# DAY 5

**Bubble Sort - Tryout**

```
class Tester {
	static int noOfSwaps = 0;
	static int noOfPasses = 0;

	public static void swap(int[] numbers, int firstIndex, int secondIndex) {
		int temp = numbers[firstIndex];
		numbers[firstIndex] = numbers[secondIndex];
		numbers[secondIndex] = temp;
		noOfSwaps += 1;
	}

	public static void bubbleSort(int[] numbers) {
		int length = numbers.length;

		for (int index1 = 0; index1 < (length - 1); index1++) {
			boolean swapped = false;
			noOfPasses += 1;
			for (int index2 = 0; index2 < (length - index1 - 1); index2++) {
				if (numbers[index2] > numbers[index2 + 1]) {
					swap(numbers, index2, index2 + 1);
					swapped = true;
				}
			}
			if (swapped == false)
				break;
		}
	}

	public static void main(String[] args) {

		int[] numbers = { 48, 40, 35, 49, 33 };
		System.out.println("Given array:");
		for (int number : numbers) {
			System.out.println(number);
		}

		bubbleSort(numbers);

		System.out.println("Sorted array:");
```

```
		for (int number : numbers) {
			System.out.println(number);
		}

		System.out.println("No. of passes: " + noOfPasses);
		System.out.println("No. of swaps: " + noOfSwaps);
	}
}
```

**Bubble Sort - Exercise 1**

```
class Tester {
	static int noOfSwaps = 0;
	static int noOfPasses = 0;

	public static void swap(int[] elements, int firstIndex, int secondIndex) {
		int temp = elements[firstIndex];
		elements[firstIndex] = elements[secondIndex];
		elements[secondIndex] = temp;
		noOfSwaps++;
	}

	public static int bubbleSort(int[] elements) {
		int n = elements.length;
		boolean swapped;

		for (int i = 0; i < n - 1; i++) {
			swapped = false;
			for (int j = 0; j < n - 1 - i; j++) {
				if (elements[j] > elements[j + 1]) {
					swap(elements, j, j + 1);
					swapped = true;
				}
			}
			noOfPasses++;
			System.out.println("Array after pass " + noOfPasses + ":");
			displayArray(elements);

			if (!swapped) {
				break;
			}
		}
		return noOfPasses;
	}

	public static void displayArray(int[] elements) {
		for (int element : elements)
```

```
                System.out.print(element + " ");
            System.out.println();
        }

        public static void main(String[] args) {

            int[] elements = { 23, 67, 45, 76, 34, 68, 90 };

            System.out.println("Given array:");
            displayArray(elements);

            int noOfPasses = bubbleSort(elements);

            System.out.println("==========================");
            System.out.println("Total number of passes needed to sort the array: " +
noOfPasses);
            System.out.println("==========================");

            System.out.println("Array after sorting:");
            displayArray(elements);
        }
}
```

**Merge Sort - Tryout**

```
class Tester {
        public static void main(String[] args) {
            int[] arr = { 19, 8, 16, 26, 45, 76 };
            mergeSort(arr, arr.length);
            for (int number : arr)
                System.out.println(number);
        }

        public static void mergeSort(int[] arr, int size) {
            if (size < 2)
                return;

            int mid = size / 2; //Dividing the array into two halves
            int[] left = new int[mid]; //Creating temporary array to the left of the mid value
            int[] right = new int[size - mid]; //Creating temporary array to the right of the mid
value

            //Copying data to temporary arrays
            for (int index = 0; index < mid; index++)
                left[index] = arr[index];

            for (int index = mid; index < size; index++)
```

```
                right[index - mid] = arr[index];

            //Invoking mergeSort() by passing left array
            mergeSort(left, mid);

            //Invoking mergeSort() by passing right array
            mergeSort(right, size - mid);

            //Invoking merge() by passing the arrays returned
            merge(arr, left, right, mid, size - mid);
        }

        public static void merge(int[] arr, int[] left, int[] right, int leftMerge, int rightMerge) {

            int firstIndex = 0; //initial index of first sub-array
            int secondIndex = 0; //initial index of second sub-array
            int thirdIndex = 0; //initial index of merged sub-array

            while (firstIndex < leftMerge && secondIndex < rightMerge) {
                if (left[firstIndex] <= right[secondIndex])
                    arr[thirdIndex++] = left[firstIndex++];
                else
                    arr[thirdIndex++] = right[secondIndex++];
            }

            while (firstIndex < leftMerge)
                arr[thirdIndex++] = left[firstIndex++];

            while (secondIndex < rightMerge)
                arr[thirdIndex++] = right[secondIndex++];
        }
}
```

