Data Structures and Algorithms Assignment 2

1. Use examples to explain the sorting algorithms.

Selection Sort: -

```
public void selectionsort(int array[])
{
    int n = array.length;
    for (int i = 0; i < n-1; i++)
    {
        int index = i;
        int min = array[i];
        for (int j = i+1; j < n; j++)
        {
            if (array[j] < array[index])
            {
                index = j;
                min = array[j];
            }
        }
        int t = array[index];
        array[index] = array[i];
        array[i] = t;
    }
}
```

The Algorithms iterates over an array find the smaller elements and swaps it to the first position then find the second smallest element and swaps with them the second elements and follows this pattern throughout the iteration.

Selection sort has a Time Complexity of **O(n2)** and a space complexity of **O(n)**. It is an inplace sorting algorithm and it is not a stable algorithm.

Bubble Sort: -

```
public class BubbleSort {

 static void sort(int[] arr) {

   int n = arr.length;
   int temp = 0;

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

        for(int x=1; x < (n-i); x++){

            if(arr[x-1] > arr[x]){

                temp = arr[x-1];
                arr[x-1] = arr[x];
                arr[x] = temp;
```

```
                }

            }
        }

    }

    public static void main(String[] args) {

    for(int i=0; i < 15; i++){

            int arr[i] = (int)(Math.random() * 100 + 1);
    }

     System.out.println("array before sorting\n");

            for(int i=0; i < arr.length; i++){

                System.out.print(arr[i] + " ");

            }

            bubbleSort(arr);
            System.out.println("\n array after sorting\n");
            for(int i=0; i < arr.length; i++){
                System.out.print(arr[i] + " ");
            }

        }
    }
```

This Algorithm similar to the bubbles iterates over an array, compares two elements and swaps the smaller or larger value to one end. This process is repeated for n number of times to completely sort the algorithm. This is an inplace sorting algorithm with a time complexity of **O(n*n)**. This is a stable algorithm.

Insertion Sort: -

```
public int[] insertionSort(int[] arr)
    for (j = 1; j < arr.length; j++) {
      int key = arr[j]
      int i = j - 1
      while (i > 0 and arr[i] > key) {
        arr[i+1] = arr[i]
        i -= 1
      }
      arr[i+1] = key
    }
    return arr;
```

In Insertion sort, they key element is compared with the previous element. If the previous elements are greater than the key element, then you move the previous element to the next position.

Quick Sort: -

```
public class Quick{
int partition (int a[], int start, int end)
{
  int pivot = a[end];
  int i = (start - 1);

  for (int j = start; j <= end - 1; j++)  {
    if (a[j] < pivot){
      i++;
      int t = a[i];
      a[i] = a[j];
      a[j] = t;
    }
  }
  int t = a[i+1];
  a[i+1] = a[end];
  a[end] = t;
  return (i + 1);
}
{
  if (start < end){
    int p = partition(a, start, end);
    quick(a, start, p - 1);
    quick(a, p + 1, end);
  }
}
void printArr(int a[], int n)  {
  int i;
  for (i = 0; i < n; i++){
    System.out.print(a[i] + " ");
  }
}
```

```
    public static void main(String[] args){
    int a[] = { 13, 18, 27, 2, 19, 25 };
    int n = a.length;
    System.out.println("\nBefore sorting array elements are - ");
    Quick q = new Quick();
    q.printArr(a, n);
    q.quick(a, 0, n - 1);
    System.out.println("\nAfter sorting array elements are - ");
    q.printArr(a, n);
    System.out.println();
    }
  }
```

In Quick sort an element the last element or any randomized element is chosen as the the pivot. Then we run a partitioning algorithm that sort the array such that the elements on the right of the pivot are greater than the pivot element and the elements on the left are smaller than the pivot element. The quicksort algorithm is called recursively to sort the entire array.

Merge Sort: -

```
public class mergesort {

        public static int[] mergesort(int[] arr,int lo,int hi) {

                if(lo==hi) {
                        int[] ba=new int[1];
                        ba[0]=arr[lo];
                        return ba;
                }

                int mid=(lo+hi)/2;
                int arr1[]=mergesort(arr,lo,mid);
                int arr2[]=mergesort(arr,mid+1,hi);
                return merge(arr1,arr2);
        }

        public static int[] merge(int[] arr1,int[] arr2) {
                int i=0,j=0,k=0;
                int n=arr1.length;
                int m=arr2.length;
                int[] arr3=new int[m+n];
                while(i<n && j<m) {
                        if(arr1[i]<arr2[j]) {
                                arr3[k]=arr1[i];
                                i++;
                        }
```

```
                else {
                        arr3[k]=arr2[j];
                        j++;
                }
                k++;
        }

        while(i<n) {
                arr3[k]=arr1[i];
                i++;
                k++;
        }

        while(j<m) {
                arr3[k]=arr2[j];
                j++;
                k++;
        }

        return arr3;

    }

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        int arr[]= {2,9,8,3,6,4,10,7};
        int[] so=mergesort(arr,0,arr.length-1);
        for(int i=0;i<arr.length;i++)
            System.out.print(so[i]+" ");
    }
}
```

