**Year**

# Student Name: Shubhankar Juyal ID: 1000017858 Major: BTIT

**Graded Lab - 1**

**Question 1:**

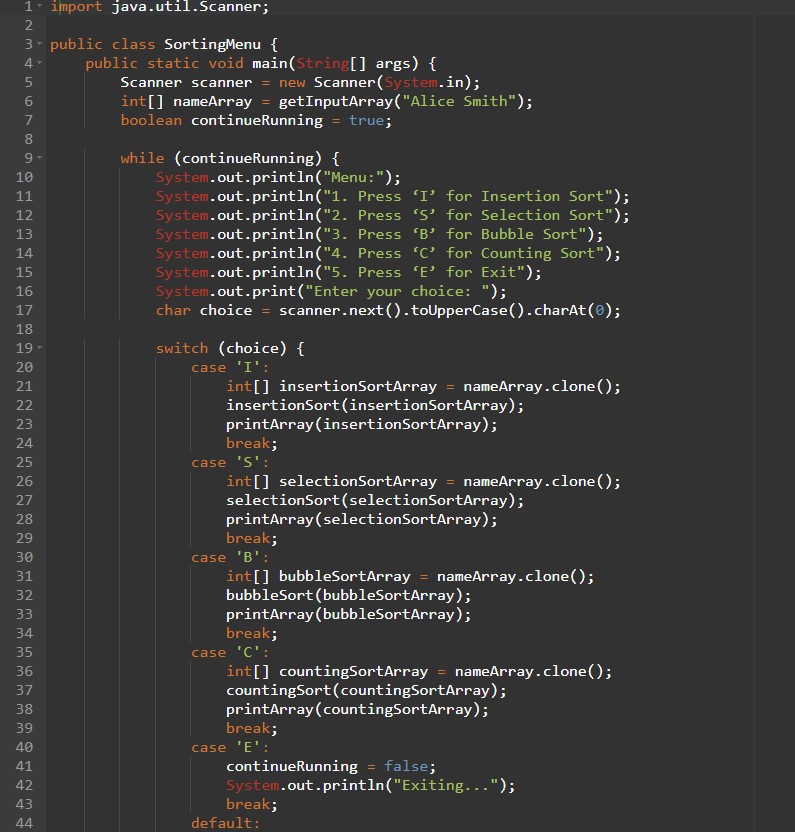
**Write a menu-driven Java program that implements Insertion Sort, Selection Sort, Bubble Sort, and Counting Sort as separate functions. Pass the input array as numbers corresponding to the letters of your full name, where each letter corresponds to its position in the alphabet (A=1, B=2, ..., Z=26). Ignore spaces and treat the input as caseinsensitive. For example, if the name is "Alice Smith" the array would be [1, 12, 9, 3, 5, 19, 13, 9, 20, 8].**

**Menu must be displayed as follow**

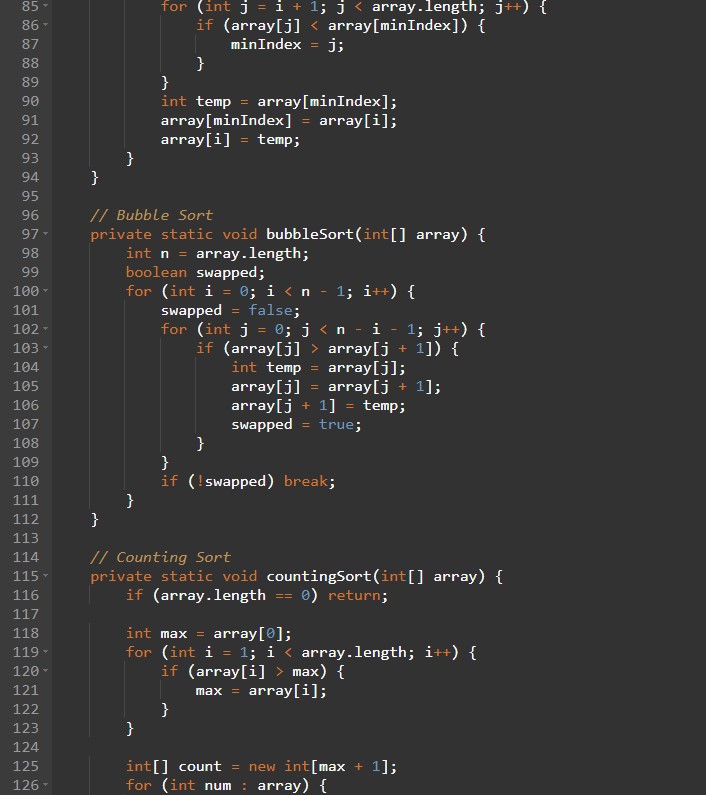
1. **Press ‘I’ for Insertion Sort**
2. **Press ‘S’ for Selection Sort**
3. **Press ‘B’ for Bubble Sort**
4. **Press ‘C’ for Counting Sort**
5. **Press ‘E’ for Exit**

