DS - Assignment - 1

① Analysis of all sorting techniques in one program which calculates the time taken by various sorting techniques.

A. Code

```
import java.util.Arrays;
import java.util.Random;

public class SortingAnalysis {

    // Bubble Sort
    public static void bubbleSort(int[] arr){
        int n = arr.length;
        for(int i=0; i<n-1; i++){
            for(int j=0; j<n-i-1; j++){
                if(arr[j] > arr[j+1]){
                    int temp = arr[j];
                    arr[j] = arr[j+1];
                    arr[j+1] = temp;
                }
            }
        }
    }
    // Selection Sort
    public static void selectionSort(int[] arr){
        int n = arr.length;
```

```
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 temp = arr[minIndex];
    arr [minIndex] = arr[i];
    arr [i] = temp;
}
// Insertion Sort
public static void insertionSort (int[] arr){
    int n = arr.length;
    for (int 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 = j-1;
        }
        arr [j+1] = key;
    }
}
```

```
// Mexge sort
public static void mexgesort(int[] axx, int l, int x){
    if (l < x) {
        int m = (l+x)/2;
        mexgesort(axx, l, m);
        mexgesort(axx, m+1, x);
        mexge(axx, l, m, x);
    }
}

public static void mexge(int[] axx, int l, int m, int x){
    int n1 = m - l + 1;
    int n2 = x - m;
    int[] L = new int[n1];
    int[] R = new int[n2];

    System.arraycopy(axx, l, L, 0, n1);
    System.arraycopy(axx, m+1, R, 0, n2);
    int k = l;
    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) {
            axx[k] = L[i];
            i++;
        }else{
            axx[k] = R[j];
            j++;
        }
        k++;
    }
```

```
While (i < n1) {
    arr[k] = L[i];
    i++;
    k++;
}
While (j < n2) {
    arr[k] = R[j];
    j++;
    k++;
}
}
// Quick Sort
public static void quickSort(int[] arr, int low, int high) {
    if (low < high) {
        int pi = partition(arr, low, high);
        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}
public static int partition(int[] arr, int low, int high) {
    int pivot = arr[high];
    int i = (low - 1);
    for (int j = low; j < high; j++) {
        if (arr[j] < pivot) {
            i++;
            int temp = arr[i];
            arr[i] = arr[j];
            arr[j] = temp;
```

```
    }
}
    int temp = arr[i+1];
    arr[i+1] = arr[high];
    arr[high] = temp;
    return i+1;
}
// Utility method to calculate the time taken by
    a sorting algorithm.
public static void calculationTime(int[] arr, String
    algorithmName) {
    int[] tempArr = Arrays.copyOf(arr, arr.length);
    long startTime = System.nanoTime();
    switch (algorithmName) {
        case "Bubble Sort":
            bubbleSort(tempArr);
            break;
        case "Selection Sort":
            selectionSort(tempArr);
            break;
        case "Insertion Sort":
            insertionSort(tempArr);
            break;
        case "Merge Sort":
            mergeSort(tempArr, 0, tempArr.length-1);
            break;
```

```
case "Quick Sort":
    quickSort(tempArr, 0, tempArr.length - 1);
    break;
default:
    System.out.println("Invalid Algorithm");
    return;
}
long endTime = System.nanoTime();
System.out.println(algorithmName + " took " + (endTime - startTime) + " nanoseconds.");
}

public static void main(String[] args) {
    Random rand = new Random();
    int n = 1000; // size of the array
    int[] arr = new int[n];

    for (int i = 0; i < n; i++) {
        arr[i] = rand.nextInt(1000);
    }

// Analyzing the time taken by different sorting algorithms.
calculateTime(arr, "Bubble Sort");
calculateTime(arr, "Selection Sort");
calculateTime(arr, "Insertion Sort");
calculateTime(arr, "Merge Sort");
calculateTime(arr, "Quick Sort");
```

}

}

Output:

10 20 30 Bubble sort took 6198600 nanoseconds.

10 15 20 30 Selection Sort took 2160900 nano seconds

10 15 30 Insertion sort took 1029100 nanoseconds

10 15 Merge Sort took 924900 nanoseconds

10 15 40 50 60 Quick sort took 788200 nanoseconds

(2) Program to perform various operations on an Array like insert(), remove(), insertAt(), removeAt(), print() [check all the operations, conditions and exceptions].

