Name - TANISHQ
Section - CST
Roll No - 34

Tanishq

Date / /

Tutorial 3

Q1. Write a linear search pseudo code to search an element in a sorted array with minimum ~~tord~~ comparisons.

Sol 1.
```
int linear_search(int A[], int n, int t)
{   if (abs(A[0]-t) > abs(A[n-1]-t))
        for(i=n-1 to 0; i--)
            if(A[i]==t)
                { return i; }
    else
        for(i=0 to n-1; i++)
            if(A[i]==t)
                return i;
}
```

Q2. Iterative Insertion Sort

```
void insertion(int A[], int n)
{
    for(i=1 to n)
    {   t = A[i];
        j = i;
        while(j>=0 && t<A[j])
        {
            A[j+1] = A[j];
            j--;
        }
        A[j+1] = t;
    }
}
```

Tanishq

Date / /

Recursive Insertion Sort

```
void insertion (int A[], int n)
{   if (n <=1)
      return;
    insertion (A, n-1);
    int last = A[n-1];
    int j = n-2;
    while (j>=0 && A[j] > last)
    {   A[j+1] = A[j];
        j--;
    }
    A[j+1] = last;
}
```

Insertion sort is also called online sorting algorithm because it will work if the elements to be sorted are provided one at a time with the understanding that the algorithm must keep the sequence sorted as more elements are added in.
Other sorting algorithms like bubble sort, insertion sort, heap sort etc are considered external sorting technique as they need the data to be sorted in advance.

Tanishq

Date / /

3. Complexity of all sorting algorithms that have been discussed in lectures-

	Best Case	Worst Case
Bubble Sort	$O(n^2)$	$O(n^2)$
Selection Sort	$O(n^2)$	$O(n^2)$
Insertion Sort	$O(n)$	$O(n^2)$
Count Sort	$O(n)$	$O(n+k)$
Quick Sort	$O(n\log n)$	$O(n^2)$
Merge Sort	$O(n\log n)$	$O(n\log n)$
Heap Sort	$O(n\log n)$	$O(n\log n)$

4.

	Inplace	Stable	Online
Bubble	✓	✓	✗
Selection	✓	✗	✗
Insertion	✓	✓	✓
Count	✗	✓	✗
Quick	✓	✗	✗
Merge	✗	✓	✗
Heap	✓	✗	✗

5. Recursive / Iterative pseudo code for binary search

Iterative

```
int binarySearch(int arr[], int x)
{
    int l=0, r= arr.length-1;
    while (l<=r)
    {
        int m = l+(r-l)/2;
        if (arr[m]==x)
            return m;
```

Tanishq

Date / /

```
    if (arr[m] < x)
        l = m + 1;
    else
        r = m - 1;
}
return -1;
}
```

Recursive

```
int binarySearch (int arr[], int l, int r, int x)
{
    if (r >= l)
    {
        int mid = l + (r - l) / 2;
        if (arr[mid] == x)
            return mid;
        else if (arr[mid] > x)
            return binarySearch (arr, l, mid - 1, x);
        else
            return binarySearch (arr, mid + 1, r, x);
    }
    return (-1);
}
```

Linear Search

Iterative - Time Complexity = O(n)
Space Complexity = O(1)

Recursive - Time Complexity = O(n)
Space Complexity = O(n)

Tanishq

Date / /

Binary Search

Iterative - Time Complexity = O(log n)
Space Complexity = O(1)

Recursive - Time Complexity = ~~Θ(n)~~ O(log n)
Space Complexity = ~~Θ(n)~~ O(log n)

6.

$T(n)$
↓
$T(n/2)$
↓
$T(n/4)$
⋮
$T^k(n/2^k)$

Recurrence Relation = $T(n/2) + O(1)$

7. Find 2 indexes such that $A[i] + A[j] = k$ in min. time complexity.

```
int n;
int A[n];
int key;
int i=0, j=n-1;
while (i<j)
{
 if ((A[i] + A[j]) == key)
    break;
 else if [(A[i] + A[j]) > key]
    j--;
 else
    i++;
}
cout << i << " " << j;
```

