Tutorial - 3
TCS - 409

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

Ans-

```
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

Ans-

```
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;
    }
}
```

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.

Q3- Complexity of all sorting algorithms that have been discussed in class:

Ans-

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

Q4- Classify the sorting algorithms:

Ans-

Sort	Inplace	Stable	Online
Bubble	✓	✓	X
Selection	✓	X	X
Insertion	✓	✓	✓
Count	X	✓	X
Quick	✓	X	X
Merge	X	✓	X
Heap	✓	X	X

Q5- Recursive / Iterative Pseudo code for Binary Search:

Iterative

```
int binary_search(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;
        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

	Time Comp.	Space Comp.
Iterative	O(n)	O(1)
Recursive	O(n)	O(n)

Binary Search

Iterative	O(log n)	O(1)
Recursive	O(log n)	O(log n)

Q6- Write recurrence relation for binary recursive sea

Ans-

T(n)

↓

T(n/2)

↓

T(n/4)

↓

⋮

T(n/2^k)

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

Q7- Find two indexes such that $A[i] + A[j] = K$ in minimum time complexity.

Ans-
```
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;
```

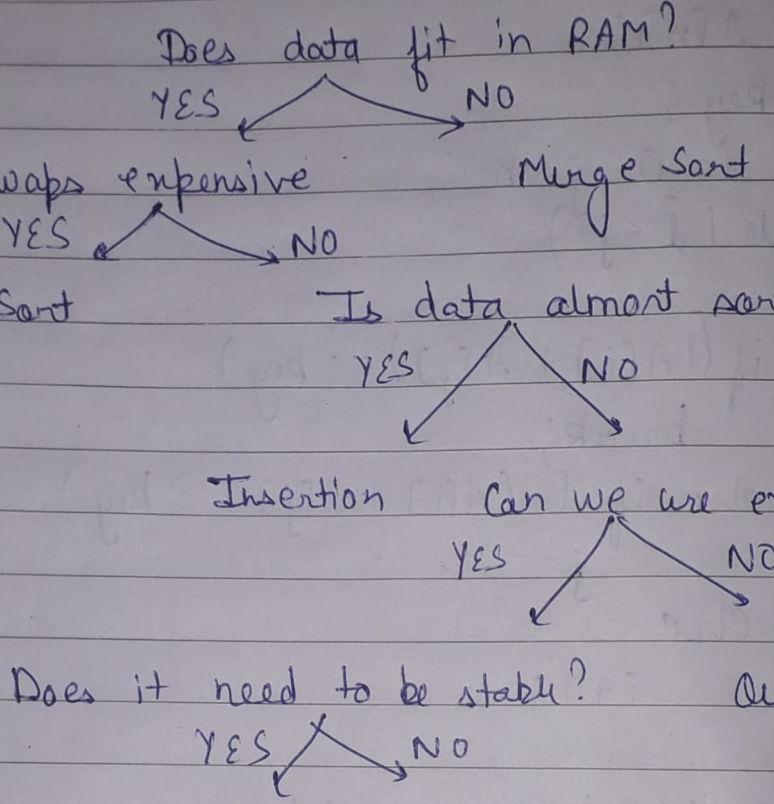
Time complexity = $O(n \log n)$

Q8- Which sorting is best for practical use? Explain

Ans- Factors that affect a sorting algorithm are:

i) Run Time
ii) Space
iii) Stable
iv) No. of swaps
v) Data size compatible with RAM.