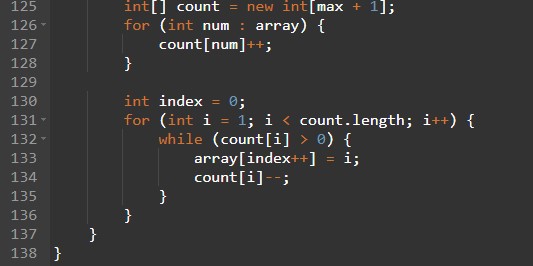
**The program must continuously run until "Exit" is selected.**

**CODE:-**

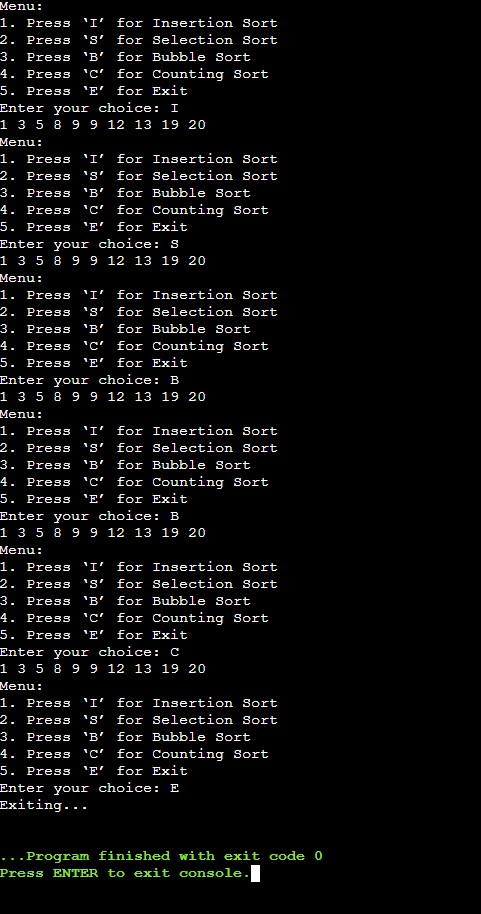








OUTPUT:-



**Non-graded – 3**

**Question 1:**

Write a Program in Java to implement Quick Sort.

Code :

import java.io.\*;

class Hello {

static void swap(int[] arr, int i, int j)

{

int temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}

static int partition(int[] arr, int low, int high)

{

int pivot = arr[high];

int i = (low - 1);

for (int j = low; j <= high - 1; j++) {

if (arr[j] < pivot) {

i++;

swap(arr, i, j);

}

}

swap(arr, i + 1, high);

return (i + 1);

}

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 void printArr(int[] arr)

{

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

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

}

}

public static void main(String[] args)

{

int[] arr = { 10, 7, 8, 9, 1, 5 };

int N = arr.length;

System.out.println("before sorting:");

printArr(arr);

quickSort(arr, 0, N - 1);

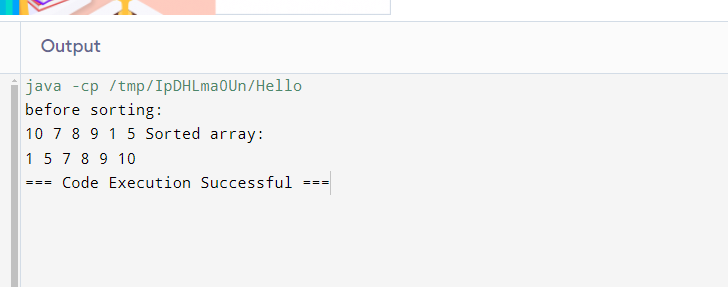
System.out.println("Sorted array:");

printArr(arr);

}

}

Output:



**Question 2:**

Write a Program in Java to implement Merge Sort.

Code:

import java.io.\*;

class hello {

static void merge(int arr[], int l, int m, int r)

{

int n1 = m - l + 1;

int n2 = r - m;

int L[] = new int[n1];

int R[] = new int[n2];

for (int i = 0; i < n1; ++i)

L[i] = arr[l + i];

for (int j = 0; j < n2; ++j)

R[j] = arr[m + 1 + j];

int i = 0, j = 0;

int k = l;

while (i < n1 && j < n2) {

if (L[i] <= R[j]) {

arr[k] = L[i];

i++;

}

else {

arr[k] = R[j];

j++;

}

k++;

}

while (i < n1) {

arr[k] = L[i];

i++;

k++;

}

while (j < n2) {

arr[k] = R[j];

j++;

k++;

}

}

static void sort(int arr[], int l, int r)

{

if (l < r) {

int m = l + (r - l) / 2;

sort(arr, l, m);

sort(arr, m + 1, r);

merge(arr, l, m, r);

}

}

static void printArray(int arr[])

{

int n = arr.length;

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

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

System.out.println();

}

public static void main(String args[])

{

int arr[] = { 12, 11, 13, 5, 6, 7 };

System.out.println("Given array is");

printArray(arr);

sort(arr, 0, arr.length - 1);

System.out.println("\nSorted array is");

printArray(arr);

}

}

Output:



**Non-graded – 4**

**Question 1:**

Write a Program in Java to implement Binary Searching.

