Name - Vansh Kalra
Section - CST
Class Roll No. - 25
Univ. Roll No. - 2017576

Dt. ________
Pg. ________

DAA Tutorial - 3

Q1.
```
for (i=0 to n)
{
    if (arr[i]== value)
        //element found
}
```

Q2. Recursive
```
    void insertion (int arr[], int n)
    {
        if (n<=1)
            return;
        insertion (arr, n-1);
        int nt= arr[n-1];
        int j=n-2;
        while (j>=0 && arr[j]>nt)
        {
            arr[j+1]= arr[j];
            j--;
        }
        arr[j+1]=nt;
    }
```

Iterative
```
    void insertionSort (int arr[], int n)
    {
        for (i=1 to n)
        {
            key = arr[i]
            j=i-1
            while (j>=0 and ~~A[j]~~ arr[j] >key)
```

Dt. ____
Pg. ____

```
    {
        arr[j+1] = arr[j];
        j--;
    }
    arr[j+1] = key;
  }
}
```

Insertion sort is online sorting because it doesn't know the exact input, more elements can be inserted into array while it is running.

Q3.

Name	Best	Worst	Average
Selection sort	$O(n^2)$	$O(n^2)$	$O(n^2)$
Bubble sort	$O(n)$	$O(n^2)$	$O(n^2)$
Insertion sort	$O(n)$	$O(n^2)$	$O(n^2)$
Heap sort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$
Quick sort	$O(n \log n)$	$O(n^2)$	$O(n \log n)$.
Merge sort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$.

Q4 Inplace sorting - Bubble sort, selection sort, insertion sort, Quick sort, Heap sort.

Stable sorting - Merge sort, bubble sort, insertion sort, count sort.

Online sorting - Insertion sort.

Dt.

Pg.

Q5. Recursive

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

Iterative

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

Time complexity - Binary search - O(log n)
Linear search - O(n).

Q6. Recurrence relation for recursive binary search =>
$T(n) = T(n/2) + 1$
where T(n) is the time required for binary search on array of size n.

~~Q7. int find (int A[], int n, int target)~~
~~{~~

```
~~int max = maximumElement(A, n);~~
~~int p[max+1];~~
```

Q7. int find(int A[], int n, int target)

```
{
    sort (A, n);
    ~~for(i=0, j=n-1; i<j; i+~~
    i=0, j=n-1;
    while (i<j)
    {
        if (A[i]+ [j]== target)
            return 1;
        else if (A[i]+A[j] < target)
            i++;
        else
            j--;
    }
    return 0;
}
```

Time complexity - O(nlogn) + O(n)
=> O(n logn).

Q8. • Quicksort is the fastest general purpose sort.
• In most practical situations, quicksort is the method of choice. If stability is important & space is available, merge sort might be best.

Q9. A pair (a[i], a[j]) is said to be inversion if a[i] > a[j].

In arr [] = {7, 21, 31, 8, 10, 1, 20, 6, 4, 5}
Total no. of inversions using merge sort = 31.

Q10. The worst case time complexity of quick sort is $O(n^2)$. This case occurs when the picked pivot is always an extreme (smallest or largest) element. This happens when input array is sorted or reverse sorted.
The best case of quick sort is when we will select pivot as a mean element.

Q11. Recurrence relation of
Merge sort ⇒ $T(n) = 2T(n/2) + n$.
Quick sort ⇒ $T(n) = 2T(n/2) + n$.

- Merge sort is more efficient & works faster than quick sort in case of larger array size or datasets.
- Worst case complexity for quick sort is $O(n^2)$ whereas $O(n\log n)$ for merge sort.

Dt. ______

Pg. ______

Q12. Stable selection sort

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

Q13.

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