Tutorial-3

Name:- Bandana Kushwaha
University Rollno:- 2014380
Sec:- A

Ans1:- linear search in a sorted array with minimum no. of comparisons -

```
int linear_search (int a[], int n, int data)
{
    for i=0 to n-1 {
        if (a[i] == data)
            return i;
        else if (a[i] > data)   // array is sorted if a[i] > data then no need to search the rest of the array.
            return -1;
    }
}
```

T.C : Best = O(1), Avg, Worst = O(n)
Space.C : O(1)

Ans2:- Pseudo code for iterative insertion sort -

```
void insertion_sort (int arr[], int n){
    int i, temp, j;
    for i ← 1 to n
        temp ← arr[i];
```

```
        j ← i-1;
    while (j>=0 && arr[j] > temp){
            arr[j+1] = arr[j];
            j ← j-1;
        }
    arr[j+1] = temp;
}
```

Pseudo code for recursive insertion Sort —

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

An online sorting algo is one that will work if the elements to be sorted are provided one at a time with the understanding that the algo. must keep the sequence sorted as more & more elements are added in. Insertion sort considers one input element per ## iterations & produces a partial solution without considering future elements. Thus insertion sort is online.

Other algo. like selection sort repeatedly selects the minimum element from the unsorted array & places it at the first which requires the entire input. Similarly bubble, quick & merge sorts also require the entire input. Therefore they are offline algo.

Ans 3,4 :-

Sorting algo	Time: Best	Time: Avg.	Time: Worst	Space: Worst	Stable	Inplace
Bubble	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$	✓	✓
Selection	$O(n^2)$	"	"	"	✗	✓
Insertion	$O(n)$	"	"	"	✓	✓
merge	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	$O(n)$	✓	✗
Quick	$O(n \log n)$	"	$O(n^2)$	$O(n)$	✗	✗
Heap	$O(n \log n)$	"	$O(n \log n)$	$O(1)$	✗	✓

Ans 5 :- Iterative pseudo code for binary search :-

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

Time Complexity - Best ~~time~~ case: $O(1)$

Avg, worst: $O(\log_2 n)$

Space: $O(1)$

Binary search recursive code:-

```
int binarySearch(int arr[], int l, int r, int x){
    if(r >= l){
        mid ← (l+r)/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;
}
```

T.C ⇒ Best: $O(1)$ & Avg, worst = $O(\log_2 n)$

Space Comp. ⇒ Best: $O(1)$

Avg, worst $O(\log_2 n)$

Ans 6:- Recurrence relation for binary search

$T(n) = T(n/2) + 1$, where $T(n)$ is the time required for binary search in an array of size n.

Ans7:- using two-pointers we can find these -

```
void find (int arr[], int n, int x) {
    int j, k;
    Sort (arr, arr+n);
    j=0;
    k= n-1;
  while (j<k)
{
  if (arr[j]+ arr[k] == x)
      cout << j << " " << k;
  else if (arr[j]+ arr[k] < x)
     j++;
  else
     k--;
}
}
```

Ans8:- Quick sort is best. If stability is important & space is available, then mergesort may be the best.

Ans9:- Inversions count:-

Inversion count indicates how far or close the array is from being sorted. If already sorted, inversion count is 0, if sorted in reverse order, inversion count is maximum.

i.e, for every $i < j \rightarrow a[i] > a[j]$ forms an inversions.

In array arr = {7, 21, 31, 8, 10, 1, 20, 6, 4, 5}

Inversion count = 31.

Ans 10:- Worst case happens when array is sorted. &
Best case happens when all elements are equal
& when we select pivot as mean element.

Ans 11:- Recurrence relation of -

Merge Sort (Best case) :- $T(n) = 2T\left(\frac{n}{2}\right) + n-1$

Quick Sort (Best Case) :- $T(n) = 2T\left(\frac{n}{2}\right) + n-1$

Merge Sort (Worst case) :- $T(n) = 2T\left(\frac{n}{2}\right) + n-1$

Quick Sort (Worst case) :- $T(n) = \frac{n(n-1)}{2}$

Ans 12:- Selection Sort can be stable if instead of swapping, the minimum element is placed in its position without swapping i.e, by placing the no. in the position by pushing every element one step forward.

Ans 13:-

```
void bubbleSort (int arr[], int n)
{
    int i, j;
    bool swapped;
    for (i=0; i<n-1; i++) {
        swapped = false;
        for (j=0; j<n-i-1; j++)
        {
```

```
        if (arr[j] < arr[j+1]) {
            swap (arr[j], arr[j+1])
                swapped = true;
        }
    }
    if (swapped == false)
        break;
}
```

Ans14:- Since, data cannot be fit into RAM, we have to use external sorting method like the merge sort.

Internal Sorting:- A data sorting process that takes place entirely within the main memory of a computer.

Eg Bubble Sort.

External Sorting:- A data sorting process that takes place when data being sorted do not fit into main memory.

Eg merge Sort.