

ALGO_DS_DAY-09:

Quick Sort:

Partitioning:

step-1: select left most element as a pivot element

step-2: shift ele's which are smaller than pivot towards left (as possible), and shift ele's which are greater than pivot towards right (as possible).

In first pass, pivot ele gets settled/fixed at its appropriate position and big size array gets divided logically into two partitions => left partition & right partition

left partition => left to j-1

right partition => j+1 to right

for left partition => value of left remains same, right = j-1

for right partition => value of right remains same, left = j+1

- we can apply partitioning on left partition as well as right partition recursively till the size of partition is >1.

partition is valid till left < right

if(left >= right) => partition is invalid

i=left;

j=right;

pivot = arr[left];

while(i < j){

 while(i <= right && arr[i] <= pivot)
 i++;

 if(i < j)//if i & j have not crossed swap them
 {
 swap(arr[i], arr[j]);
 }

}

//swap pivot ele with jth element

swap(arr[j], arr[left]);

=> for partitioning quick sort algo takes log n, and no. Passes is depends on size of an array, as size of an increases no. of passes i.e. no. Of times partitioning required also increases and in avg as well as best case time required for this algo is $n \cdot \log n$
 $T(n) = O(n \log n)$

=> worst case occurs in quick sort if either array is already sorted or array ele's are exactly in a reverse order, which rarely occurs.

[10 20 30 40 50 60]

pass-1: partitioning => [10 20 30 40 50 60] => pivot = 10

[LP] 10 [20 30 40 50 60]

pass-2: partitioning => [20 30 40 50 60] => pivot = 20
[LP] 20 [30 40 50 60]

pass-3: partitioning => [30 40 50 60] => pivot = 30
[LP] 30 [40 50 60]

pass-4: partitioning => [40 50 60] => pivot = 40
[LP] 40 [50 60]

pass-5: partitioning => [50 60] => pivot = 50
[LP] 50 [60]

- in quick sort, under worst case partitioning takes $O(n)$, array is not gets divided equally into two partitions, and no. of passes = $n-1$

total no. of comparisons = $n * (n-1)$

$T(n) = O(n * (n-1))$

$T(n) = O(n^2 - n)$

$T(n) = O(n^2)$

Merge Sort:

works on two principles:

1. as the size of an array is min sorting is efficient.
2. it is always efficient to merge two already sorted arrays into a single array in a sorted manner.

algorithm:

- step-1: divide big size array logically into smallest size (i.e. having size 1) subarray's.

- we can divide array into subarray's logically by means of calculating mid pos

$mid = (left + right) / 2$,

by means of calculating mid pos, big size array gets divided logically into two subarray's

left subarray & right subarray

left subarray => left to mid

right subarray => mid+1 to right

for left subarray => value of left remains same, right = mid

for right subarray => value of right remains same, left = mid+1

for divisioning it takes $O(\log n)$ time, as size of an array increases time required for dividing also gets increases

$T(n) = O(n * \log n)$

- merge two already sorted lists into a single list in a sorted manner

list1 => head => null

list2 => head => null

```
list3 => head => 5 -> 10 -> 15 -> 20 -> 25 -> 30 -> 35 -> 40 -> 45  
-> 50 -> 60 -> null
```

+ "Graph" : it is a non-linear/advanced data structure, which is a collection of logically related similar and dissimilar type of data elements, which contains

- finite set of elements called as vertices, also referred as a "nodes", and finite set of ordered/unordered pairs of vertices called as an "edges", also referred as an "arcs", which may contains weight/cost/value and it may be -ve.

example:

google map

to store info about cities and info about paths between cities graph data structure is used.

- to store info about city => vertices => City class object
- to store info about paths between cities => edges => Path class objects

```
class City{  
    String cityCode;  
    String cityName;  
    String stateName;  
    String countryName;  
    .....  
    .....  
    .....  
}
```

```
class Path{  
    String pathCode;  
    String pathName;  
    String srcCityName;  
    String destName;  
    float distInKm;  
    .....  
    .....  
}
```

- graph is a collection of 100's of city class objects and 1000's of path class objects between those cities.

- we can apply algo's like to find shortest distance of all vertices from given source vertex like dijkstra's algo, floyd-warshall algo or to find MST like prim's & kruskal's.

- if we want to store information in a network we can go for graph data structure.

SunBeam