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



Merge Sort Quick Sort

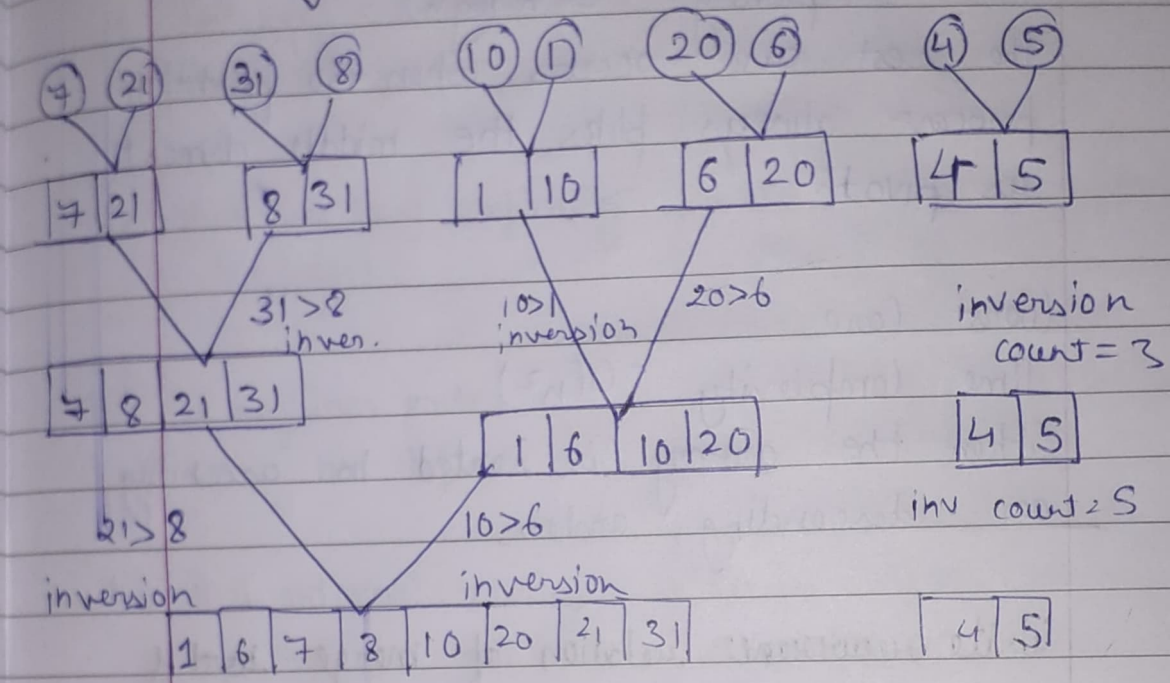
Q9 What do you mean by no. of inversions in an array? Count the no. of inversions in an array
arr[] = {7, 21, 31, 8, 10, 1, 20, 6, 4, 5}
using Merge Sort.

Ans 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 the inverse 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

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

Inversion count = 31

Q10- Which case quick sort will give the best and worst case complexity

Ans-

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.

Q11- Write recurrence relation of merge sort & quick sort in best & worst case. What are the similarities & differences between complexities of two algorithms & why?

Ans-

Best Cases
Merge Sort = $2T(n/2) + n$
Quick Sort = $2T(n/2) + n$

Worst Case
Merge Sort = $2T(n/2) + n$
Quick Sort = $T(n-1) + n$

Similarity: They both work on the concept of divide & conquer technique
- Both have same best case complexity

	Merge Sort	Quick Sort
1)	The array is divided into just two halves	The array is divided in any ratio.
2)	Worst case time complexity $O(n \log n)$	Worst case Time Complexity $O(n^2)$
3)	It requires extra space i.e., not inplace	It does not require extra space i.e, inplace
4)	It is external sorting algorithm & stable	It is internal sorting algorithm & not stable
5)	Works consistently on any size of dataset	Works fast on small datasets.

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

Ans-
```
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)  // do shifting instead
    {                // of swapping
        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 its algorithm so that it does not scan the whole sorted array.

Ans

```
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 (j == 0)
  break;
}
```

Your computer has 2GB RAM & you are to sort a 4 GB array.
Which algorithm should be used & why?
Also explain internal & external sorting.

Ans When the data set is large enough to fit inside RAM, we ought to use MergeSort because it uses the 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.
∴ at a time only a part of array is taken into RAM.

External Sorting

It is used to sort massive amounts of data. It is required when the data doesn't fit inside RAM & instead they must reside 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.

During merging, the sorted subfiles are combined into a single large file.

Internal Sorting

It is a type of sorting which is used when the entire dataset is small enough to reside within RAM.
Then there is no need of external memory for program execution.
It is used when i/p is small.

eg. Insertion Sort, Quick Sort, Heap Sort etc.