Tanishq

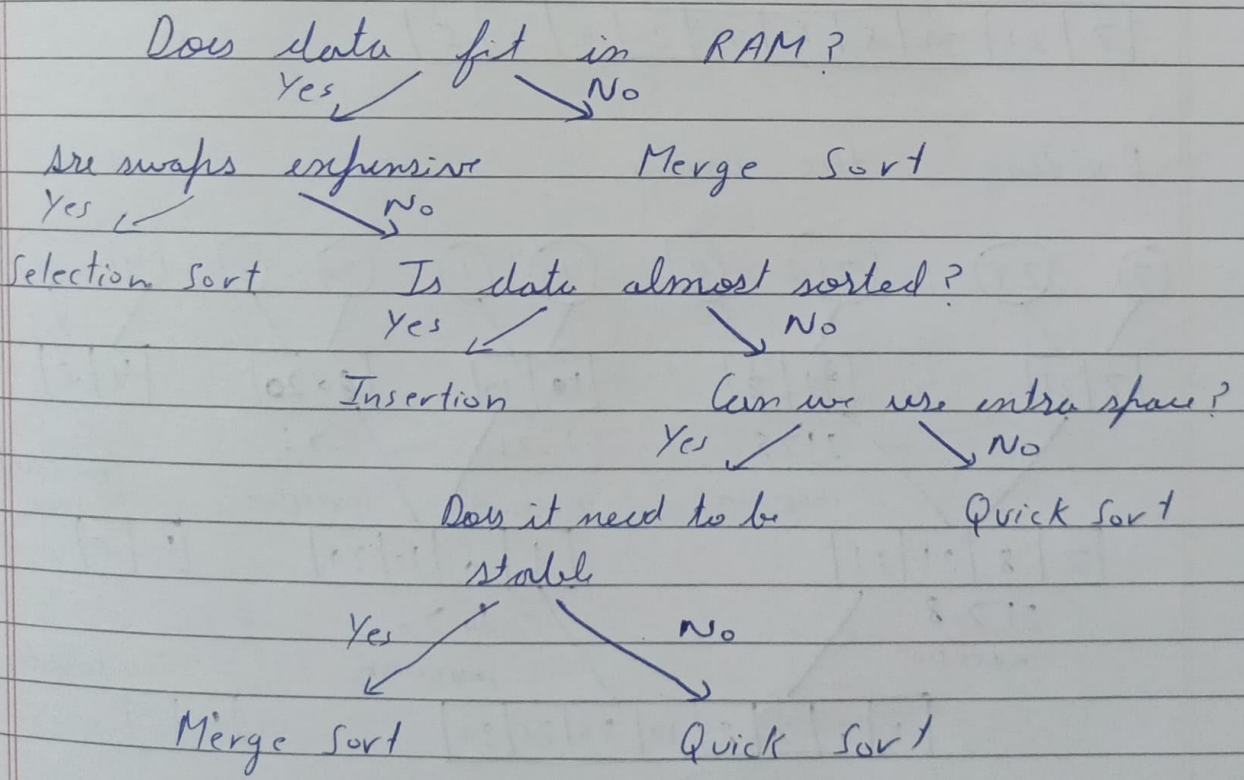
Date / /

Time complexity = O(nlogn)

8. Which sorting is best for practical use? Explain
Ans8 Factors affecting or deciding whether a sorting algorithm is good or not:-

i) Run time
ii) Space
iii) Stable
iv) No. of swaps
v) Will the data fit in RAM

There is no best sorting algorithm. It depends on the situation or the type of array provided.



Tanishq

Date / /

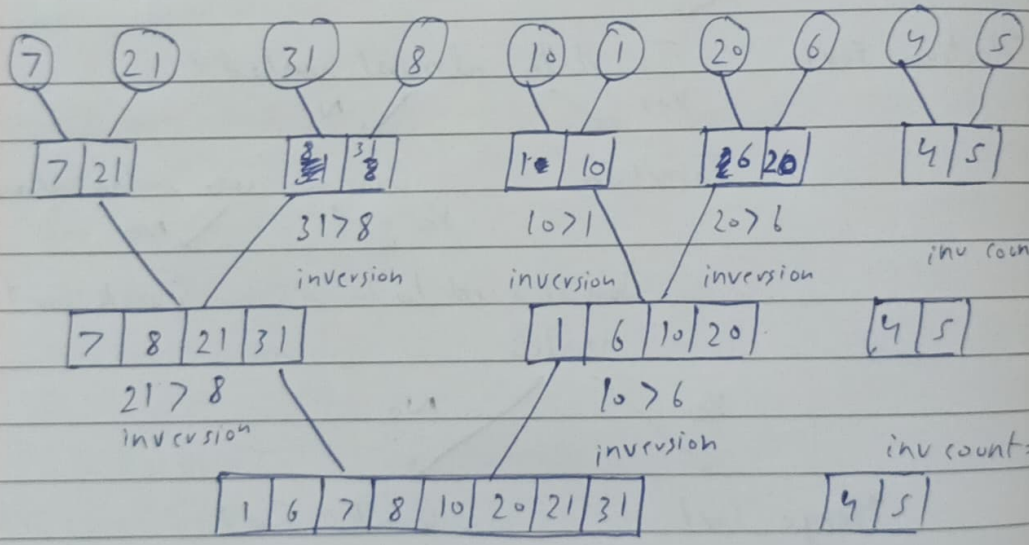
Q9. What do you mean by no. of inversions in an array? Count the no of inversions in an array arr[] = {7, 21, 31, 8, 10, 1, 20, 6, 4, 5} using merge sort.

