Mid

Roll: 2019014032 , Nazia Shehnaz Joynab
Course Code : 203

Ans. to the Ques. No : 01 :

Worst case of quick sort algorithm is $O(n^2)$ when the picked pivot element is the smallest or largest element. The worst case ~~e~~ time complexity of quick sort can be improved ~~Reduced~~ if the pivot element is the median element of the input array. The time complexity would be $O(n \log n)$ is worst case.

Example : Let us take an array of ~~5 si~~ size 5

A

50	30	20

~~0 2~~

A

50	40	30	20	10
0	1	2	3	4

if we take last element as pivot

~~pp~~ ie. pivot = ~~10~~ 20

pe pseudo code for quicksort algo if pivot is the last element of the array :

Partition (int arr[],

the partition process :

A[

```
Partition ( A, st, end)
{
    if ( i =
int pivot = A[end];
int i = st - 1; //index of f
for ( int j = st ; j <= end - 1; j++)
{
    if (A[j] <= pivot)
    {
        i++;
        swap A[i] and A[j];
        i++
    }
}
```

swap A[i+1] and A[end];

return (i+1).

}

// quicksort algo

quicksort (arr[], st, end).

{ if (st < end)

{ pi = partition (A, st, end);

quicksort (A, st, pi - 1);

quicksort (A, pi + 1, end);

}

}

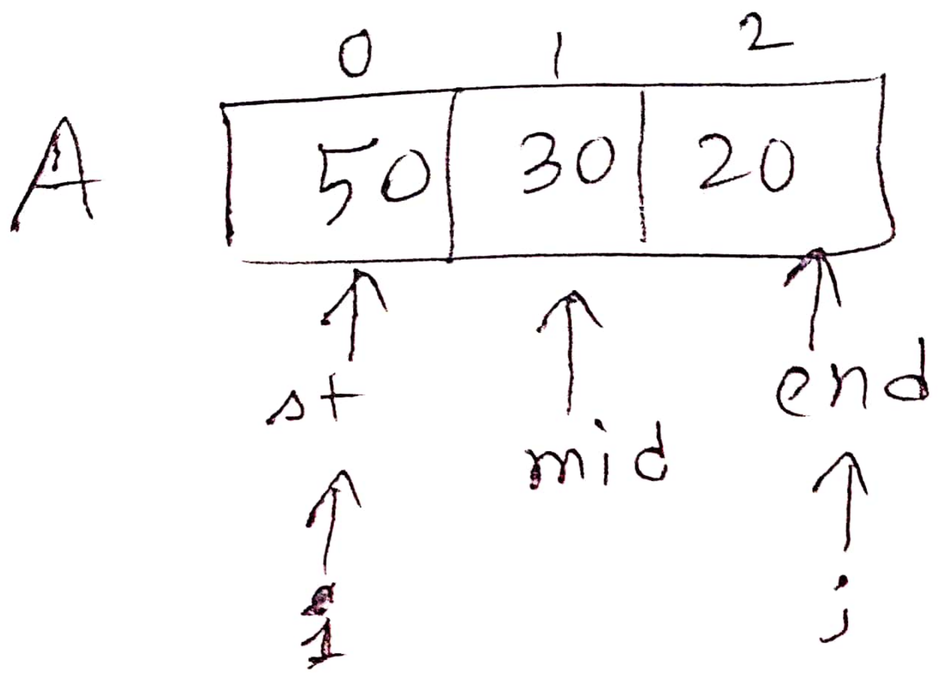
pseudocode for quicksort if pivot is the median element of the array :

```
void quicksort (A, st, end, n)
                //n=size of array
{
    int i, j, pivot;
    if (st>=end)
        return;
    i = st;
    j = end;
    int mid = (st + end) /2;
    pivot = A [mid];
    while (i<=j)
    {
        while(a[i]<pivot)
        {
            i++;
        }
    }
```

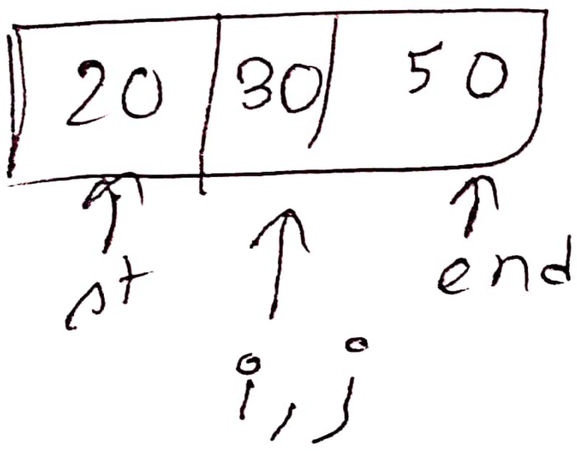
```
while (a[j] > pivot){
        j--;}
    if (i<=j)
{   swap(a[i], a[j]);
        i++;
        j--;
    }
} ← end of first while loop

    //recursion
    quicksort(A, st, j, n);
    quicksort(A, i, end, n);
} ← end of function
```