A: Code:

```
import java.util.Arrays;

public class ArrayOperations {
    private int[] arr;
    private int size;
    private int capacity;

    public ArrayOperations(int capacity) {
        this.capacity = capacity;
        this.arr = new int[capacity];
        this.size = 0;
    }
```

```
// Method to insert an element at the end of the
public void insert (int element) {
    if (size == capacity) {
        System.out.println ("Array is full. Cannot ins
                                element.");
        return;
    }
    arr [size] = element;
    size++;
}
public void remove () {
    if (size == 0) {
        System.out.println ("Array is empty. Cann
                            remove element.");
        return;
    }
    size--;
}
public void insertAt (int index, int element) {
    if (size == capacity) {
        System.out.println ("Array is full. Cannot
            insert element.");
        return;
    }
    if (index < 0 || index > size) {
```

```
System.out.println("Index out of bounds, Cannot
                    insert element ");
    return;
}
for (int i= size ; i > index ; i--) {
    arr[i] = arr[i-1];
}
arr[index] = element;
size++;
}
public void removeAt(int index) {
    if(size ==0) {
        System.out.println("Assay is empty Cannot
                            remove element ");
        return;
    }
    if (index < 0 || index >= size) {
        System.out.println("Index out of bounds
                            Cannot remove element");
        return;
    }
    for (int i = index ; i < size - 1 ; i++) {
        arr [i] = arr [i+1];
    }
    size--;
}
```

```
public void print() {
    if(size==0) {
        System.out.println("Array is empty.");
        return;
    }
    for(int i=0; i<size; i++) {
        System.out.print(arr[i]+" ");
    }
    System.out.println();
}
public static void main(String[] args) {
    ArrayOperations array = new ArrayOperations();
    array.insert(10);
    array.insert(20);
    array.insert(30);
    array.print();
    array.insertAt(1,15);
    array.print();
    array.removeAt(2);
    array.print();
    array.remove();
    array.print();
```

```
array.pinsert(40);
array.insert(50);
array.insert(60);
array.pxint();
}
```

Output:

10 20 30
10 15 20 30
10 15 30
10 15
10 15 40 50 60.

(3) Program to build a stack using Dynamic Array.
[check all the operations, conditions, and exceptions]

A:- Code:

```
class DynamicStack {
    private int[] stack;
    private int top;
    private int capacity;
public DynamicStack(int initialCapacity) {
    this.stack = new int[initialCapacity];
    this.capacity = initialCapacity;
    this.top = -1;
}
```

```
public void push(int data) {
    if (top == capacity - 1) {
        resizeStack();
    }
    stack[++top] = data;
}

public int pop() {
    if (isEmpty()) {
        throw new RuntimeException("Stack
        Underflow: Attempted to pop from an
        empty stack.");
    }
    return stack[top--];
}

public int peek() {
    if (isEmpty()) {
        throw new RuntimeException("Stack
            is empty: Cannot peek.");
    }
    return stack[top];
}

public boolean isEmpty() {
    return top == -1;
}
```

```
public int size() {
    return top + 1;
}
private void resizeStack() {
    int newCapacity = capacity * 2;
    int[] newStack = new int[newCapacity];
    System.arraycopy(stack, 0, newStack, 0, capacity);
    capacity = newCapacity;
    stack = newStack;
}
public void display() {
    if (isEmpty()) {
        System.out.println("Stack is empty.");
    } else {
        System.out.print("Stack elements: ");
        for (int i = 0; i <= top; i++) {
            System.out.print(stack[i] + " ");
        }
        System.out.println();
    }
}
public static void main(String[] args) {
    DynamicStack stack = new DynamicStack();
```

```
stack.Push(10);
stack.Push(20);
stack.display();

stack.push(30);
stack.display();

System.out.println("Top element is: "+stack
                    .peek());
System.out.println("Popped element is: "+stackpop
                    ());
stack.display();
System.out.println("Stack size: "+stack.size()
System.out.println("Popped element is: "+stack.pop
stack.display();
}
}
```

Output:

```
Stack elements: 10 20
stack elements: 10 20 30
Topelement is: 30

Popped element is: 30
stack elements: 10 20
Popped element is: 20
Stack elements: 10
Stack size: 1

popped element is: 10
stack is empty
```