Date.
Page No.

Assignment 2

Q1 Write linear Search Pseudocode to Search an Element in a Sorted array with minimum Comparisions.

Ans 1 Pseudocode for insertion Sort

```
#include <iostream>
using namespace std;
int linearSearch (int arr[], int key , int size){

for (int i=0; i< size ; i++){
    if (arr[i] == key ){
        return i;
    }
    else if (arr[i] > key) {
        return -1; // element not found
    }
}
return -1;
}
```

Date.
Page No.

Assignment - 2

Q1 Write linear Search Pseudocode to Search an element in a Sorted array with minimum Comparisions.

Ans 1 Pseudocode for insertion Sort

```
#include <iostream>
using namespace std;
int linearSearch (int arr[], int key, int size){

for (int i=0; i< size ; i++){
    if (arr[i] == key ){
            return i;
    }
    else if (arr[i] > key) {
            return -1; // element not found
    }
}
    return -1;
}
```

Date.
Page No.

Q2 Write Pseudocode for iterative and recursive insertion Sort. Insertion Sort is called only online Sorting why? What about other Sorting algorithms that has been discussed in lectures?

Ans 2 (a) Insertion Sort iterative

```
# include <iostream>
using namespace std;

void insertion_iter(int arr[], int size){

        for(int i=1 ; i<size ; i++) {
     int key = arr[i];
     int j   = i-1;
     while (j>=0 && arr[j] >key) {

          arr [j+1 ]= arr( j];
          j= j-1;

      }
     arr[j+] = key;
   }
}
```

(b) Insertion Sort recursive

```
void insertion - recursive (int arr[], int n) {
      if( n<=1)
            return;
```

Date
Page No.

```
insertion_recur(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;

   }
```

Sometimes called as "online Sorting algorithm" because it can sort list of elements as they are being recieved one at a time, without having to wait for the entire list to be recieved or processed first.

Date.
Page No.

Q3

Ans Complexity of all Sorting Algorithms:

	Best	Avg	Worst	Space Complexity
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		$O(n \log n)$	$O(n^2)$	$O(n)$
Heap Sort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	$O(1)$

Q4

Ans (i) Inplace: Sorts the input array by rearranging the elements within the array itself. For eg:

- Bubble Sort
- Selection Sort
- Insertion Sort

Date.
Page No.

(ii) Stable: Preserves the relative order of equal elements in the array. Elements with the same value are sorted in same order.

- Bubble Sort
- Insertion Sort
- Merge Sort
- Count Sort

(iii) Online: Sorts the stream of elements as they ~~carries~~ arrives.

- Insertion Sort

Q5

Ans 5 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, l, mid-1, x);

    return binary(arr, mid+1, r, x);
}
return -1;
}
```

Date.
Page No.

* Iterative Code:

```
int bin( 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	Avg	worst	Space Complexity
Binary Search (Recursive)	O(1)	O(log n)	O(log n)	O(log n)
Binary (Iterative)	O(1)	O(log n)	O(log n)	O(1)
Linear Search (Recursive)	O(1)	O(n)	O(n)	O(n)
Linear (Iterative)	O(1)	O(n)	O(n)	O(1)

Date.
Page No.

Q6

Ans 6 The recurrence ~~rela~~ relation expresses the ~~th~~ time complexity of the binary search algorithm in terms of its sub-problems. The algo divides the input array in half each iteration and solves a sub problem of size n/2.

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

where:

$T(n) \rightarrow$ array size is n (Time complexity)

$T(n/2) \rightarrow$ array size is (n/2)

Q7

Ans 7 The following algorithm is suitable for the given task:

Step 1: Sort the input array in non-decreasing order

Step 2: Initialize two pointers i and j, to point to the first and last elements of the array respectively.

Step 3: while $i < j$, compute the sum $A[i] + A[j]$.

Step 4: If sum == k, return i and j

Step 5: If sum < k, increment i by 1.

Step 6: If sum > k, decrement j by 1

The time complexity of the above algo is O(n).

Date.
Page No.

Q8
Ans Quick Sort is widely used Sorting algorithm that has an average time Complexity of $O(n \log n)$ and is often faster than other ~~people~~ popular Sorting algorithms

QuickSort is particularly efficient for large datasets and can be easily implemented in place to save memory.

However, its worst case time complexity is $O(n^2)$ which can occur when the input data is already sorted.

Q10
Ans10 * **Best Case**: The element chosen should be the median of the array. If the element is chosen as the median at each step, then the partioning step will divide the array into two sub arrays of equal size, resulting in balanced tree of recursive calls. In this case, the time complexity of Quick Sort is $O(n \log n)$.

* **Worst Case**:
In worst case the element chosen at each step is either largest or smallest element in the sub-array.

Note that: worst occurs when the array is already sorted or reverse sorted. Time Complexity will be $O(n^2)$

Date.
Page No.

Q11

Ans Recurrence Relation for merge Sort

Best Case → $T(n) = 2T(n/2) + O(n)$

Worst Case → $T(n) = 2T(n/2) + O(n\log n)$

Recurrence Relation for Quick Sort

Best Case → $T(n) = 2T(n/2) + O(n)$

Worst Case → $T(n) = T(n-1) + O(n)$

Similarities between the two algorithms is that they have Same average and case time Complexity i.e $O(n\log n)$.

Q12

Ans Yes, it is possible to implement the Stable version of Selection Sort.

```
#include<iostream>
using namespace std;

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;
        }
    }
```

Q13

Ans Yes, we can modify the bubble sort algorithm to optimize it so that it does not scan the entire array once it is already sorted.

```
#include < iostream>
using namespace std;

void Bubble Sort (int arr[], int n ){

    bool swapped;
    for(int i=0; i<n; i++) {

        swapped = false
```

Date.
Page No.

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