QuickSort:

Advantages

- It is in-place since it uses only a small auxiliary stack.
- It requires only ***n (log n)*** time to sort **n** items.
- It has an extremely short inner loop.
- This algorithm has been subjected to a thorough mathematical analysis, a very precise statement can be made about performance issues.

Disadvantages

- It is recursive. Especially, if recursion is not available, the implementation is extremely complicated.
- It requires quadratic (i.e., n2) time in the worst-case.
- It is fragile, i.e. a simple mistake in the implementation can go unnoticed and cause it to perform badly.

Properties:

- The quick sort is an **in-place, divide-and-conquer, massively recursive** sort algorithm.
- The efficiency of the algorithm is majorly impacted by which element is chosen as the pivot point.
- The worst-case efficiency of the quick sort is o(n^2) when the list is sorted and left most element is chosen as the pivot.
- As long as the pivot point is chosen randomly, the quick sort has an algorithmic complexity of O(n log(n)).
- If the **partitioning** is **balanced**, the algorithm runs asymptotically as fast as merge
- If the **partitioning** is **unbalanced**, however, it can run asymptotically as slowly as insertion sort

Worst-case partitioning:

The worst-case behavior for quick sort occurs when the partitioning routine produces one sub-problem with (n-1) elements and one with 1 element.

- Assume that this unbalanced partitioning arises in each recursive call. which evaluates to 0(n^2)

Best-case partitioning:

PARTITION produces two sub-problems, each of size no more than n/2, since one is of size $\lfloor n/2 \rfloor$ and one of size $\lfloor n/2 \rfloor$ -1.

O(n lg n)

Average-case Partitioning / Balanced Partitioning:

The running time is **O(n log n)** whenever the split has constant proportionality in each (sub)partition like 1:9

Pros (Quick sort):

- Quick sort is in-place sorting algorithm.In-place sorting means, it does not use additional storage space to perform sorting.

Cons:

- Not stable

Insertion Sort

Properties:

- **INSERTION-SORT** can take different amounts of time to sort two input sequences of the same size depending on how nearly sorted they already are.
- In INSERTION-SORT, the **best case** occurs if the array is already sorted.

T [Best Case]= O(n)

- If the array is in reverse sorted order i.e in decreasing order, INSERTION-SORT gives the **worst case** results.

T [Worst Case]= $\theta(n^2)$

- **Average Case**: When half the elements are sorted while half not
- The running time of insertion sort therefore belongs to both $\Omega(n)$ and $O(n^2)$

Pros:

- For nearly-sorted data, it's incredibly efficient (very near O(n) complexity)
- It works in-place, which means no auxiliary storage is necessary i.e. requires only a constant amount O(1) of additional memory space
- Efficient for (quite) small data sets.
- Stable, i.e. does not change the relative order of elements with equal keys

Cons:

- It is less efficient on list containing more number of elements
- Insertion sort needs a large number of element shifts

Selection Sort

The main advantage of the selection sort is that it performs well on a small list. Furthermore, because it is an in-place sorting algorithm, no additional temporary storage is required beyond what is needed to hold the original list. The primary disadvantage of the selection sort is its poor efficiency when dealing with a huge list of items. Similar to the bubble sort, the selection sort requires n-squared number of steps for sorting n elements. Additionally, its performance is easily influenced by the initial ordering of the items before the sorting process. Because of this, the selection sort is only suitable for a list of few elements that are in random order.

Merge Sort:

Properties

- Merge Sort's running time is **Ω(n log n)** in the **best-case, O(n log n)** in the **worst-case,** and **Θ(n log n)** in the **average-case** (when all permutations are equally likely).
- The **space complexity** of Merge sort is **O(n)**. This means that this algorithm takes a lot of space and may slower down operations for the last data sets.

Pros:

- It is **quicker for larger lists** because unlike insertion it doesn't go through the whole list several times.
- The merge sort is **slightly faster than the heap sort** for larger sets
- $O(nlogn)$ worst case asymptotic complexity.
- Stable sorting algorithm

Cons

- **Slower** comparative to the other sort algorithms **for smaller data sets**
- Marginally slower than quick sort in practice
- Goes through the whole process even if the list is sorted
- It uses more memory space to store the sub elements of the initial split list.
- It requires twice the memory of the heap sort because of the second array.

