Assignment 2

Ans 1

```
int Search(int arr[], int N, int x)
{
    for(int i=0; i<N; i++)
    {
        if(arr[i]==x)
            return i;
    }
    return -1;
}
```

Ans 2

- Iterative insertion sort

```
void InsertionSort(arr, N)
{
    for(i=1; i<N; i++)
    {
        int key = arr[i]
        int j = i-1;
        while(j>=0 && arr[j]>key)
        {
            arr[j+1]=arr[j];
            j--;
        }
        arr[j+1]=key;
    }
}
```

Page No.
Date

- Recursive insertion sort.

```
void RecInsertionsort(int arr[], int N)
{
  if(N<=1)
    return;

  RecInsertionSort(arr, N-1);
 int last=arr[N-1];
 int j=N-2;
while(j>=0 && arr[j]>last)
{
  arr[j+1]=arr[j];
  j--;
 }
 arr[j+1]=last;
}
```

- Insertion sort is considered an "online" algorithm because it can efficiently handle incoming data.
As new elements arrive, they are inserted into the sorted portio of the array without requiring a full re-sort.

Ans 3

- Selection Sort
Time complexity:
Best: $O(n^2)$
Avg: $O(n^2)$
Worst: $O(n^2)$
Space complexity: $O(1)$

- Bubble Sort.
Time Complexity:
Best: $O(n)$
Avg: $O(n^2)$
worst: $O(n^2)$
Space complexity: $O(1)$

- Insertion Sort
  Time complexity:
  Best: $O(n)$
  Avg: $O(n^2)$
  Worst: $O(n^2)$
  Space complexity: $O(1)$

- Quick Sort
  Time complexity:
  Best: $O(n \log(n))$
  Avg: $O(n \log(n))$
  Worst: $O(n^2)$
  Space complexity: $O(n)$

- Heap Sort
  Time complexity:
  Best: $O(n \log(n))$
  Avg: $O(n \log(n))$
  Worst: $O(n \log(n))$
  Space complexity: $O(1)$

- MergeSort
  Time complexity
  Best: $O(n \log(n))$
  Avg: $O(n \log(n))$
  Worst: $O(n \log(n))$
  Space complexity: $O(n)$.

- Count Sort
  Time complexity
  Best: $O(n+k)$
  Avg: $O(n+k)$
  Worst: $O(n+k)$
  Space complexity: $O(k)$

- Radix Sort
  Time complexity:
  Best: $O(nk)$
  Avg: $O(nk)$
  Worst: $O(nk)$
  space complexity: $O(n+k)$.

Ans 4

- In-place sorting Algorithms
  - Bubble sort
  - selection Sort
  - insertion sort
  - Heap sort.

- online sorting Algorithms
  - Insertion sort.

- Stable sorting Algorithms
  - Merge sort
  - Counting sort
  - Insertion Sort

$c = \log_2 3$ → $\theta(n)$. Ans.

Ans 5.6

Linear Search.

```
for( i=0 ; i<n ; i++)
{
    if (arr[i] == key)
        return i;
}
return -1;
```

- T.C. = O(n).
- Best → O(1)
- Avg → O(n).
- S.C. = O(1).

Binary Search.

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

(l = 0, r = n-1)

MAARS

Page No.
Date

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

T.C. → $O(\log n)$
Best → $O(1)$
S.C. → $O(\log n)$.

iterative way.

```
bool binarysearch(int arr[], int l, int r, int key).
{
  while (l<=r)
  {
    int mid = l+(r-l)/2;
    if (arr[mid] == key)
        return true;
    else if (arr[mid] < key)
        l = mid+1;
    else
        r = mid-1;
  }
  return false;
}
```

TC → same
B.C. → Same.
S.C → O(1).

Page No.
Date

Outline Sorting Algorithms.
- insertion sort.

Ans 7

```
vector<int> twoSumIndices (vector<int>& A, int K)
{
    unordered_map <int, int> element_indices;
    for(int i=0; i< A.size(); i++)
    {
        int Complement = K - A[i];
        if(element_indices.find(complement) !=
                element_indices.end())
        {
            return {element_indices[complement], i};
        }
        element_indices[A[i]]=i;
    }
    return{};
}
```

Ans 8 Quicksort is one of the most efficient sorting algorithm making it widely used. In practice quicksort is often favoured due to its cache efficiency and overal performance. However the choice of sorting algorithm depends on the specific context, data distribution and available memory.

Page No.
Date

Ans 9 An inversion occurs when two elements in an array are out of order relative to their indices. specifically,
- for elements arr[i] and arr[j], if arr[i] > arr[j] and i < j then they form an inversion.
- or an inversion represents how far (or close) the array is from being sorted.

```
int mergeAndCount(int arr[], int temp[], int left, int mid, int right)
{
    int inv_count = 0;
    int i = left;
    int j = mid+1;
    int k = left;
    while(i<= mid && j<=right)
    {
        if(arr[i]<=arr[j])
            temp[k++] = arr[i++];
        else
        {
            temp[k++] = arr[j++];
            inv_count += (mid-i+1);
        }
    }
    while (i<=mid)
        temp[k++] = arr[i++];
    while(j<=right)
        temp[k++]=arr[j++];
    for(int p=left; p<=right; p++)
    {
        arr[p] = temp[p];
    }
    return inv_count;
}
```

```
int mergeSortAndCount(int arr[], int temp[], int left, int right)
{
    int inv_count = 0;
    if (left < right)
    {
        int mid = (left + right) / 2;
        inv_count += mergeSortAndCount(arr, temp, left, mid);
        inv_count += mergeSortAndCount(arr, temp, mid + 1, right);
        inv_count += mergeAndCount(arr, temp, left, mid, right);
    }
    return inv_count;
}
```

Ans 10

Best Case

- The best case occurs when we select the pivot as the mean (ie. the middle element).
- In this case, the partioning process evenly divides the array into two subarrays

Best. T.C. for Quick sort is $O(n \log(n))$

- Worst Case
  - The worst case occurs when the array gets divided into two parts one with $(N-1)$ elements and the other with 1 element.
  - for each recursive call, the pivot selection results in an unbalanced partition.

Worst case T.C. is $O(n^2)$

Page No.
Date

Ans 4) • Recurrance Relation for Best case Merge Sort.

$$T(n) = 2T\left(\frac{n}{2}\right) + \theta(n).$$

• Recurrance Relation for worst case merge sort.

$$T(n) = 2T\left(\frac{n}{2}\right) + \theta(n).$$

• Recurrance Relation for best case Quick Sort.

$$T(n) = 2T\left(\frac{n}{2}\right) + \theta(n)$$

• Recurrance Relation for worst case Quick sort.

$$T(n) = T(n-1) + \theta(n).$$

→ Similarities and differences.

1) Both merge sort & quick sort use a divide-and-conquer approach.
2) In their best cases, both algorithms have a time complexity of $O(n \log n)$.

Differences

• worst case complexity

Merge sort: $O(n \log n)$

quick sort: $O(n^2)$

• Stability

Merge sort is a stable sorting algorithm

Quick sort is not inherently stable due to its partioning process.

Page No.
Date

Ans 12

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

Ans 13

```
void optimizedBubblesort(int arr[], int n)
 { int lastswap = n-1;
  for(int i=1; i<n; i++)
   { bool issorted = true;
    int currentswap = -1;
   for(int j=0; j<lastswap; j++)
    { if(arr[j] > arr[j+1])
      { swap(arr[j], arr[j+1]);
        issorted = false;
       currentswap = j;
      }
    }
  if(issorted)
    return;
}lastswap = currentswap;
```