The Merge Sort Algorithm, a divide and Conquer algorithm it divides the array into two havles then calls itself for the two halves and then merges the two sorted halves. The sorted array is then merged to form the final sorted array.

The time complexity of the algorithm is **O(n*long(n).** The Space complexity is **O(n).** This is an example of an outplace sorting algorithm and this is a stable algorithm.

2. What Are the Benefits of Stacks?

- Stacks are a recursive data structure as it follows the LIFO principle. This makes them the best data structure to perform reverse action such as reverse a word. This results in efficient data management which is not easily possible with arrays or linked list.
- The compliers user stack to calculate the value of expressions by converting expressions to prefix and postfix form.
- Stacks are efficient in management of functions because when a function is called the local variables are stored in a stack and it is automatically destroyed once returned.
- They give us control over how memory is allocated or deallocated.
- Variable in stack cannot be resized and stacks is difficult to be corrupted hence making it more reliable and secure.

3. What is the difference between a stack and a queue?

The stack data structure is a linear data structure that follow in the **LIFO** principle that is the element that the element that was last inserted would be the first to be out and the element that was first stored will be the last to be retrieved. They only support two function that are to push the data and the pop out the data. Back button for navigations is an excellent example of stacks.

The queue data structures are also a linear data structure that follows the **FIFO** principle the element that will be first inserted will also be the one to be out first. The element is entering through the front of the queue and move to the rear end where they are executed. They two functions are Enque and Deque. Alert and notification systems tend to have a FIFO behaviour and queue are best suited for these scenarios.

4. What are the different forms of queues?

There are five different forms of queues, and they are:

Circular Queue: In a circular queue the last position in the queue is connected to the first element of the queue. They also strictly follow the FIFO principle. These queues are primarily used in Memory Management, Traffic Systems and CPU Scheduling.

Input restricted Queue: In an input restricted queue the input can only be taken from the rear side of the queue. But the deletion is permitted to be performed from both the ends. These Queues also strictly stick to the FIFO principle. In cases such as irrelevant data, performance issues etc where the recently inserted data need not be removed these queues find their application there.

Output restricted Queue: In these queues the input can be taken from both the sides, but the deletion can only be performed from the front side. These queues are used in some sort of priority list where it is necessary to insert the data in the first place.

Double ended Queue: These are queues in which data can be both inserted and deleted from both the front and rear side of the queue. They support both the clockwise and anticlockwise rotation.

Priority Queue: Priority Queue is a special type of queue in which elements is associated with a priority and is served according to its priority. There are two types of priority and they are :

Ascneding Priotiy Queue: In this queue elements are inserted by the elements removed will be the smallest.

Descending Priority Queue: In this queue the elements can be inserted arbitrarily but the elements being removed will the largest elements.

Application of these Priority Queues can be seen CPU scheduling Algorithms.

5.Why should I use Stack or Queue data structures instead of Arrays or Lists, and when should I use them?

Since both Stack and Queue works under the principles such as LIFO and FIFO they offer better data management than an Array or a List. Data cannot be randomly inserted in a stack or queue since the

adhere to a strict structure. In the case of list of array random access is allowed, which makes them less secure and more susceptible to corruption.
When there is a need to insert or delete data in a particular order then a queue should be preferred as it sticks to a FIFO principle. Stack and its LIFO principle are better used when you want to access data in a reverse order. Arrays and list should only be used is scenarios where data can be stored and retrieved anywhere and there is not much need for a stricter principle.

6. What is the significance of Stack being a recursive data structure?

Every recursive function will have two characteristics. That is the computation used to solve problem is identical for every position in the data set. The result of the initial computation will depend on the result of the subsequent computations. These raise to the need of stack a recursive data structure. A stack will allow store the current state while the program runs to calculate the downstream. And when we return from the downstream the store state is popped out and the computation is competed.
This efficiency and simplicity that the stack offers is what makes it an advantage data structure for recursive algorithms.