Heap Sort

Properties:

- Heap sort involves **building a Heap data structure** from the given array and then **utilizing the Heap to sort the array**
- Heap data structure is always a Complete Binary Tree, which means all levels of the tree are fully filled
- A.heap_size of an array is initially the size of the array. At first iteration, after exchanging root of the max_heap tree (A[1]) with A[i] = A[A.length] (last element inside array A)
- Doing extract_max(), A.heap_size value will be decreased by 1
- max_heap structure should be max_heapified: A[Parent(i)] >= A[i], where Parent(i) returns i/2 of heap tree.
- Initially create a Heap. extract_max(), put element of the heap in the array until we have the complete sorted list in our array.
- The Heap Sort sorting algorithm seems to have a worst case complexity of O(n log(n))

Pros:

- Heapsort and merge sort are asymptotically optimal comparison sorts

Cons: N/A

Heap Sort vs Merge Sort:

- The time required to merge in a merge sort is counterbalanced by the time required to build the heap in heapsort

- **Heap Sort is better :**The Heap Sort sorting algorithm uses O(1) space for the sorting operation while Merge Sort which takes O(n) space

- **Merge Sort is better**
  * The merge sort is slightly faster than the heap sort for larger sets
  * Heapsort is not stable because operations on the heap can change the relative order of equal items.

Heap Sort vs Insertion Sort:

Similarity

- Heap sort and insertion sort are both used comparison based sorting technique

Differences

- Heap Sort is not stable whereas Insertion Sort is.

- When already sorted, Insertion Sort will not sort every element again where as Heap Sort will use extract max and heapify again and again
  When already sorted, Insertion Sort takes O(n) TC whereas Heap Sort will take O(n log(n)) time
  Insertion Sort is not efficient for large input data whereas Heap Sort is.

Heap Sort vs Quick Sort:

- Heapsort is O(n log n) guaranted, what is much better than worst case in Quicksort

- Heapsort is an excellent algorithm, but a good implementation of quicksort, selection problems, usually beats it in practice i.e O(n)

- QS runs fast, much faster than Heap and Merge algorithms.
- Quicksort almost doesn’t do unnecessary element swaps. Swap is time consuming
- With Heapsort, even if all of your data is already ordered, you are going to swap 100% of elements to order the array

Quick Sort vs Merge Sort:

Similarity:

- Both are sorting techniques
- Both are built on the **divide and conquer** method in which the set of elements are parted and then combined after rearrangement

Differences:

- In quick sort the pivot element is used for the sorting while merge sort does not use pivot element for performing the sorting.
- In quick sort, the splitting of a list of elements is not necessarily divided into half unlike Merge Sort where the array is always divided into half (n/2)
- Worst case complexity for Merge sort is ***$O(n \log n)$*** whereas for Quick Sort, it is ***$O(n^2)$***
- Additional storage space requirement is less in Quick Sort than Merge Sort
- Quick sort is in-place sorting algorithm while Merge Sort requires additional storage space to perform sorting
- The quick sort usually requires more comparisons than merge sort for sorting a large set of elements
- Merge sort is stable while quick sort is not.
- Merge Sort is more more efficient for larger arrays

Insertion sort vs Merge Sort

Similarity

- Both are comparison based sorting algorithms

Difference:

- To work on an almost sorted array, Insertion sort takes linear time i.e. O(n) while Merge and Quick sort takes O(n*logn) complexity to sort

Binary Search Tree

What are advantages & disadvantages of binary search tree?

Advantages of BST are:

- we can always keep the cost of insert(), delete(), lookup() to O(logN) where N is the number of nodes in the tree - so the benefit really is that lookups can be done in logarithmic time which matters a lot when N is large.
- We have an ordering of keys stored in the tree. ...
- We can impleme.

What are disadvantages of binary search tree?

Disadvantages: **It's more complicated than linear search, and is overkill for very small numbers of elements**. It works only on lists that are sorted and kept sorted....

- Time complexity : O(logn)
- Better search algorithm than linear search.
- Disadvantage: Array should be sorted.