Code :

import java.util.Scanner;

public class BinarySearch {

public static int binarySearch(int[] arr, int target) {

int left = 0;

int right = arr.length - 1;

while (left <= right) {

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

if (arr[mid] == target) {

return mid;

}

if (arr[mid] < target) {

left = mid + 1;

}

else {

right = mid - 1;

}

}

return -1;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the number of elements in the array: ");

int n = scanner.nextInt();

int[] arr = new int[n];

System.out.println("Enter " + n + " sorted elements:");

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

arr[i] = scanner.nextInt();

}

System.out.print("Enter the target element to search for: ");

int target = scanner.nextInt();

int result = binarySearch(arr, target);

if (result != -1) {

System.out.println("Element found at index: " + result);

} else {

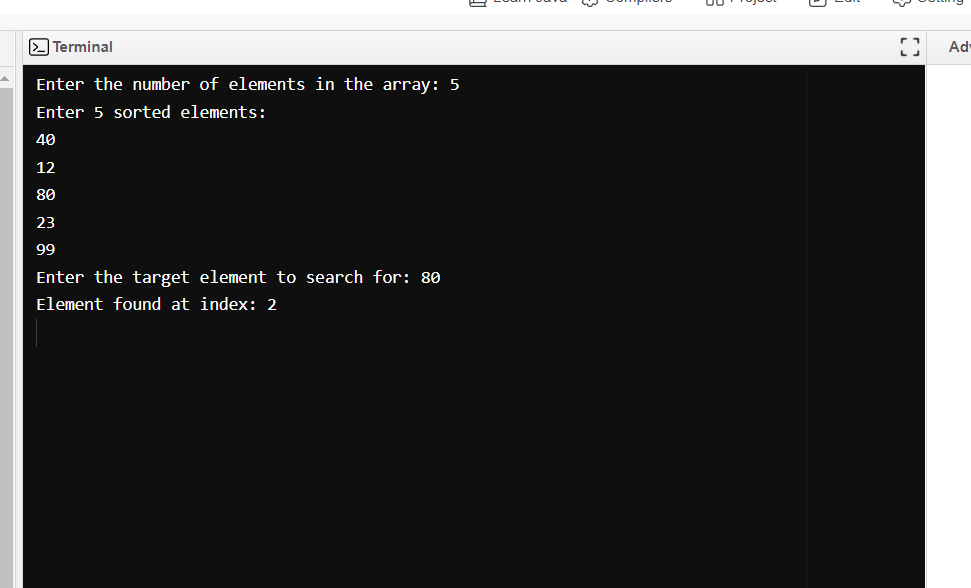
System.out.println("Element not found.");

}

}

}

Output :



**Question 2:**

Write a Program in Java to implement Heap Sort.

Code :

import java.util.Scanner;

public class HeapSort {

public static void heapify(int arr[], int heapSize, int rootIndex) {

int largest = rootIndex;

int leftChild = 2 \* rootIndex + 1;

int rightChild = 2 \* rootIndex + 2;

if (leftChild < heapSize && arr[leftChild] > arr[largest]) {

largest = leftChild;

}

if (rightChild < heapSize && arr[rightChild] > arr[largest]) {

largest = rightChild;

}

if (largest != rootIndex) {

int swap = arr[rootIndex];

arr[rootIndex] = arr[largest];

arr[largest] = swap;

heapify(arr, heapSize, largest);

}

}

public void printArray(int arr[]) {

for (int value : arr) {

System.out.print(value + " ");

}

System.out.println();

}

public static void main(String args[]) {

Scanner sc = new Scanner(System.in);

System.out.print("Enter the number of elements in the array: ");

int n = sc.nextInt();

int arr[] = new int[n];

System.out.println("Enter the elements of the array: ");

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

arr[i] = sc.nextInt();

}

HeapSort hs = new HeapSort();

System.out.println("Original array:");

hs.printArray(arr);

for (int i = n / 2 - 1; i >= 0; i--) {

heapify(arr, n, i);

}

for (int i = n - 1; i > 0; i--) {

int temp = arr[0];

arr[0] = arr[i];

arr[i] = temp;

heapify(arr, i, 0);

}

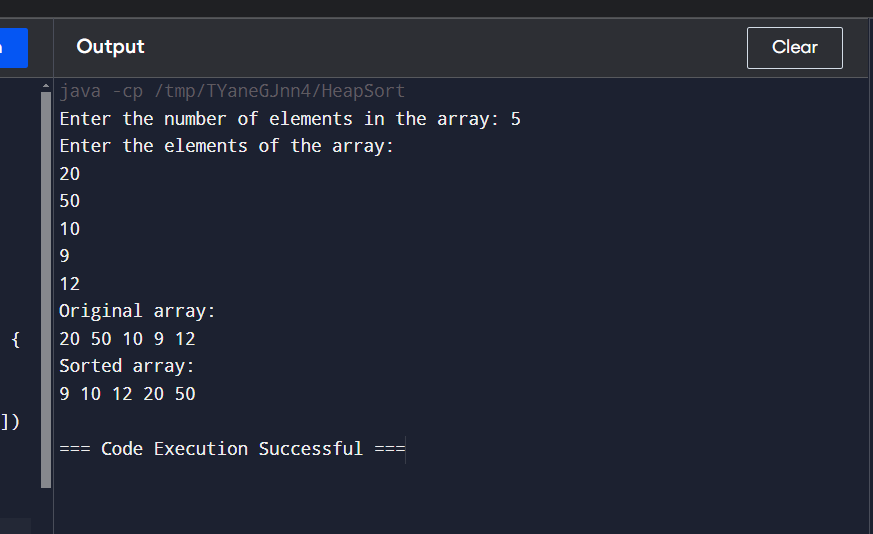
System.out.println("Sorted array:");

hs.printArray(arr);