Ans 9. Inversion in an array indicates how far the array is from being sorted. If the array is already sorted, the inversion count is 0, but if the array is sorted in reverse order, then the inversion count is maximum.

Condition for inversion:-
$a[i] > a[j]$ & $i < j$

7	21	31	8	10	1	20	6	4	5

Dividing the array



7>1, 7>6, 8>1, 8>6, 21>10, 21>20, 31>1, 31>6, 31>10, 31>20,
21>1, 21>6 total inv in this step = 12

inv count = 17

Twinkle

Date / /

1	4	5	6	7	8	10	20	21	31

6>4, 6>5, 7>4, 7>5, 8>4, 8>5, 10>4, 10>5, 20>4, 20>5, 21>4, 21>5, 31>4, 31>5

total inv in this step = 14

inv count = 31 Ans

10. In which case quick sort will give the best & worst case complexity?

Ans 10. Best Case

Time Complexity = $O(n \log n)$

The best case occurs when the partition process always picks the middle element as pivot.

Worst Case

Time Complexity = $O(n^2)$

When the array is sorted in ascending or descending order.

11. Write recurrence relation of merge sort & quick sort in best & worst case. What are the similarities & differences b/w complexities of two algorithms & why?

Ans 11

Best case

Merge Sort = $2T(n/2) + n$

Quick Sort = $2T(n/2) + n$

Worst case

Merge sort = $2T(n/2) + n$

Quick sort = $T(n-1) + n$

Tanishq

Date / /

Similarities - They both work on the concept of divide & conquer algorithm.
Both have best case complexity of $O(n \log n)$.

Differences :-

Merge Sort	Quick Sort
i) The array is divided into just 2 halves	i) The array is divided in any ratio.
ii) Worst case complexity is $O(n \log n)$	ii) Worst case complexity $O(n^2)$.
iii) It requires extra space i.e. NOT inplace.	iii) It does not require extra space i.e., inplace.
iv) It is external sorting algorithm & stable	iv) It is internal sorting algorithm & NOT stable
v) Works consistently on any size of data set.	v) Works fast on small data sets.

Q12. Selection sort is not stable by default but you can write a version of stable selection sort.

```
void selection (int A[], int n)
{
  for(int i=0; i<n-1; i++)
  {  int min=i;
     for(int j=i+1; j<n; j++)
     {
       if (A[min] > A[j])
          min=j;
       int key = A[min];
       while (min >i)
       {
```

Tanishq

Date / /

```
        A[min] = A[min-1]
        min--;
    }
    A[i] = key;
  }
}
```

Q13. Bubble sort scans the whole array even when the array is sorted. Can you modify the bubble sort algorithm so that it does not scan the whole array once it is ~~a~~ sorted?

```
void bubblesort (int A[], int n)
{
  int i, j;
  int f = 0;
  for (i = 0; i < n ; i++)
  {
    for (j = 0; j < n-1; j++)
    {
      if (A[j] > A[j+1])
      {
        swap(A[j], A[j+1])
        f = 1;
      }
    }
    if (f == 0)
    break;
  }
```

Tanishq

Date / /

Q14. Your computer has 2 GB RAM & you are to sort a 4 GB array. Which algorithm you are going to use for this purpose & why?
Also explain the concept of external & internal sorting.

Ans 14 When the data set is large enough to fit inside RAM, we ought to use merge sort, because it uses divide & conquer approach in which it keeps dividing the array into smaller parts until it can no longer be splitted. It then merges the array divided in n parts. Therefore at a time only a part of array is taken on RAM.

External Sorting- It is used to sort massive amounts of data. It is required when the data does not fit inside RAM & instead they must inside in the slower external memory.
During sorting, chunks of small data that can fit in main memory are read, sorted & written out to a temporary file.

Internal Sorting - Internal sorting is a type of sorting which is used when the entire collection of data is small enough to reside within RAM. Then there is no need of external memory for program execution
It is used when input is small.
Eg- Insertion sort, Quick sort, Heap sort, etc.