Tutorial-3.

Ans1.

```
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;
    }
```

Ans2.

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;
     }
  }
```

Recursive Insertion Sort

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

Insertion Sort is also called online sorting algorithm bcoz 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.

DATE / /
PAGE No.

Ans 3.

	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	$O(n\log n)$	$O(n\log n)$

Ans 4.

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

Ans 5. Iterative binary Search

```
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;
        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 binary Search (arr , l, m-1, x);
        else
        return binary search (arr , m+1, x);
    else
    return binary Search ( arr, mid +1 , r, x);
    }
    return (-1);
}
```

Linear Search

Iterative :- Time Complexity = O(n)

Space Complexity = O(1)

Recursive :- Time complexity = O(n)

Space Comp. = O(n)

Binary Search

Iterative = Time Complexity = O(logn)

Space Comp = O(n)

Recursive → Time Complexity = O(logn)

Space Complexity = O(logn)

Ans6. $T(n)$

$\downarrow$

$T(n/2)$

$\downarrow$

$T(n/4)$

$\downarrow$

$\vdots$

$T(n/2^n)$

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

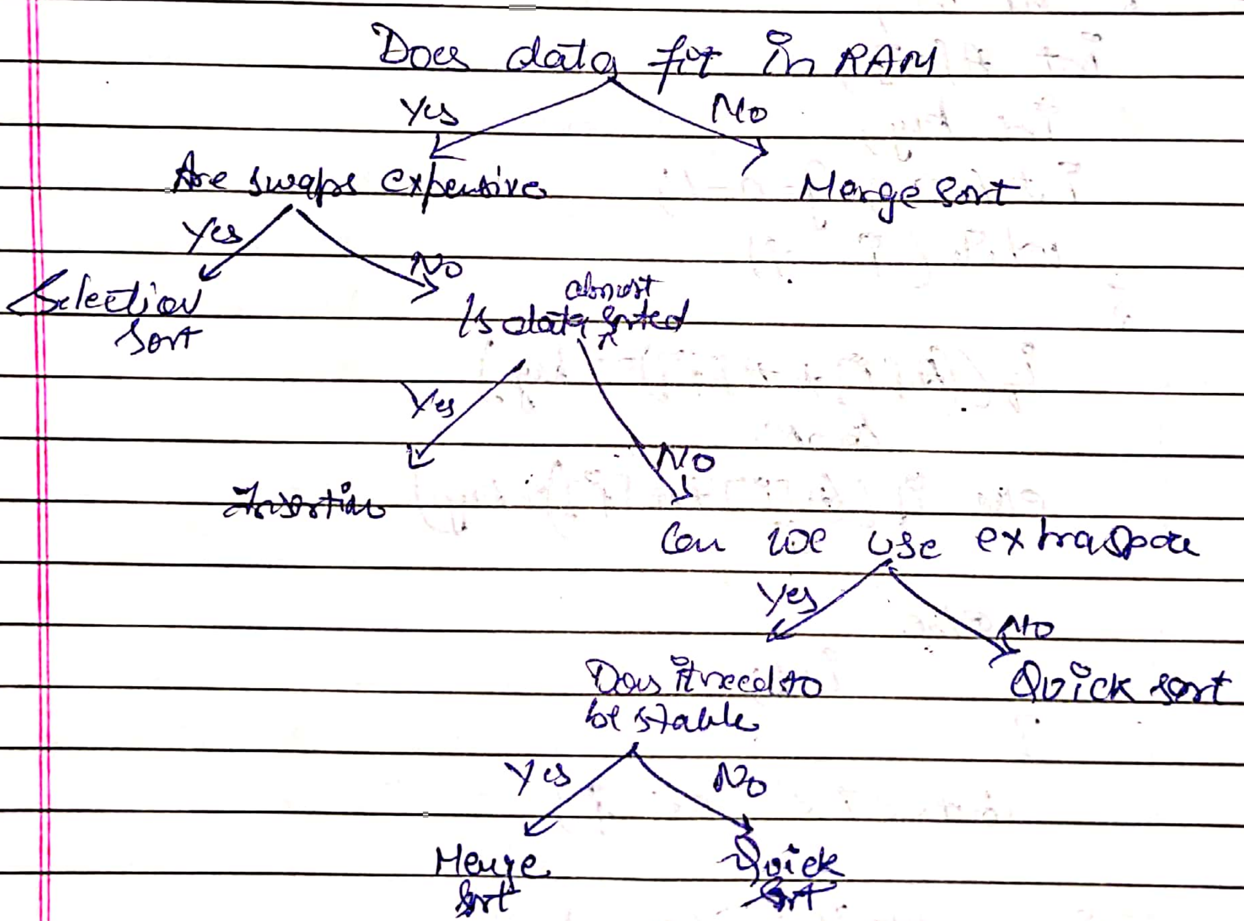
Ans7.
```
{ int n;
  int A[n];
  int key;
  int i=0 ; j=n-1;
  while (i<j)
  {
    if((A[i] + A[j])==key)
        break;
    else if ((A[i]+A[j])>key)
        j--;
    else
        i++;
  }
  Cout<<i<<" "<< j;
```

Time Complexity = O(nlogn)

Ans. Factors affecting or deciding whether a sorting algorithm is good or not:-

1. Run time
2. Space
3. Stable
4. No. of swaps
5. Will the data fit in RAM

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



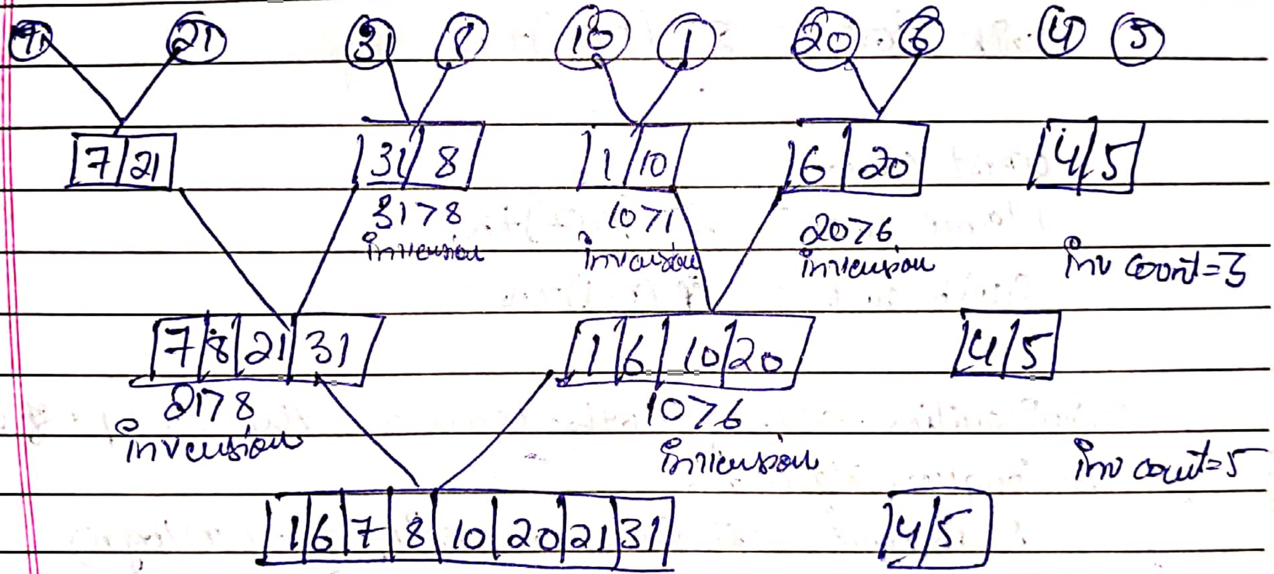
Ans 9. Inversion is an array indicates how far the array is from being sorted. If the array is already sorted, the inversion count is 0. But if the array is sorted in reverse order, then the inversion count is maximum.

Condition for inversion

$a[i] > a[j]$ and $i < j$

7	21	31	8	10	1	20	6	4	5

Divide 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

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 = 14

inv count = 31

Ans10. 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 Comp. = $O(n^2)$.

When the array is sorted in ascending or descending order.

Ans11. Best Cases

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

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

Worst case:

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

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

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
1.	The array is divided into just two halves	The array is divided in any ratio.
2.	Worst case complexity is $O(n\log n)$.	Worst case 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 data set	Works fast on small data sets.

Ans 2. 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)        // shift instead of swap
        {
            A[min] = A[min-1];
            min --;
        }
        A[i]=key;
    }
}
```

Ans 3. 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 sorted?

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

Que: Your computer has 2GB RAM & you are to sort a 4GB array. Which algorithm you are going to use for this purpose & why?

When the data set is large enough to fit inside RAM, we ought to use Merge Sort. 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 merged 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 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

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, quick, heap sort.