}

}

Output :



**Graded – 2**

**Question 1:**

**Write a Java program to implement the Heap Sort algorithm. The program should perform the following tasks:**

1. **Display the input array before and after sorting.**
2. **Count and display the total number of comparisons and swaps inside the heapify algorithm.**
3. **After each heapify call, display the current state of the heap (array) and the root element.**
4. **Display the total number of times the heapify method is called.**

**Code :**

**import java.util.Scanner;**

**public class HeapSort {**

**private static int comparisonCount = 0;**

**private static int swapCount = 0;**

**private static int heapifyCount = 0;**

**public static void heapSort(int[] array) {**

**int n = array.length;**

**for (int i = n / 2 - 1; i >= 0; i--) {**

**heapify(array, n, i);**

**}**

**for (int i = n - 1; i > 0; i--) {**

**swap(array, 0, i);**

**heapify(array, i, 0);**

**}**

**}**

**public static void heapify(int[] array, int n, int i) {**

**heapifyCount++;**

**int largest = i;**

**int left = 2 \* i + 1;**

**int right = 2 \* i + 2;**

**if (left < n) {**

**comparisonCount++;**

**if (array[left] > array[largest]) {**

**largest = left;**

**}**

**}**

**if (right < n) {**

**comparisonCount++;**

**if (array[right] > array[largest]) {**

**largest = right;**

**}**

**}**

**if (largest != i) {**

**swap(array, i, largest);**

**System.out.println("Heap after swap:");**

**for (int j : array) {**

**System.out.print(j + " ");**

**}**

**System.out.println();**

**System.out.println("Root element: " + array[0]);**

**heapify(array, n, largest);**

**}**

**}**

**public static void swap(int[] array, int i, int j) {**

**int temp = array[i];**

**array[i] = array[j];**

**array[j] = temp;**

**swapCount++;**

**}**

**public static void main(String[] args) {**

**Scanner scanner = new Scanner(System.in);**

**System.out.print("Enter the number of elements in the array: ");**

**int n = scanner.nextInt();**

**int[] array = new int[n];**

**System.out.println("Enter the elements of the array:");**

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

**array[i] = scanner.nextInt();**

**}**

**System.out.println("Original array:");**

**for (int j : array) {**

**System.out.print(j + " ");**

**}**

**System.out.println();**

**heapSort(array);**

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

**for (int j : array) {**

**System.out.print(j + " ");**

**}**

**System.out.println();**

**System.out.println("Total comparisons in heapify: " + comparisonCount);**

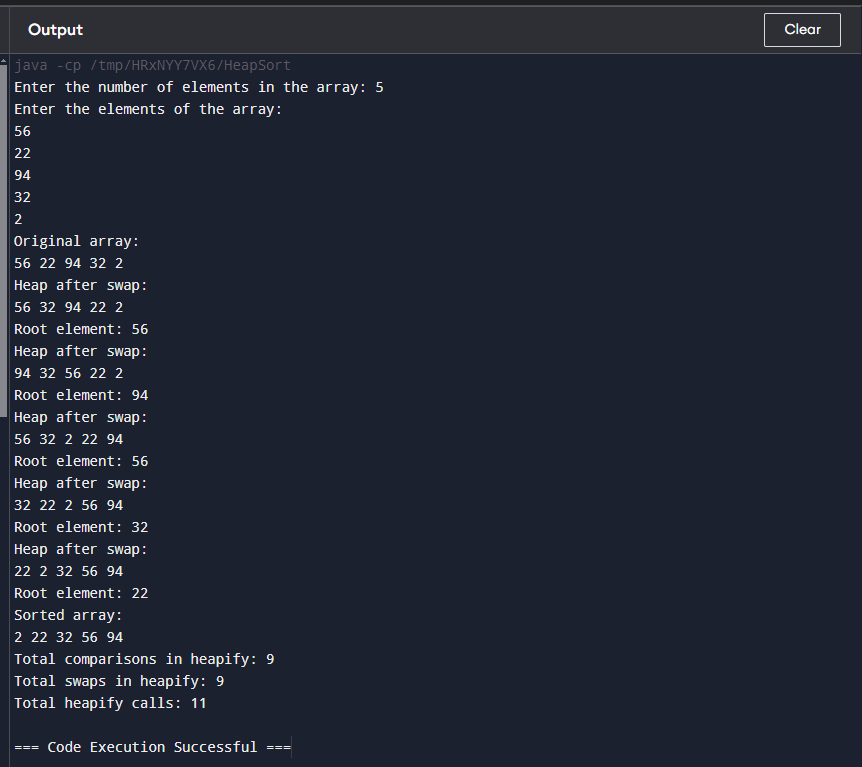
**System.out.println("Total swaps in heapify: " + swapCount);**