Simulation of quicksort using median as pivot element:



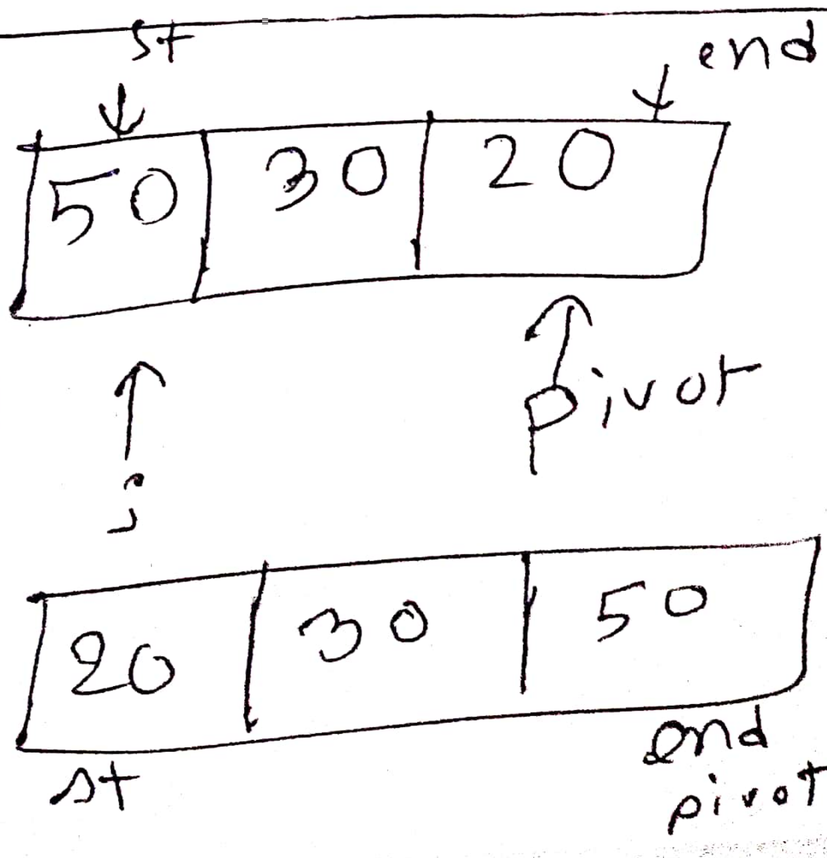
pivot = 30



// ~~qt~~ call quicksort for (A, st, i, n)

// call quicksort for (A, i, end, n).

Taking pivot as end element



← swapping A[i+1], A[end]

Ans. to the Ques. No: 02:

Given a graph G
where n vertices,
m edges,
are present.
and degree of vertex v is
[deg(v)].

property 1:

For an undirected graph

$$\sum deg(v) = 2m$$

if there are v vertices in a graph, then ~~all~~ + summation of all degrees of all vertices is equal to the twice of m.

Proof: Each edge is counted twice.

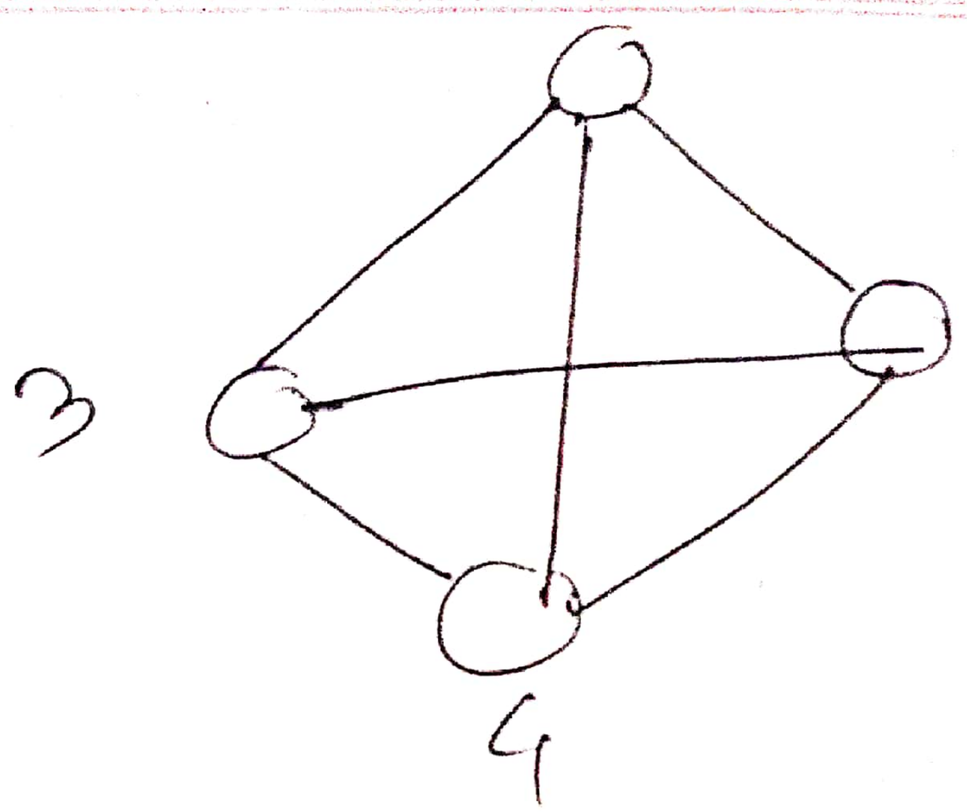


Fig 1

$deg(1) = 2$
$deg(2) = 2$
$deg(3) = 2$
$deg(4) = 2$

total $= 4 \times 2 = 8$ edges

$= 2 \times m$

$= 2 \times 4$

Property 2:

