Date.
Page No.

Assignment 02

A1.
```
int binarySearch (int arr[], int Key, int s)
{
  for (int i=0; i<s; i++)
  {
     If (arr [i] == Key)
        return i;
     else if (arr [i] > Key)
     {
        return -1;
     }
  }
  return -1;
}
```

A2.
```
void insertion sort (int arr[], int s)
{
   for (int i=1; i< size; i++)
   {
      int j = i-1;
      while (j>0 && arr[j] > Key)
      {
         arr[j+1] = arr[j]
         j=j-1;
      }
      arr [j+1] = Key;
}
```

(ii) Insertion sort recursive

```
void insertion (int arr[], int n)
{
    if (n==1)
        return;
    insertion (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 considers one input element per iteration and produces partial solution without considering future elements. Thus insertion sort is an online algorithm.

Complexity of all sorting Algorithm

Date.
Page No.

	Best Case	Avg. Case	Worst	Space
Bubble Sort	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$
Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$
Insertion Sort	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$
Merge Sort	$O(n\log n)$	$O(n\log n)$	$O(n\log n)$	$O(n)$
Quick Sort	$O(n\log n)$	$O(n\log n)$	$O(n^2)$	$O(n)$
Heap Sort	$O(n\log n)$	$O(n\log n)$	$O(n\log n)$	$O(1)$

A4. Inplace: Sort the input array by rearranging the Elements within the array itself for eg.
→ Bubble Sort
→ Selection Sort
→ Insertion Sort

(2) Stable: pressure the relative order of Equal Elements in the array itself. Elements with the Same value are Sorted in Same order.
→ Bubble Sort
→ Insertion Sort
→ Merge Sort
→ Count Sort

Date.
Page No.

(3). Online : sorts the ~~...~~ name of elements as they arrive
→ Insertion sort

A5. Recursive code for binary search

```
int binary (int arr[], int l , int r, int x)
{
    if (r>=l)
        int mid = l +(r-l)/2;
        if (arr [mid]==x)
            return mid;
        if (arr [mid ] >x)
            return binary (arr,
                        mid +1, r, x);
    }
    return -1;
}
```

Iterative code for binary search

```
int binary (int arr [], int n, int x)
{
    int l=0 , r = n-1;
    while (l< =r)
    {
        int mid = l+(r-l)/2;
        if (arr[mid]==x)
            return mid;
```

```
if (arr[mid] < x)
        l = mid+1;
    else
        r = mid-1;
    }
    return -1;
}
```

	Best Case	Avg.	Worst	Space comp.
Binary Search (Recursive)	O(1)	O(logn)	O(logn)	O(logn)
Binary search (Iterative)	O(1)	O(logn)	O(logn)	O(1)
Linear Search	O(1)	O(n)	O(n)	O(n)
Linear (Iterative)	O(1)	O(n)	O(n)	O(n)

A6. The recurrence relation expresses the time complexity of binary search algorithm into terms of its sub problems.

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

$T(n)$ = array size is n
$T(n/2)$ = array size is $n/2$
$O(1)$ = is the time complexity for comparing middle element to target element

A7. Step 1: Sort the input array in non decreasing order

Step 2: Initialize true pointers i & j to point to first & last element of array respectively.

Step 3: While i < j, compute A[i] + A[j]

Step 4: if sum == K return i & j

Step 5: if sum < K inc i by 1.

Step 6: if sum > K dec i by 1.

TC = O(n).

A8. Quicksort is widely used algorithm that has an avedge TC of O(nlogn). It is faster than other popular Sorting Algorithm. It is efficient for large dataset and saves memory.

A9. array { 7, 21, 31, 8, 10, 1, 20, 6, 4, 5 }

```
int getinvcount (int arr[], int n)
{
    int invcount = 0;
    for (int i=0; i<n-1; i++)
        for (int j = i+1; j<n; j++)
```

Date.
Page No.

```
{
    if (arr[i] > arr[j])
        invcount++;
   }
  }
 return invcount;
}
```

A10. Best case: The pivot element chosen should be the median of an array. if pivot is median then array is divided in two sub array of equal size. TC of this case is $O(\log n)$

Worst case: The pivot element is either the largest or smallest. TC $= O(n^2)$

A11. for MergeSort.
Best Case $\rightarrow T(n) = 2T(n/2) + O(n)$
Worst case $\rightarrow T(n) = 2T(n/2) + O(n\log n)$

for quicksort

Best Case $\rightarrow$ $2T(n/2) + O(n)$
Worst Case $\rightarrow$ $T(n-1) + O(n)$

A12: Yes, it is possible

```
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 [j] < arr [min])
      { min = j;
      }
  }
int temp = arr [i];
  arr [i] = arr [min];
    arr [min] = temp;
      }
    }
```

A13:

```
void bubble sort (int arr [], int n)
  {
    bool swapped;
    for (int i=0; i<n; i++)
      {
        swapped = false;
      for (int j=0; j<n-i-1; j++)
        {
          swap (arr[j], arr [j+1]);
          swapped = true;
        }
      }
    if (! swapped )
      {
        break;
      }
  }
```

A14. It is not possible to load the entire array into memory for sorting algorithm using internal sorting. In this case we should need to use External sorting algorithm that operate on disk of memory.