**MERGE SORT EXERCISE-1**

```
class Tester {

   public static void mergeSort(int[] elements, int size) {
      if (size < 2) {
         return;
      }
      int mid = size / 2;
      int[] left = new int[mid];
      int[] right = new int[size - mid];

      for (int i = 0; i < mid; i++) {
```

```
            left[i] = elements[i];
        }
        for (int i = mid; i < size; i++) {
            right[i - mid] = elements[i];
        }
        mergeSort(left, mid);
        mergeSort(right, size - mid);

        merge(elements, left, right, mid, size - mid);
    }

    public static void merge(int[] elements, int[] left, int[] right, int leftMerge, int rightMerge) {
        int i = 0, j = 0, k = 0;
        while (i < leftMerge && j < rightMerge) {
            if (left[i] <= right[j]) {
                elements[k++] = left[i++];
            } else {
                elements[k++] = right[j++];
            }
        }
        while (i < leftMerge) {
            elements[k++] = left[i++];
        }
        while (j < rightMerge) {
            elements[k++] = right[j++];
        }
    }

    public static void displayArray(int[] elements) {
        for (int element : elements)
            System.out.print(element + " ");
        System.out.println();
    }

    public static void main(String[] args) {
        int[] elements = { 95, 56, 20, 98, 34, 77, 80 };

        System.out.println("Given Array:");
        displayArray(elements);

        mergeSort(elements, elements.length);

        System.out.println("Sorted Array:");
        displayArray(elements);
    }
}
```

**Linear Search - Assignment 1**

```
class Tester {

    public static int searchEmployeeId(int[] employeeIds, int employeeIdToBeSearched) {
        for (int i = 0; i < employeeIds.length; i++) {
            if (employeeIds[i] == employeeIdToBeSearched) {
                return i + 1; // returning the number of iterations (1-based index)
            }
        }
        return -1; // Employee ID not found
    }

    public static void main(String a[]) {
        int[] employeeIds = { 8011, 8012, 8015, 8016, 8020, 8022, 8025 };
        int employeeIdToBeSearched = 8022;

        int numberOfIterations = searchEmployeeId(employeeIds, employeeIdToBeSearched);

        if (numberOfIterations == -1)
            System.out.println("Employee Id " + employeeIdToBeSearched + " is not found!");
        else
            System.out.println("Employee Id " + employeeIdToBeSearched + " is found! Number of
iterations: " + numberOfIterations);
    }
}
```

**Binary Search - Assignment 1**

```
class Tester {

    public static int searchCustomerId(int[] customerIds, int customerIdToBeSearched) {
        int left = 0;
        int right = customerIds.length - 1;

        while (left <= right) {
            int mid = left + (right - left) / 2;

            if (customerIds[mid] == customerIdToBeSearched) {
                return mid; // Customer ID found at index mid
            }

            if (customerIds[mid] < customerIdToBeSearched) {
                left = mid + 1; // Search in the right half
            } else {
                right = mid - 1; // Search in the left half
            }
```

```
        }

        return -1; // Customer ID not found
    }

    public static void main(String[] args) {
        int[] customerIds = { 80451, 80462, 80465, 80479, 80550, 80561, 80665, 80770 };
        int customerIdToBeSearched = 80462;

        int index = searchCustomerId(customerIds, customerIdToBeSearched);

        if (index == -1)
            System.out.println("Customer Id " + customerIdToBeSearched + " is not found!");
        else
            System.out.println("Customer Id " + customerIdToBeSearched + " is found at index
position " + index + "!");
    }
}
```