**System.out.println("Total heapify calls: " + heapifyCount);**

**}**

**}**

**Output :**

****

**Question 2:**

**Write a program in Java to Implement Merge Sort. In merge algorithm perform comparison between subarrays by taking last element of each subarray as INFINITY. Display the input array before and after sorting. Also display the total number of times the merge algorithm is called.**

**Code :**

**import java.util.Scanner;**

**public class MergeSort {**

**private static int mergeCount = 0;**

**public static void mergeSort(int[] array, int left, int right) {**

**if (left < right) {**

**int mid = (left + right) / 2;**

**mergeSort(array, left, mid);**

**mergeSort(array, mid + 1, right);**

**merge(array, left, mid, right);**

**}**

**}**

**public static void merge(int[] array, int left, int mid, int right) {**

**mergeCount++;**

**int n1 = mid - left + 1;**

**int n2 = right - mid;**

**int[] leftArray = new int[n1 + 1];**

**int[] rightArray = new int[n2 + 1];**

**for (int i = 0; i < n1; i++) {**

**leftArray[i] = array[left + i];**

**}**

**for (int j = 0; j < n2; j++) {**

**rightArray[j] = array[mid + 1 + j];**

**}**

**leftArray[n1] = Integer.MAX\_VALUE;**

**rightArray[n2] = Integer.MAX\_VALUE;**

**int i = 0, j = 0;**

**for (int k = left; k <= right; k++) {**

**if (leftArray[i] <= rightArray[j]) {**

**array[k] = leftArray[i];**

**i++;**

**} else {**

**array[k] = rightArray[j];**

**j++;**

**}**

**}**

**}**

**public static void main(String[] args) {**

**Scanner scanner = new Scanner(System.in);**

**System.out.print("Enter the number of elements in the array: ");**

**int size = scanner.nextInt();**

**int[] array = new int[size];**

**System.out.println("Enter the elements of the array:");**

**for (int i = 0; i < size; i++) {**

**array[i] = scanner.nextInt();**

**}**

**System.out.println("Input Array:");**

**for (int num : array) {**

**System.out.print(num + " ");**

**}**

**System.out.println();**

**mergeSort(array, 0, array.length - 1);**

**System.out.println("Sorted Array:");**

**for (int num : array) {**

**System.out.print(num + " ");**

**}**

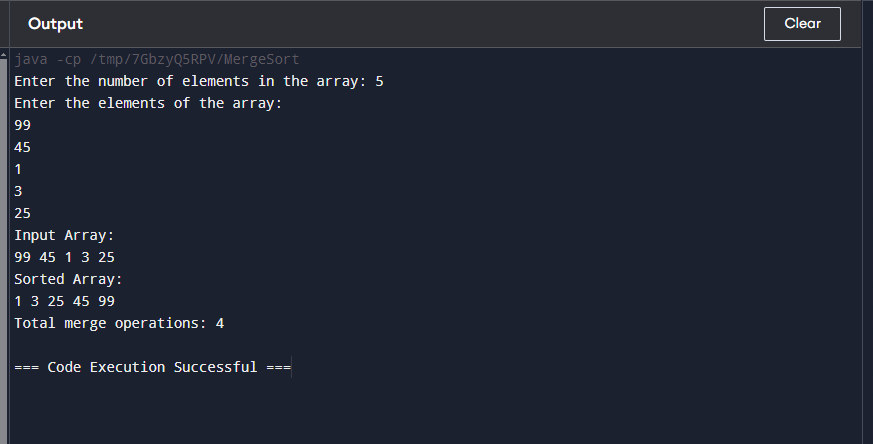
**System.out.println();**

**System.out.println("Total merge operations: " + mergeCount);**

**}**

**}**

**Output :**

****

**Non-graded – 5**

**Question 1:**

Write a program in Java to implement Activity Selection Problem.

Code :

import java.util.Collections;

import java.util.Comparator;

import java.util.Scanner;

import java.util.Vector;

class pair {

int st;

int et;

pair(int start, int end) {

this.st = start;

this.et = end;

}

}

public class A {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.println("enter the size of the array: ");

int n = sc.nextInt();

System.out.println("for size of the array : ");

Vector<pair> v = new Vector<>();

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

System.out.println("enter start time: ");

int s = sc.nextInt();

System.out.println("enter end time: ");

int e = sc.nextInt();

v.add(new pair(s, e));

}

Collections.sort(v, Comparator.comparingInt(a -> a.et));

int count = 0;

int tempe = 0;

for (pair i : v) {

if (i.st >= tempe) {

count++;

tempe = i.et;

}

}

System.out.println("Number of activities selected = " + count);

}

}

Output:



**Graded – 3**

**Question 1:**

**Write a Java program to solve the Activity Selection Problem using Heap Sort. Given below are a set of activities, each with a start time and a finish time.**

