in linear time despite the fact that insertion sort is $O(n^2)$. The algorithm degrades as certain buckets get more of the numbers, the worst case being when all the numbers are in a single bucket. The data in the input array must uniformly distributed across the range of bucket values to avoid the polynomial time. The size of each bucket should be equal to the number of buckets. As bucket sort is polynomial in the worse case Quicksort is a more optimal sorting algorithm.

Time complexity analysis:

Worst Case	Average Case	Best Case
$O(n^2)$	$\Theta(n+k)$	$\Omega(n+k)$

In-place?	Stable?
Yes	Yes

Gnome sort

Gnome sort is a sorting algorithm which is similar to insertion sort, except that moving an element to its proper place is accomplished by a series of swaps, similar to a bubble sort. It is conceptually simple, requiring no nested loops. The average, or expected, running time is O(n2) but tends towards O(n) if the list is initially almost sorted.

The algorithm always finds the first place where two adjacent elements are in the wrong order and swaps them. It takes advantage of the fact that performing a swap can introduce a new out-of-order adjacent pair next to the previously swapped elements. It does not assume that elements forward of the current position are sorted, so it only needs to check the position directly previous to the swapped elements.

Time complexity analysis:

Worst Case	Average Case	Best Case
$O(n^2)$	$O(n^2)$	$\Omega(n^2)$

In-place?	Stable?
Yes	Yes

Comb sort

Comb sort improves on bubble sort. The basic idea is to eliminate turtles, or small values near the end of the list, since in a bubble sort these slow the sorting down tremendously. Rabbits, large values around the beginning of the list, do not pose a problem in bubble sort.

In bubble sort, when any two elements are compared, they always have a gap (distance from each other) of 1. The basic idea of comb sort is that the gap can be much more than 1. The inner loop of bubble sort, which does the actual swap, is modified such that gap between swapped elements goes down (for each iteration of outer loop) in steps of a "shrink factor" k: [n/k, n/k^2, n/k^3, ..., 1].

The gap starts out as the length of the list n being sorted divided by the shrink factor k (generally 1.3; see below) and one pass of the aforementioned modified bubble sort is applied with that gap. Then the gap is divided by the shrink factor again, the list is sorted with this new gap, and the process repeats until the gap is 1. At this point, comb sort continues using a gap of 1 until the list is fully sorted. The final stage of the sort is thus equivalent to a bubble sort, but by this time most turtles have been dealt with, so a bubble sort will be efficient.

The shrink factor has a great effect on the efficiency of comb sort. k = 1.3 has been suggested as an ideal shrink factor by the authors of the original article after empirical testing on over 200,000 random lists. A value too small slows the algorithm down by making unnecessarily many comparisons, whereas a value too large fails to effectively deal with turtles, making it require many passes with 1 gap size.

The pattern of repeated sorting passes with decreasing gaps is similar to Shellsort, but in Shellsort the array is sorted completely each pass before going on to the next-smallest gap. Comb sort's passes do not completely sort the elements. This is the reason that Shellsort gap sequences have a larger optimal shrink factor of about 2.2.

Time complexity analysis:

Worst Case	**Average Case**	**Best Case**
$O(n^2)$	$\Theta(n^2/2^p)$,where p is the number of increments	$\Omega(n \log(n))$

In-place?	**Stable?**
Yes	No

Tim sort

Timsort is a hybrid stable sorting algorithm, derived from merge sort and insertion sort, designed to perform well on many kinds of real-world data. It uses techniques from Peter McIlroy's "Optimistic Sorting and Information Theoretic Complexity", in Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 467–474, January 1993. It was implemented by Tim Peters in 2002 for use in the Python programming language. The algorithm finds subsequences of the data that are already ordered, and uses that knowledge to sort the remainder more efficiently. This is done by merging an identified subsequence, called a run, with existing runs until certain criteria are fulfilled. Timsort has been Python's standard sorting algorithm since version 2.3. It is also used to sort arrays of non-primitive type in Java SE 7, on the Android platform, and in GNU Octave.

Time complexity analysis:

Worst Case	Average Case	Best Case
O(n log(n))	O(n log(n)	O(n)

In-place?	Stable?
No	Yes

Tournament sort

Tournament sort improves upon the naive selection sort by using a priority queue to find the next element in the sort. In the naive selection sort, it takes O(n) operations to select the next element of n elements; in a tournament sort, it takes O(log n) operations (after building the initial tournament in O(n)). Tournament sort is a variation of heapsort.

Time complexity analysis:

Worst Case	Average Case	Best Case
O(n log(n))	O(n log(n)	O(n log(n))

In-place?	Stable?
No	No

IN SHORT:

Red and black is used for searching whereas for sorting it is better to use Heapsort(for local projects).The time complexity is for the worst case is "2nlogn" and best case is "n". But it is not stable, if you prefer the stable sort it is better to use the merge sort. O(n) = n log n. If the sorting is for priority of top 10 or least 10, then use priority Queue. (Max-PQ and Min-PQ). Source : (use this links for the code, it is an industrial code and the code is written in generics(can use any datatypes))
Heap: https://algs4.cs.princeton.edu/code/edu/princeton/cs/algs4/Heap.java.html Merge: https://algs4.cs.princeton.edu/code/edu/princeton/cs/algs4/Merge.java.html For other sorting and searching algorithms: https://algs4.cs.princeton.edu/code/

Bitonic Sort

The Bitonic Sort is a parallel comparison-based sorting algorithm which does O(nlogn) comparisons. It is also called as the Bitonic Merge Sort. The Bitonic Sort is based on the concept of converting the given sequence into a Bitonic Sequence. A Bitonic Sequence is a sequence of numbers which is first strictly increasing then after a point strictly decreasing. Although, the number of comparisons are more than that in any other popular sorting algorithm, It performs

better for the parallel implementation because elements are compared in predefined sequence which must not be depended upon the data being sorted. The predefined sequence is called Bitonic sequence. Therefore it is suitable for implementation in hardware and parallel processor array.

Time complexity analysis:

Worst Case	Average Case	Best Case
$O(\log^2 n)$	$O(\log^2 n)$	$O(\log^2 n)$

In-place?	Stable?
Yes	No

Pancake Sort

Pancake sorting is the colloquial term for the mathematical problem of sorting a disordered stack of pancakes in order of size when a spatula can be inserted at any point in the stack and used to flip all pancakes above it. A pancake number is the minimum number of flips required for a given number of pancakes. Unlike a traditional sorting algorithm, which attempts to sort with the fewest comparisons possible, the goal is to sort the sequence in as few reversals as possible.

Time complexity analysis:

Worst Case	Average Case	Best Case
O(n^2)	O(n)	O(n)

In-place?	Stable?
No	No

Feel Free to connect with me –

https://www.linkedin.com/in/himanshushekhar16/