**Merge Sort - Assignment 1**

```
import java.util.Arrays;
class Tester {

    public static void mergeSort(int[] elements, int size) {
        if (size < 2) {
            return; // Base case: If the array is of size 1 or less, it's already sorted
        }

        // Find the middle point to divide the array into two halves
        int mid = size / 2;

        // Create temporary arrays for left and right halves
        int[] left = new int[mid];
        int[] right = new int[size - mid];

        // Populate left and right arrays
        System.arraycopy(elements, 0, left, 0, mid);
        System.arraycopy(elements, mid, right, 0, size - mid);

        // Recursively sort the two halves
        mergeSort(left, mid);
        mergeSort(right, size - mid);
```

```
    // Merge the sorted halves
    merge(elements, left, right, mid, size - mid);
}

public static void merge(int[] elements, int[] left, int[] right, int leftMerge, int rightMerge) {
    int i = 0, j = 0, k = 0;

    // Compare elements from left and right arrays and merge them into elements array
    while (i < leftMerge && j < rightMerge) {
        if (left[i] <= right[j]) {
            elements[k++] = left[i++];
        } else {
            elements[k++] = right[j++];
        }
    }

    // Copy remaining elements from left array, if any
    while (i < leftMerge) {
        elements[k++] = left[i++];
    }

    // Copy remaining elements from right array, if any
    while (j < rightMerge) {
        elements[k++] = right[j++];
    }
}

// Function to find the median
public static double findMedian(int elements[]) {
    int n = elements.length;
    if (n % 2 == 1) {
        // Odd number of elements, median is the middle element
        return elements[n / 2];
    } else {
        // Even number of elements, median is the average of the middle two elements
        int mid1 = elements[n / 2 - 1];
        int mid2 = elements[n / 2];
        return (double)(mid1 + mid2) / 2;
    }
}

public static void main(String[] args) {
    int elements[] = { 64, 34, 25, 12, 22, 11, 90 };

    // Perform merge sort
    mergeSort(elements, elements.length);
```

```
        // Find and print the median
        System.out.println("Sorted Array: " + Arrays.toString(elements));
        System.out.println("Median: " + findMedian(elements));
    }
}
```

**Bubble Sort - Assignment 1**

```
public class Tester {
    static int noOfSwaps = 0;
    static int noOfPasses = 0;

    public static void swap(int[] elements, int firstIndex, int secondIndex) {
        int temp = elements[firstIndex];
        elements[firstIndex] = elements[secondIndex];
        elements[secondIndex] = temp;
        noOfSwaps++;
    }

    public static int bubbleSort(int[] elements) {
        int n = elements.length;
        boolean swapped;
        noOfPasses = 0;

        for (int i = 0; i < n - 1; i++) {
            swapped = false;
            for (int j = 0; j < n - 1 - i; j++) {
                if (elements[j] > elements[j + 1]) {
                    swap(elements, j, j + 1);
                    swapped = true;
                }
            }
            noOfPasses++;

            // If no elements were swapped, array is already sorted
            if (!swapped) {
                break;
            }
        }

        return noOfPasses;
    }

    public static void displayArray(int[] elements) {
        for (int element : elements)
            System.out.print(element + " ");
```

```
        System.out.println();
    }

    public static void main(String[] args) {
        int[] elements = { 23, 67, 45, 76, 34, 68, 90 };

        System.out.println("Given array:");
        displayArray(elements);

        int noOfPasses = bubbleSort(elements);

        System.out.println("=========================");
        System.out.println("Total number of passes needed to sort the array: " + noOfPasses);
        System.out.println("Total number of swaps: " + noOfSwaps);
        System.out.println("=========================");

        System.out.println("Array after sorting:");
        displayArray(elements);
    }
}
```

**Brute Force - Exercise 1**

```
class Tester {

    public static int calculatePower(int num, int p) {
        int result = 1;
        for (int i = 0; i < p; i++) {
            result *= num;
        }
        return result;
    }

    public static void main(String[] args) {
        System.out.println(calculatePower(2, 3)); // Expected output: 8
        System.out.println(calculatePower(5, 4)); // Expected output: 625
        System.out.println(calculatePower(3, 0)); // Expected output: 1
        System.out.println(calculatePower(7, 2)); // Expected output: 49
    }
}
```