**start[] = {1, 3, 0, 5, 8, 5}**

**finish[] = {2, 4, 6, 7, 9, 9}**

**Code :**

**public class ActivitySelection {**

**static class pair {**

**int start;**

**int finish;**

**pair(int start, int finish) {**

**this.start = start;**

**this.finish = finish;**

**}**

**}**

**public static void heapSort(pair[] a) {**

**int n = a.length;**

**for (int i=n/2-1;i>=0;i--) {**

**heapify(a, n, i);**

**}**

**for (int i=n-1;i>0;i--) {**

**pair temp = a[i];**

**a[i] = a[0];**

**a[0] = temp;**

**heapify(a, i, 0);**

**}**

**}**

**static void heapify(pair[] a, int n, int i) {**

**int largest = i;**

**int left = 2 \* i + 1;**

**int right = 2 \* i + 2;**

**if (left < n && a[left].finish > a[largest].finish) {**

**largest = left;**

**}**

**if (right < n && a[right].finish > a[largest].finish) {**

**largest = right;**

**}**

**if (largest != i) {**

**pair swap = a[i];**

**a[i] = a[largest];**

**a[largest] = swap;**

**heapify(a, n, largest);**

**}**

**}**

**public static int helper(pair[] a) {**

**heapSort(a);**

**int count = 1;**

**int n = a.length;**

**int l = a[0].finish;**

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

**if (a[i].start >= l) {**

**count++;**

**l = a[i].finish;**

**}**

**}**

**return count;**

**}**

**public static void main(String[] args) {**

**pair[] a = {new pair(1, 2),new pair(3, 4),new pair(0, 6),new pair(5, 7),new pair(8, 9),new pair(5, 9)};**

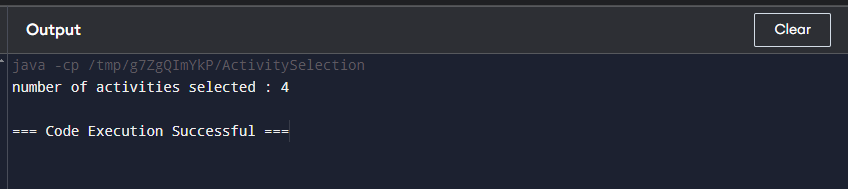
**int cnt = helper(a);**

**System.out.println("number of activities selected : " + cnt);**

**}**

**}**

**Output :**

****

**Question 2:**

**Write a Java program to implement the Quick Sort algorithm on array {67, 34. 6, 8, 100, 45, 33, 5, 78, 90} where the last element is chosen as the pivot. Scan the given array as input from user. Display the input array before and after sorting. Also display the total number of times the partition method is called.**

**Code :**

**import java.util.Scanner;**

**public class QuickSortExample {**

**static int Count = 0;**

**public static void main(String[] args) {**

**Scanner scanner = new Scanner(System.in);**

**System.out.print("enter the size of the array: ");**

**int n = scanner.nextInt();**

**int[] a = new int[n];**

**System.out.println("enter elements in the array:");**

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

**a[i] = scanner.nextInt();**

**}**

**System.out.println("Input array:");**

**for (int i : a) {**

**System.out.print(i + " ");**

**}**

**System.out.println();**

**quickSort(a, 0, n - 1);**

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

**for (int i : a) {**

**System.out.print(i + " ");**

**}**

**System.out.println();**

**System.out.println("total number of times partition method is called: " + Count);**

**}**

**public static void quickSort(int[] array, int low, int high) {**

**if (low < high) {**

**int pivotIndex = partition(array, low, high);**

**quickSort(array, low, pivotIndex - 1);**

**quickSort(array, pivotIndex + 1, high);**

**}**

**}**

**public static int partition(int[] array, int low, int high) {**

**Count++;**

**int pivot = array[high];**

**int i = (low - 1);**

**for (int j = low; j < high; j++) {**

**if (array[j] <= pivot) {**

**i++;**

**int temp = array[i];**

**array[i] = array[j];**

**array[j] = temp;**

**}**

**}**

**int temp = array[i + 1];**

**array[i + 1] = array[high];**

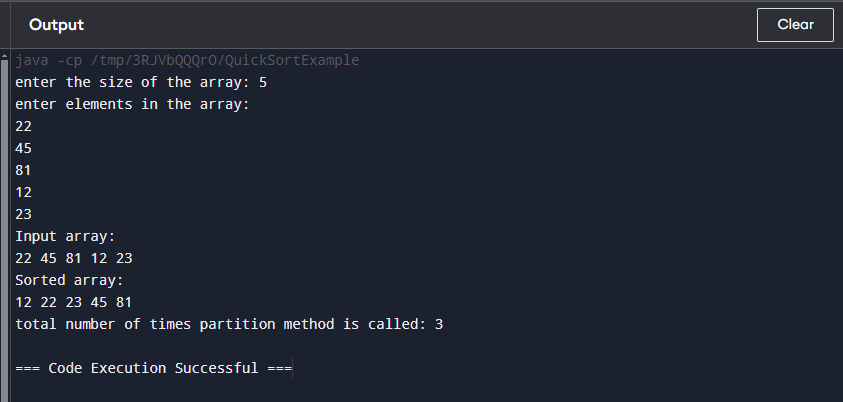
**array[high] = temp;**

**return i + 1;**

**}**

**}**

**Output :**

****

**Non-graded – 6**

Question 1:

Write a program in Java to implement Fractional Knapsack Problem.

