DAA

Assignment No 1

```
class MergeSort {
    // Recursive Merge Sort function
    static void mergeSort(int[] arr, int left, int right) {
        // Base condition: Only proceed if left index is smaller than right
        if (left < right) {
            // Find the middle point
            int mid = (left + right) / 2;

            // Recursively sort the left half
            mergeSort(arr, left, mid);

            // Recursively sort the right half
            mergeSort(arr, mid + 1, right);

            // Merge the two sorted halves
            merge(arr, left, mid, right);
        }
    }
    // Function to merge two sorted halves of the array
    static void merge(int[] arr, int left, int mid, int right) {
        // Sizes of two temporary subarrays
        int n1 = mid - left + 1;   // left half size
        int n2 = right - mid;      // right half size

        // Temporary arrays to hold data
```

```
int[] leftArray = new int[n1];
int[] rightArray = new int[n2];

// Copy data to temporary arrays
for (int i = 0; i < n1; i++)
    leftArray[i] = arr[left + i];     // copy left half
for (int j = 0; j < n2; j++)
    rightArray[j] = arr[mid + 1 + j]; // copy right half
// Initial indexes for left, right, and merged array
int i = 0, j = 0;
int k = left; // index for original array

// Merge the temp arrays back into arr[left..right]
while (i < n1 && j < n2) {
    if (leftArray[i] < rightArray[j]) {
        arr[k] = leftArray[i];
        i++;
    } else {
        arr[k] = rightArray[j];
        j++;
    }
    k++;
}
// Copy remaining elements of leftArray[], if any
while (i < n1) {
    arr[k] = leftArray[i];
    i++;
```

```
            k++;
        }
        // Copy remaining elements of rightArray[], if any
        while (j < n2) {
            arr[k] = rightArray[j];
            j++;
            k++;
        }
    }
    // Utility function to print array
    public static void printArray(int[] arr) {
        for (int i : arr)
            System.out.print(i + " ");
        System.out.println();
    }
    // Driver code
    public static void main(String[] args) {
        int[] arr = {10, 54, 7, 56, 23, 8};

        // Call merge sort on full array
        mergeSort(arr, 0, arr.length - 1);
        System.out.println("Sorted Array:");
        printArray(arr);
    }
}
```

Assignment No 2

```
class QuickSort {
    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);
        }
    }

    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;
    }
```

```
    static void printArray(int[] arr) {
        for (int n : arr)
            System.out.print(n + " ");
        System.out.println();
    }
    public static void main(String[] args) {
        int[] arr = {10, 54, 7, 56, 23, 8};
        quickSort(arr, 0, arr.length - 1);
        System.out.println("Sorted Array:");
        printArray(arr);
    }
}
```

Assignment No 3

```
import java.util.Arrays;
public class FractionalKnapsack {
    public static void main(String[] args) {
        int[] val = {60, 100, 120};
        int[] wt  = {10, 20, 30};
        int W = 50;

        double[][] ratio = new double[val.length][2];
        for (int i = 0; i < val.length; i++) {
            ratio[i][0] = i;
            ratio[i][1] = (double) val[i] / wt[i];
        }
        Arrays.sort(ratio, (a, b) -> Double.compare(b[1], a[1])); // sort descending
```

```
        double finalValue = 0;
        for (double[] r : ratio) {
            int idx = (int) r[0];
            if (W >= wt[idx]) {
                finalValue += val[idx];
                W -= wt[idx];
            } else {
                finalValue += r[1] * W;
                break;
            }
        }
        System.out.println("Maximum value: " + finalValue);
    }
}
```