**Divide and Conquer - Exercise 1**

```
class Tester {
```

```
    public static int[] getMaxMin(int arr[], int low, int high) {
        int[] maxMin = new int[2];

        // If there is only one element
        if (low == high) {
            maxMin[0] = arr[low];
            maxMin[1] = arr[low];
            return maxMin;
        }

        // If there are two elements
        if (high == low + 1) {
            if (arr[low] > arr[high]) {
                maxMin[0] = arr[low];
                maxMin[1] = arr[high];
            } else {
                maxMin[0] = arr[high];
                maxMin[1] = arr[low];
            }
            return maxMin;
        }

        // If there are more than two elements
        int mid = (low + high) / 2;
        int[] leftMaxMin = getMaxMin(arr, low, mid);
        int[] rightMaxMin = getMaxMin(arr, mid + 1, high);

        // Compare results of two halves
        maxMin[0] = Math.max(leftMaxMin[0], rightMaxMin[0]);
        maxMin[1] = Math.min(leftMaxMin[1], rightMaxMin[1]);

        return maxMin;
    }

    public static void main(String args[]) {
        int arr[] = {1000, 10, 5, 1, 2000};

        int[] maxMin = getMaxMin(arr, 0, arr.length - 1);

        System.out.println("Minimum value is " + maxMin[1]);
        System.out.println("Maximum value is " + maxMin[0]);
    }
}
```

**Greedy Approach - Exercise 1**

```
class Tester {
```

```
    public static int findMaxActivities(int start[], int finish[]) {
        int n = start.length;
        int count = 1; // The first activity is always selected

        // The index of the last selected activity
        int lastSelectedActivity = 0;

        for (int i = 1; i < n; i++) {
            // If this activity has start time greater than or equal to the finish
            // time of the last selected activity, then select it
            if (start[i] >= finish[lastSelectedActivity]) {
                count++;
                lastSelectedActivity = i;
            }
        }

        return count;
    }

    public static void main(String[] args) {
        int start[] = {1, 3, 0, 5, 8, 5};
        int finish[] = {2, 4, 6, 7, 9, 9};

        System.out.println("Maximum number of activities: " + findMaxActivities(start, finish));
    }
}
```

**Dynamic Programming - Tryout 1**

```
class Tester {

    public static int fibonacci(int num) {
        //If passed input is 0, return 0
                    if (num == 0)
                        return 0;
                    //If passed input is 1, return 1
                    else if(num == 1)
                            return 1;
                    else
                            return fibonacci(num - 1) + fibonacci(num - 2);
            }

            public static void main(String[] args) {
                int num = 5;
                    System.out.println(num+"th fibaonacci number: "+fibonacci(num));
    }
```

```
}
```

**Dynamic Programming - Tryout 2**

```
class Tester {

   public static int fibonacci(int num) {
      //Declare an array to store Fibonacci numbers
      int f[] = new int[num + 1];
      int index;

      //0th and 1st number of the series are 0 and 1
      f[0] = 0;

      if (num > 0) {
         f[1] = 1;

         for (index = 2; index <= num; index++) {
            //Add the previous 2 numbers in the series and store the sum
            f[index] = f[index - 1] + f[index - 2];
         }
      }

      return f[num];
   }

   public static void main(String[] args) {
      int num = 9;
                  System.out.println(num+"th fibonacci number : "+fibonacci(num));
   }
}
```

**Dynamic Programming - Exercise 1**

```
class Tester {

         public static int cutRod(int[] price, int n) {
                  int[] dp = new int[n + 1]; // dp[i] will store the maximum price obtainable for a
rod of length i
                  dp[0] = 0; // A rod of length 0 has a maximum price of 0

                  // Build the dp array in bottom-up manner
                  for (int i = 1; i <= n; i++) {
                           int maxPrice = Integer.MIN_VALUE;
                           for (int j = 0; j < i; j++) {
                                    maxPrice = Math.max(maxPrice, price[j] + dp[i - j - 1]);
```

```
                }
                dp[i] = maxPrice;
            }

            return dp[n];
        }

        public static void main(String[] args) {
            int price[] = { 1, 5, 8, 9, 10, 17, 17, 20 };
            int n = 4;
            System.out.println("Maximum price: " + cutRod(price, n));
        }
}
```

