Assisnment 2

1) Write linear search pseudocode to search an element in a sorted array with minimum comparisons

```
int linear_search(int a[], int n, int x)
{
    i=0;
    while(i<n && a[i]<x)
    {
        i++;
    }
    if(i<n && a[i]==x)
    {
        return i;
    }
    else
    {
        return -1;
    }
}
```

2) Write pseudo code for iterative and recusive insertion sort. Insertion sort is called online sort Why? What about other sorting algorithms that has been discussed in lectures?

→ iterative:

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

recursive :

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

=> Insertion sort is called online sort because it can sort a list of data elements as they are received or generated, one at a ~~time~~ time, without having to wait for the entire list to be available.

=> Other sorting algorithms like bubble sort and selection sort are simple but inefficient for large datasets, while merge sort and ~~selection sort~~ quick sort are more efficient but require additional memory space.

4) Divide all the sorting algorithms into inplace/stable/online sorting.

Inplace Sorting:-
- Bubble Sort
- Insertion sort
- selection sort
- Quick sort
- ~~Heap~~ Heap sort

Stable Sorting:-
- Insertion sort
- Merge sort
- Counting sort

Online Sorting:-
- Insertion sort
- Merge sort

5) Write recursive / iterative pseudo code for binary search. What is the time and space complexity of Linear and Binary Search (Recursive and iterative)

```
int binarysearch(int a[], int l, int h, int key)
{
    if (l>h)
        return -1;

    int mid =(l+h)/2;
    if (a[mid]==key)
        return mid;
    else if (key > a[mid])
        ~~return binarysearch(a, l, mid++, key);~~
        return binarysearch(a, mid+1, h, key);
    else
        return binarysearch(a, l, mid-1, key);
}
```

Time complexity = O(log n)

Space complexity = O(log n)

```
int binary search (int a[], int l, int h, int key)
{
    while (l<=h)
    {
        int mid = (l+h)/2;
        if (a[mid] == key)
            return mid;
        else if (key > a[mid])
            l = mid+1;
        else
            h = mid-1;
    }
    return -1;
}
```

Time complexity = O(logn)

Space complexity = O(1)

6) Write recurrence relation for binary recursive search.

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

7) Find two indexes such that a[i] + a[j] = k in minimum ~~m~~ time complexity?

```
for(i=0 ; i<n-2; i++)
{
    for(j=i+1; j<n-1; j++)
    {
        ~~for(k=j+1;~~
        if (a[i] + a[j] == k)
        {
            printf(" indox are : %d %d 9", i,j));
        }   return 0;
    }
  }
}
```

8) Which sorting is best for practical uses? Explain.

Quick sort is a widely used sorting algorithm due to its effeciency and versatility. It is suitable for large datasets and can sort them in place, meaning that it doesn't require additional memory to sort the data. Its worst case time complexity is $O(n^2)$ which can occur when input dat is already sorted.

10) In which cases Quick Sort will give the best and the worst case time complexity?

→ In the best case, the pivot element always divides the input array into two equal subarrays, and each recursive call of the algorithm is applied to a subarray of half the size. In this case, the time complexity of quick sort is $O(n \log n)$, which is same as average case.

→ In the worst case, the pivot element is always chosen as either the smallest or largest element in subarray, which result in one subarray of size $n-1$ and another subarray of size 1. In this case, ~~the~~ time complexity of quicksort is $O(n^2)$, which is significantly worse than average case. This worst case can occur when input in array is already sorted.

11) Write Recurrence Relation of Merge sort and Quick Sort in the best and worst case? What are similakities and differences between complexities of two algorithms and why?

⇒ Recurrence Relation :-

Merge sort:

Best case - $2T(n/2) + O(n)$

Worst case - $2T(n/2) + O(n)$

Quick sort:

Best case - $2T(n/2) + O(n)$

Worst case - $T(n-1) + O(n)$

Similarity:

Both merge sort and quick sort have a time complexity of $O(n \log n)$ on average.

Difference :-

In the worst case, quick sort has a time complexity of $O(n^2)$ while merge sort maintains its $O(n \log n)$ time complexity.

~~12~~)

12) Selection sort is not stable by default but can you write a version of stable selection sort.

```
void selection (int a[], int n)
{
    for (i=0 ; i<n-1 ; i++)
    {
        int min=i;
        for (int j=i+1 ; j<n ; j++)
        {
            if (a[j] < a[min])
                min =j;
        }
        int temp = a[i];
        a[i]= a [min];
        a[min]= temp;
    }
}
```

13) Bubble sort scans whole array even when array is sorted. Can you modify the bubble sort so that it doesn't scan whole array once it is sorted.

```
void bubble (int a[], int n)
{
    int i,j,flag;
    for (i=0; i<n-1; i++)
    {
        flag =0;
        for(j=0 ; j<n-i-1 ; j++)
        {
            int temp = a[j]
            a[j] = a[j+1];
            a[j+1]= temp;
            flag =1;
        if (flag == 0)
            break;
    }
  }
}
```

3) Complexity of all the sorting algorithm that has been discussed in lectures.

	Best case	Average case	Worst case	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	$o(n \log n)$	$o(n \log n)$	$O(n^2)$	$o(n)$
Heap sort	$o(n \log n)$	$o(n \log n)$	$o(n \log n)$	$o(1)$