**import java.util.Arrays;**

**import java.util.Comparator;**

**class Item {**

**int weight;**

**int value;**

**public Item(int weight, int value) {**

**this.weight = weight;**

**this.value = value;**

**}**

**}**

**class Main{**

**public static double getMaxValue(int capacity, Item[] items) {**

**Arrays.sort(items, new Comparator<Item>() {**

**public int compare(Item a, Item b) {**

**double r1 = (double) a.value / a.weight;**

**double r2 = (double) b.value / b.weight;**

**return Double.compare(r2, r1);**

**}**

**});**

**double totalValue = 0.0;**

**for (Item item : items) {**

**if (capacity == 0) break;**

**if (item.weight <= capacity) {**

**capacity -= item.weight;**

**totalValue += item.value;**

**} else {**

**totalValue += (double) item.value \* capacity / item.weight;**

**capacity = 0;**

**}**

**}**

**return totalValue;**

**}**

**public static void main(String[] args) {**

**Item[] items = {**

**new Item(10, 60),**

**new Item(20, 100),**

**new Item(30, 120)**

**};**

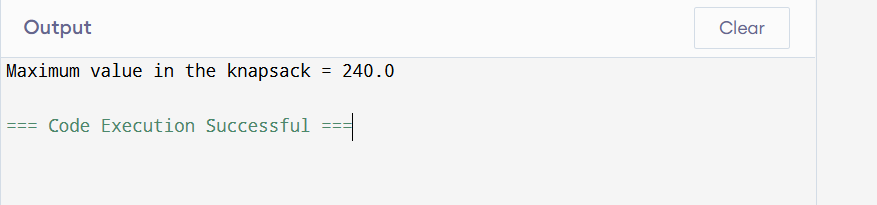
**int capacity = 50;**

**double maxValue = getMaxValue(capacity, items);**

**System.out.println("Maximum value in the knapsack = " + maxValue);**

**}**

**}**

****

Question 2:

Write a program in Java to implement Job Scheduling with Deadlines

**import java.util.Arrays;**

**import java.util.Comparator;**

**class Job {**

**int id; // Job ID**

**int deadline; // Deadline for the job**

**int profit; // Profit for the job**

**public Job(int id, int deadline, int profit) {**

**this.id = id;**

**this.deadline = deadline;**

**this.profit = profit;**

**}**

**}**

**class Main {**

**// Function to schedule jobs and maximize profit**

**public static int[] scheduleJobs(Job[] jobs, int maxTime) {**

**// Sort jobs by profit in descending order**

**Arrays.sort(jobs, new Comparator<Job>() {**

**@Override**

**public int compare(Job a, Job b) {**

**return b.profit - a.profit;**

**}**

**});**

**boolean[] timeSlots = new boolean[maxTime]; // Keep track of occupied time slots**

**int[] jobSequence = new int[maxTime]; // Store the sequence of jobs**

**int totalProfit = 0;**

**for (Job job : jobs) {**

**// Find a free slot for this job (starting from its deadline)**

**for (int t = Math.min(maxTime - 1, job.deadline - 1); t >= 0; t--) {**

**if (!timeSlots[t]) {**

**timeSlots[t] = true;**

**jobSequence[t] = job.id;**

**totalProfit += job.profit;**

**break;**

**}**

**}**

**}**

**System.out.println("Total Profit: " + totalProfit);**

**return jobSequence;**

**}**

**// Main method to test the implementation**

**public static void main(String[] args) {**

**Job[] jobs = {**

**new Job(1, 4, 20), // Job ID, Deadline, Profit**

**new Job(2, 1, 10),**

**new Job(3, 1, 40),**

**new Job(4, 1, 30)**

**};**

**int maxTime = 4; // Maximum time slots available**

**int[] scheduledJobs = scheduleJobs(jobs, maxTime);**

**System.out.print("Scheduled Jobs: ");**

**for (int jobId : scheduledJobs) {**

**if (jobId != 0) {**

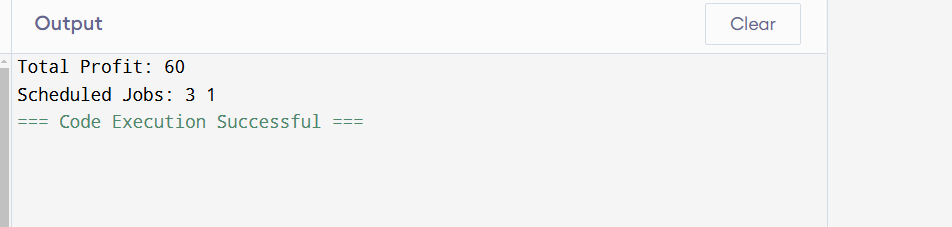
**System.out.print(jobId + " ");**

**}**

**}**

**}**

**}**

****

**Non-graded – 7**

**Question 1:**

Write a program in Java to Implement single source shortest path (Dijkstra Algorithm).

