**Sorting Techniques (Important points)**

**Comparison based sorts :-**

1. Bubble sort - n^2 – can be made adaptive and is stable- k passes funda (we get k largest elements after k passes)

(comparing two consecutive elements at a time in n-1 passes)

1. Insertion sort - n^2- (adaptive and stable)- best suited for linked list (no k passes funda)

(Considering first element to be already sorted and then using insertion technique from the end of already sorted list)

1. Selection sort – Only sort which gives minimum number of passes – n passes –

k- passes funda ( we get k smallest elements after k passes) – neither adaptive nor stable.

(Moving j & k initializing at i and then find the minimum element from whole pass for ith position and then swapping)

1. Quick Sort – best case and avg case – nlogn

Worst case – n^2 (when partion is done on either of the ends, eg- when the list is already sorted)

* Neither adaptive nor stable
* Here we take a pivot element and then interchange the larger elements than pivot from left with smaller elements than pivot on right (n) and then partition happens in (logn) time i.e. (total =nlogn)

1. Merge Sort – (nlogn) uses merging + sorting – It is the only comparison based sort which uses extra space for merging

As merging requires already sorted array, so in merge sort , we consider the full unsorted array as an array of n sorted listst and we perform logn passes in which we compare n elements. (nlogn – complexity)

Merging – time complexity = (m+n)

* not adaptive but stable !!

1. Count Sort – (n) ---(actually, m+n) ; m=size of count array having size = 1+max element of array to be sorted ; n=size of original array.

- index based sort, consumes a lot of extra space, fastest sort

1. Radix Sort- (n) – it takes more time than count sort because count sort has only one pass whereas radix sort has no. of passes = largest digit numbers. And we do them from last digit onwards. But advantage of it is that it consumes very less space extra array equal to the number system used and total no. of nodes in those bins = size of the original array means, it also need an extra array of linked lists i.e. (n) . Eg- for decimal numbers, 10 bins will be used (0 to 9) with n nodes.

**Concept : last digit of a number = (num/1)%10;**

**Ten’s place digit = (num/10)%10;**

**100th place digit = (num/100)%10; and so on…**

(The stopping criteria would be if the divisor say 10000 is greater than all the numbers in the original array -> STOP !!

1. Bucket/Bin Sort – similar to count sort except duplicate numbers are stored in continuous nodes i.e. array of linked list is extra needed. So total space consumed = n+m ; m= largest element of the original array and n= no.of nodes created which are equal to total number of elements in the original array. Complexity= n+m = (n)
2. Shell Sort – It’s an extension of Insertion Sort. It is useful for a very large sized list of elements. It has concept of GAPS = floor(n/2); and continue till gap>0 . When gap==1 ; we are doing insertion sort. In Shell Sort, because we have already done shiftings in the earlier gaps, so there are very less number of shifting required in the last gap as compared to the usual Insertion Sort. Time Comlexity- if gap divided by 2 ( nlogn ) or if gap is prime number – n^3/2 or n^5/3

**NOTE : during a gap, if the two elements are swapped after comparison and if the previos gap >= GAP, then you have to compare those two also and continue this same procedure if those two also get swapped.**

It’s a comparison based sort. As Insertion Sort is adaptive, so shell sort is also. It’s an inplace sorting because it doesn’t need any extra space. Among all comparison based sorts, only merge sort needs extra space , otherwise all the sorts are inplace sorts !!