2019BTECS00058
Devang K
Batch T7

Additional Task for week 1 & 2
Study of all sorting algorithms.

Q1) Classification of Sorting Algorithms

→ Sorting algorithms are categorized on the following basis :-

1) By number of comparisons.
→ Comparison-based sorting algorithms check the elements of the list by key comparison operation & need atleast O(n log n) comparisons for most inputs.
→ Best case behaviour is O(n log n) and worst-case is O(n²)
Eg: Counting sort, bucket sort, radix sort, etc.

2) By Number of Swaps.
→ Sorting algorithms are categorized by the no. of swaps.

3) By memory Usage.
→ Some sorting algorithms are 'in place' and they need O(1) or O(log N) memory to create auxillary locations for sorting data temporarily.

4) By recursion.
→ Sorting algorithms are either recursive (quicksort) or non-recursive (selection sort) and these are some algorithms that use both.

5) By stability.
→ Sorting algorithm is stable if 2 elements with equal

values appear in the same order in o/p as it was in i/p. The stability of a sorting algorithm can be checked with how it treats equal elements.
→ Stable algorithms preserve the relative order of equal elements, while unstable algorithms - don't.

6) By adaptability.
→ In a few sorting algorithms, the complexity changes based on pre-sorted input, i.e. pre-sorted array of the input affects the running time. The algorithms that take this adaptability into account are called adaptive algorithms.
Eg → Bubble, Insertion, Quick → Adaptive.

7) Internal Sorting
→ Sorting algorithms that use main memory exclusively during the sort are called internal sorting algorithms.
Eg → Bubble, Insertion, Quick

8) External Sorting
→ Sorting algorithms that use external memory, during the sorting, come under this category.
→ They are comparitively slower than internal sorting algorithms.
Eg → Merge sort algorithm.

Q2) Performance of Sorting Algorithms

1) Worst Case Analysis (Usually Done)

→ In the worst case analysis, we calculate the upper bound running of an algorithm. we must know the case that causes maximum number of operations to be executed.

→ For linear search, the worst case happens when the element to be searched is not present in the array. when x is not present, the search function compares it with all elements of arr[] one by one.

∴ Worst case complexity of linear search would be $\Theta(n)$

2) Average Case Analysis.

→ In average case analysis, we take all possible inputs and calculate computing time for all of the inputs. Sum all the calculated values & divide the sum by total number of inputs. we must know the distribution of cases.

3) Best case Analysis.

→ In the best-case analysis, we calculate lower bound on running time of an algorithm, we must know the case that causes min. no. of operations to be executed.

→ In the linear search problem, the best case occurs when x is present at the first locator. The no. of operations in the best case is constant. So, the time complexity in the best case would be $\Theta(1)$

→ Mostly, we use worst case analysis - practical. Average case is not easy to determine and best case is bogus.