**import java.util.\*;**

**class Dijkstra {**

**private static int[] dijkstra(int[][] graph, int src) {**

**int V = graph.length;**

**int[] dist = new int[V];**

**boolean[] visited = new boolean[V];**

**Arrays.fill(dist, Integer.MAX\_VALUE);**

**dist[src] = 0;**

**for (int i = 0; i < V - 1; i++) {**

**int u = minDistance(dist, visited);**

**visited[u] = true;**

**for (int v = 0; v < V; v++) {**

**if (!visited[v] && graph[u][v] != 0 && dist[u] != Integer.MAX\_VALUE && dist[u] + graph[u][v] < dist[v]) {**

**dist[v] = dist[u] + graph[u][v];**

**}**

**}**

**}**

**return dist;**

**}**

**private static int minDistance(int[] dist, boolean[] visited) {**

**int min = Integer.MAX\_VALUE, minIndex = -1;**

**for (int v = 0; v < dist.length; v++) {**

**if (!visited[v] && dist[v] <= min) {**

**min = dist[v];**

**minIndex = v;**

**}**

**}**

**return minIndex;**

**}**

**public static void main(String[] args) {**

**int[][] graph = {**

**{0, 10, 20, 0, 0},**

**{10, 0, 5, 16, 0},**

**{20, 5, 0, 20, 10},**

**{0, 16, 20, 0, 5},**

**{0, 0, 10, 5, 0}**

**};**

**int source = 0;**

**int[] distances = dijkstra(graph, source);**

**System.out.println("Vertex \t Distance from Source");**

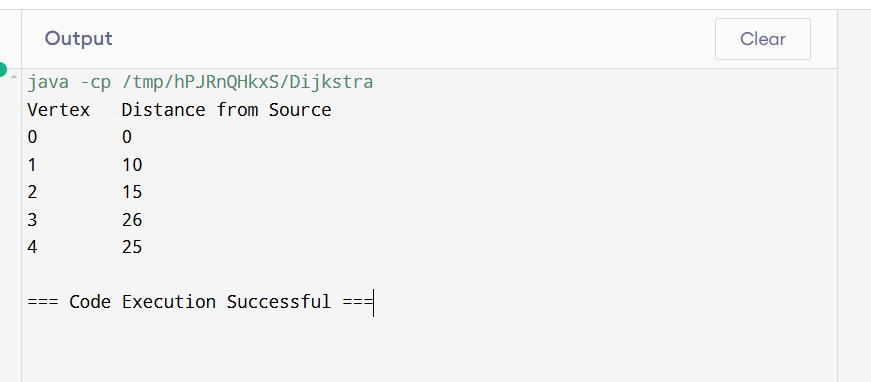
**for (int i = 0; i < distances.length; i++) {**

**System.out.println(i + " \t\t " + distances[i]);**

**}**

**}**

**}**

****

**Question 2:**

Write a program in Java to Implement 0-1 Knapsack problem using Dynamic Programming.

class Knapsack {

public static int knapSack(int W, int[] weights, int[] values, int n) {

int[][] dp = new int[n + 1][W + 1];

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

for (int w = 0; w <= W; w++) {

if (i == 0 || w == 0) {

dp[i][w] = 0;

} else if (weights[i - 1] <= w) {

dp[i][w] = Math.max(values[i - 1] + dp[i - 1][w - weights[i - 1]], dp[i - 1][w]);

} else {

dp[i][w] = dp[i - 1][w];

}

}

}

return dp[n][W];

}

public static void main(String[] args) {

int[] values = {60, 100, 120};

int[] weights = {10, 20, 30};

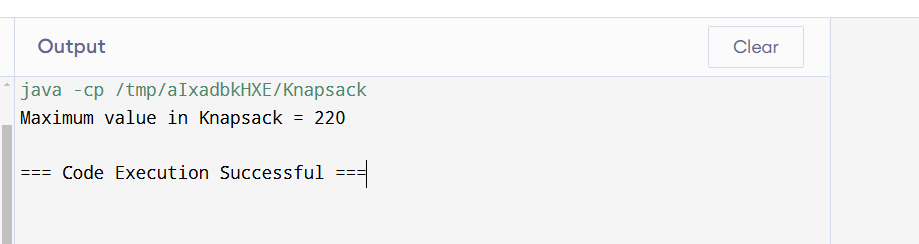
int W = 50;

int n = values.length;

System.out.println("Maximum value in Knapsack = " + knapSack(W, weights, values, n));

}

}



**Question 3:**

Write a program in Java to Implement all pair shortest path.

import java.util.\*;

class FloydWarshall {

final static int INF = 99999;

public static void floydWarshall(int[][] graph) {

int V = graph.length;

int[][] dist = new int[V][V];

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

System.arraycopy(graph[i], 0, dist[i], 0, V);

}

for (int k = 0; k < V; k++) {

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

for (int j = 0; j < V; j++) {

if (dist[i][k] != INF && dist[k][j] != INF && dist[i][k] + dist[k][j] < dist[i][j]) {

dist[i][j] = dist[i][k] + dist[k][j];

}

}

}

}

printSolution(dist);

}

public static void printSolution(int[][] dist) {

int V = dist.length;

System.out.println("Shortest distances between every pair of vertices:");

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

for (int j = 0; j < V; j++) {

if (dist[i][j] == INF)

System.out.print("INF ");

else

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

}

System.out.println();

}

}

public static void main(String[] args) {

int[][] graph = {

{0, 3, INF, 7},

{8, 0, 2, INF},

{5, INF, 0, 1},

{2, INF, INF, 0}

};

floydWarshall(graph);

}

}