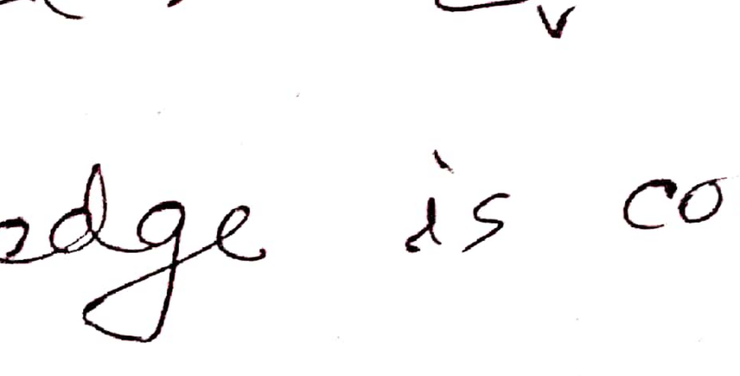
For a directed graph

$$\sum_v indeg(v) = \sum_v outdeg(v) = m$$

Proof: each edge is counted once for in-degree and once for out degree.

Ex: ~~indegree of de 1 in fig 1~~

$\neq$



Property 3 :

If G is a simple **undirected** graph, ~~them~~ then

$$m \leq n(n-1)/2$$

and

If G is a simple **directed** graph, then

$$m \leq n(n-1)$$

Proof : each vertex has degree at most $(n-1)$. then we can use property 1 & 2, to proove that.

Property 4:

Let G be an undirected graph with n vertices and m edges.

→ If G is connected then $\boxed{m \geq n-1}$

→ If G is a tree $\boxed{m = n-1}$

→ If G is a forest $\boxed{m \leq n-1}$

Proof: Let G be a connected graph. Delete edges one by one from G keeping G connected. At last there will remain $(n-1)$ edges.

Ans. to the Ques. No:03:

differences between adjacency matrix and adjacency list. ~~here suppose there~~

adjacency list	adjacency matrix
① It is an one dimentional array, Vector/linked list	① It is a 2D array. ~~here~~
② takes less memory space than adjacency matrix. space complexity is O(V+E)	② takes more space. storage space is required in worst case is O(|V|²)
③ To add an edge it ~~is~~ requires O(1) time	③ To add an edge it requires O(1) time.
④ To remove an edge, we have traverse through all the edges ~~f~~ in worst case. So, the time complexity is O(|E|)	④ To remove an edge say from i to j ~~ma~~ we need to make matrix[i][j] =0, which needs O(1) time

⑤ time complexity for searching an edge in adj. list in $O(\|V\|)$.	⑤ time complexity for searching an element in matrix is $O(1)$. if we know the ~~ma~~ two vertices i and j, we can find matrix[i][j]
⑥ Adding a vertex to adj ~~matrix~~ list takes $O(1)$ time	⑥ Adding a new vertex to adjacenty matrix means creating a new array of V+1 size. To do that we need to copy the whole matrix. So time complexity for this is $O(\|V\|^2)$

CT 3 | Roll : 2019140032 | Course Code : 203 | Nazia Shehnaz Joynab

Ans. to the Ques. No : 04 :

betn

diff^n bubble sort, insertion sort, selection sort, merge sort and quick sort:

~~void~~

(1) time complexity of bubble sort (if not optimized), is $O(n^2)$ in best case, worst case and average case. but if the of algo is optimized, then the worst case complexity is improved by $O(n)$. Bubble sort can be used for small ~~set of~~ size of arrays

(2) time complexity for ~~insertion~~ ~~sort~~ selection sort ~~in~~ is $O(n^2)$ in worst case, best case and average case.

time complexity of

③ Insertion sort is $O(n^2)$ is worst case. and in best case, ~~time~~ ~~con~~ it is $\Theta(n)$

④ Time complexity of merge sort is $O(n \log n)$ is best, worst and avercgge case.

⑤ Time complexity of quick sort is $O(n \log n)$ is best case. And in worst case, its the pivot is the end element, then the time complexity is $o(n^2)$. and if the pivot is median element, time complexity is $O(n \log n)$

for large data sets, mergesort is used.

Ans: to the Ques. No: 05:

A[3:

A	38	27	43	3	9	82	10

mergesort in A: n = size = 7