**Brute Force - Assignment 1**

```
class Tester {

        public static int[][] multiply(int arr1[][], int arr2[][]) {
            int n = arr1.length;
            int[][] arr3 = new int[n][n];

            // Perform matrix multiplication
            for (int i = 0; i < n; i++) {
                for (int j = 0; j < n; j++) {
                    for (int k = 0; k < n; k++) {
                        arr3[i][j] += arr1[i][k] * arr2[k][j];
                    }
                }
            }

            return arr3;
        }

        public static void main(String[] args) {
            int arr1[][] = new int[][] { { 2, 4 }, { 1, 4 } };
            int arr2[][] = new int[][] { { 1, 4 }, { 1, 3 } };

            int[][] arr3 = multiply(arr1, arr2);

            // Displaying the result matrix arr3
            for (int i = 0; i < arr3.length; i++) {
                for (int j = 0; j < arr3.length; j++) {
                    System.out.print(arr3[i][j] + " ");
                }
                System.out.println();
            }
```

```
        }
}
```

**Divide and Conquer - Assignment 1**

```
class Tester {

    public static int findMaxSum(int arr[], int low, int high) {
        if (low == high) {
            return arr[low]; // Base case: Only one element
        }

        int mid = (low + high) / 2;

        // Recursively find the maximum sum in left and right subarrays
        int leftMax = findMaxSum(arr, low, mid);
        int rightMax = findMaxSum(arr, mid + 1, high);
        int crossMax = findMaxCrossingSubarraySum(arr, low, mid, high);

        // Return the maximum of the three sums
        return Math.max(Math.max(leftMax, rightMax), crossMax);
    }

    public static int findMaxCrossingSubarraySum(int arr[], int low, int mid, int high) {
        // Calculate maximum sum for the left part including middle element
        int leftSum = Integer.MIN_VALUE;
        int sum = 0;
        for (int i = mid; i >= low; i--) {
            sum += arr[i];
            if (sum > leftSum) {
                leftSum = sum;
            }
        }

        // Calculate maximum sum for the right part including middle element
        int rightSum = Integer.MIN_VALUE;
        sum = 0;
        for (int i = mid + 1; i <= high; i++) {
            sum += arr[i];
            if (sum > rightSum) {
                rightSum = sum;
            }
        }

        // Return the sum of the maximum sums of left and right parts
        return leftSum + rightSum;
    }
```

```
    public static void main(String[] args) {
        int arr[] = { -2, -5, 6, -2, -3, 1, 5, -6 };
        System.out.println("Maximum sum: " + findMaxSum(arr, 0, arr.length - 1));
    }
}
```

**Greedy Approach - Assignment 1**

```
class Tester {

    public static int findSwapCount(String inputString) {
        int unbalancedCount = 0;
        int swapsNeeded = 0;

        for (int i = 0; i < inputString.length(); i++) {
            char current = inputString.charAt(i);

            if (current == ')') {
                unbalancedCount++;
            } else if (current == '(') {
                if (unbalancedCount > 0) {
                    unbalancedCount--;
                    swapsNeeded++; // Increment swaps for each necessary correction
                }
            }
        }

        return swapsNeeded;
    }

    public static void main(String args[]) {
        String inputString = "())()(";
        System.out.println("Number of swaps: " + findSwapCount(inputString));
    }

}
```

**Dynamic Programming - Assignment 1**

```
import java.util.ArrayList;
import java.util.HashSet;
import java.util.List;
import java.util.Set;

class Tester {
```

```
    static int count = 0;
    static Set<String> memo = new HashSet<>();

    public static void findWordSegments(List<String> wordsList, String inputString) {
        count = 0;
        memo.clear();
        findSegmentsHelper(wordsList, inputString);
    }

    private static void findSegmentsHelper(List<String> wordsList, String input) {
        if (input.isEmpty()) {
            count++;
            return;
        }

        if (memo.contains(input)) {
            return;
        }

        for (String word : wordsList) {
            if (input.startsWith(word)) {
                findSegmentsHelper(wordsList, input.substring(word.length()));
            }
        }

        memo.add(input);
    }

    public static void main(String[] args) {
        List<String> wordsList = new ArrayList<>();
        wordsList.add("i");
        wordsList.add("like");
        wordsList.add("pizza");
        wordsList.add("li");
        wordsList.add("ke");
        wordsList.add("pi");
        wordsList.add("zza");

        String inputString = "ilikepizza";
        findWordSegments(wordsList, inputString);
        System.out.println("Number of segments: " + count);
    }
}
```