DATE. / /

Tutorial - 3

Ans 1

```
void linearSearch (int A[], int n, int key)
{ int flag = 0;
   for (int i=0; i<n; i++)
    {
        if (A[i] == key)
        {  flag = 1;
           break;
        }
    }
   if (flag == 0)
      cout << "Not found";
   else
      cout << "found";
}
```

Ans 2

Iterative:

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

PAGE NO.
DATE. / /

Recursive :-

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

Insertion Sort is an online algorithm Because insertion sort considers one input element per iteration and produces a partial solution without considering future elements.

But in case of other sorting algorithm, we require access to the entire input, thus they are offline algorithm.

Ans 3.

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

Ans 4.

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

Ans 5. Recursive:-

```
int binarySearch(int arr[], int l, int r, int key)
{
    if (r>=l)
    { int mid = l+(r-l)/2;
```

```
if (arr[mid] == key)  return mid;
  if (arr[mid] > key)
  return BinarySearch (arr, l, mid-1, key);

  return binarySearch (arr, mid+1, r, key);
  }
    return -1;
}
```

Iterative:

```
     int BinarySearch (int arr[], int l, int r, int key)
{
  while ( l<=r)
  { int m= l+ (r-l)/2;
     if (arr[m]==key)
         return m;
    if (arr[m] > key)
         l= m+1;
    else
         r= m-1;
  }
    return -1;
}
```

	Time Complexity		Space Complexity	
	Recursive	Iterative	Recursive	Iterative
Linear Search	O(n)	O(n)	O(1)	O(1)
Binary Search	O (logn)	O(logn)	O(1)	O(1)

PAGE NO.
DATE. / /

Ans 6:

Recurrence relation for binary recursive search :-

$$T(n) = T(n/2) + 1.$$

Ans 7.

```
void sum (int A[], int K, int n)
{
    sort (A, A+n);
    int i=0, j=n-1;
    while (i<j)
  {  if  (A[i] + A[j]==K)
          break;
     else if (A[i]+ A[j]>K)
          j--;
     else
          i++;
   }
  print (i,j).
}
```

Here sort function has $O(n\log n)$ complexity and for while loop it is $O(n)$.

$\therefore$ Overall complexity = $O(n\log n)$.

Ans 8 In practical uses, we mostly prefer merge sort because of its stability and it can best for very large data. further more, the time complexity of merge sort is same in all cases that is O(nlogn).

Ans 9 Inversion count for an array indicates — how far (or close) the array is from being sorted. If the array is already sorted, inversion count is 0, but if the array is sorted in reverse order the inversion count is maximum.

Ans 10 When the array is already sorted or sorted in reverse order quick sort gives the worst case time complexity i.e, $O(n^2)$. But when the array is totally unsorted, it will give best case time complexity, i.e, O(nlogn).

Ans 11

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

Both the algorithms are based on the divide and conquer algorithm. Both the algorithms have the same time complexity in the best case and average because both the algorithms

PAGE NO.
DATE. / /

divide array into sub-parts, sort them and finally merge all the sorted parts.

Ans 12

As the selection sort is not stable because it changes the relative position of some elements after sorting.
Selection sort can be made stable if instead of swapping, the minimum element is placed in its position without swapping i.e, by placing the number in its position by pushing every element one step forward. In simple words use insertion sort technique which means inserting element in its correct place.

pseudo code for stable selection sort:-

```
void stableSelectionSort (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)
  {
     a[min]= a[min-1]
       m--; }
     a[i] = key; }}
```

PAGE NO.
DATE. / /

Ans13. We can modify bubble sort by placing a flag variable. If array is already sorted we can halt the process by checking the flag variable if its value changes or not.

Pseudo Code for Modified Bubble Sort:-

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

Ans14. For the array of 4 GB, we use the external sorting because array size is greater than the RAM of computer.

→ External Sorting:- These are sorting algorithms that can handle large data amounts which cannot fit in the main memory. Therefore only a part of the Array resides

PAGE NO.
DATE. / /

in the RAM during execution.
E.g:- K-Way Merge Sort.

→ Internal Sorting:-
These are sorting algorithms where the whole array needs to be in the RAM during execution.
E.g:-
Bubble Sort, Selection Sort, etc.