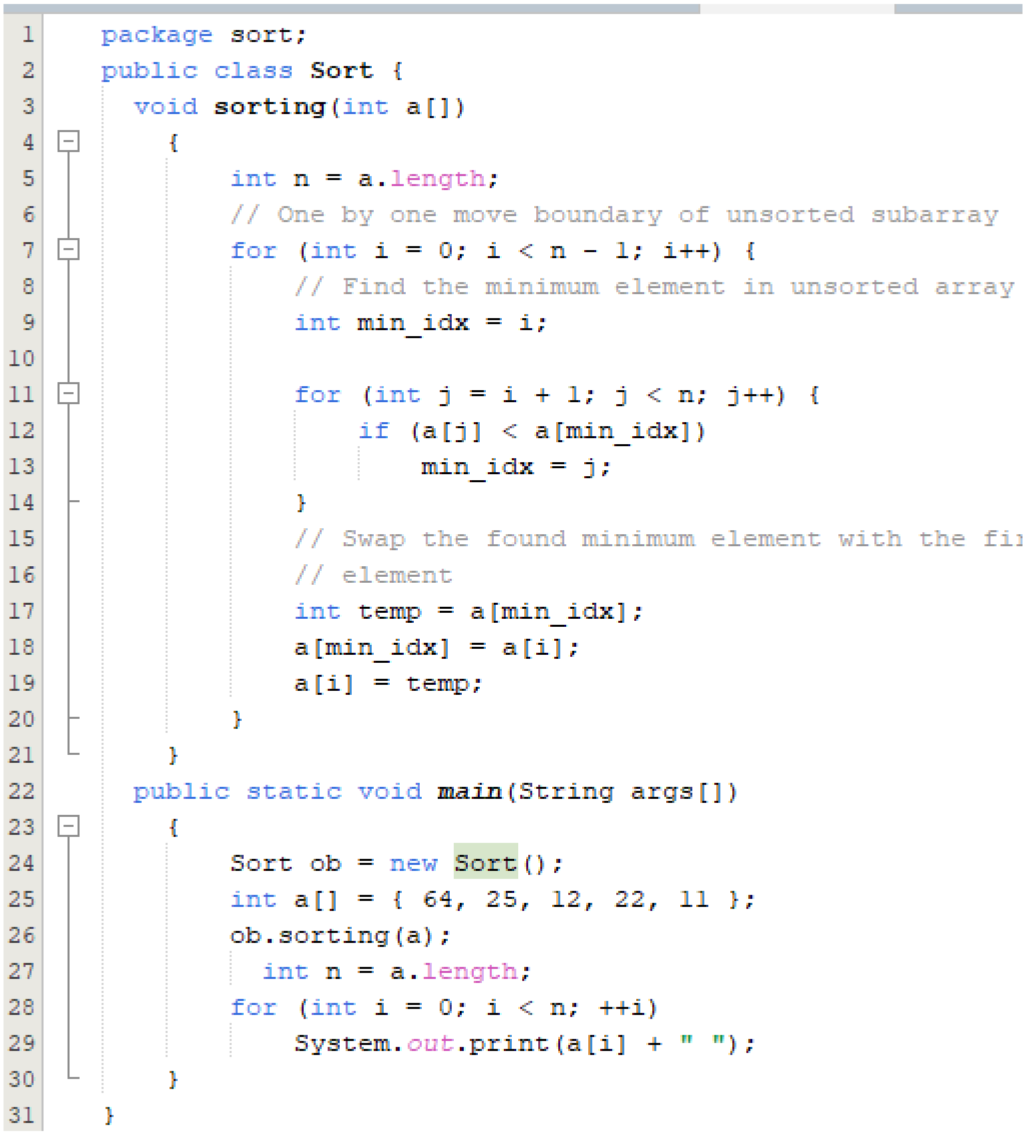
# LAB # 05

# Sorting on Linear Array

## Lab Task

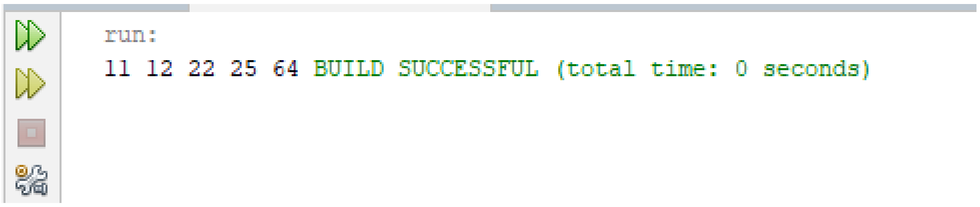
1. Write a program for Selection sort that sorts an array containing numbers, prints all the sort values of array each followed by its location.

**CODE:**



```
package sort;
public class Sort {
  void sorting(int a[])
    {
        int n = a.length;
        // One by one move boundary of unsorted subarray
        for (int i = 0; i < n - 1; i++) {
            // Find the minimum element in unsorted array
            int min_idx = i;

            for (int j = i + 1; j < n; j++) {
                if (a[j] < a[min_idx])
                    min_idx = j;
            }
            // Swap the found minimum element with the fi
            // element
            int temp = a[min_idx];
            a[min_idx] = a[i];
            a[i] = temp;
        }
    }
  public static void main(String args[])
    {
        Sort ob = new Sort();
        int a[] = { 64, 25, 12, 22, 11 };
        ob.sorting(a);
          int n = a.length;
        for (int i = 0; i < n; ++i)
            System.out.print(a[i] + " ");
    }
}
```

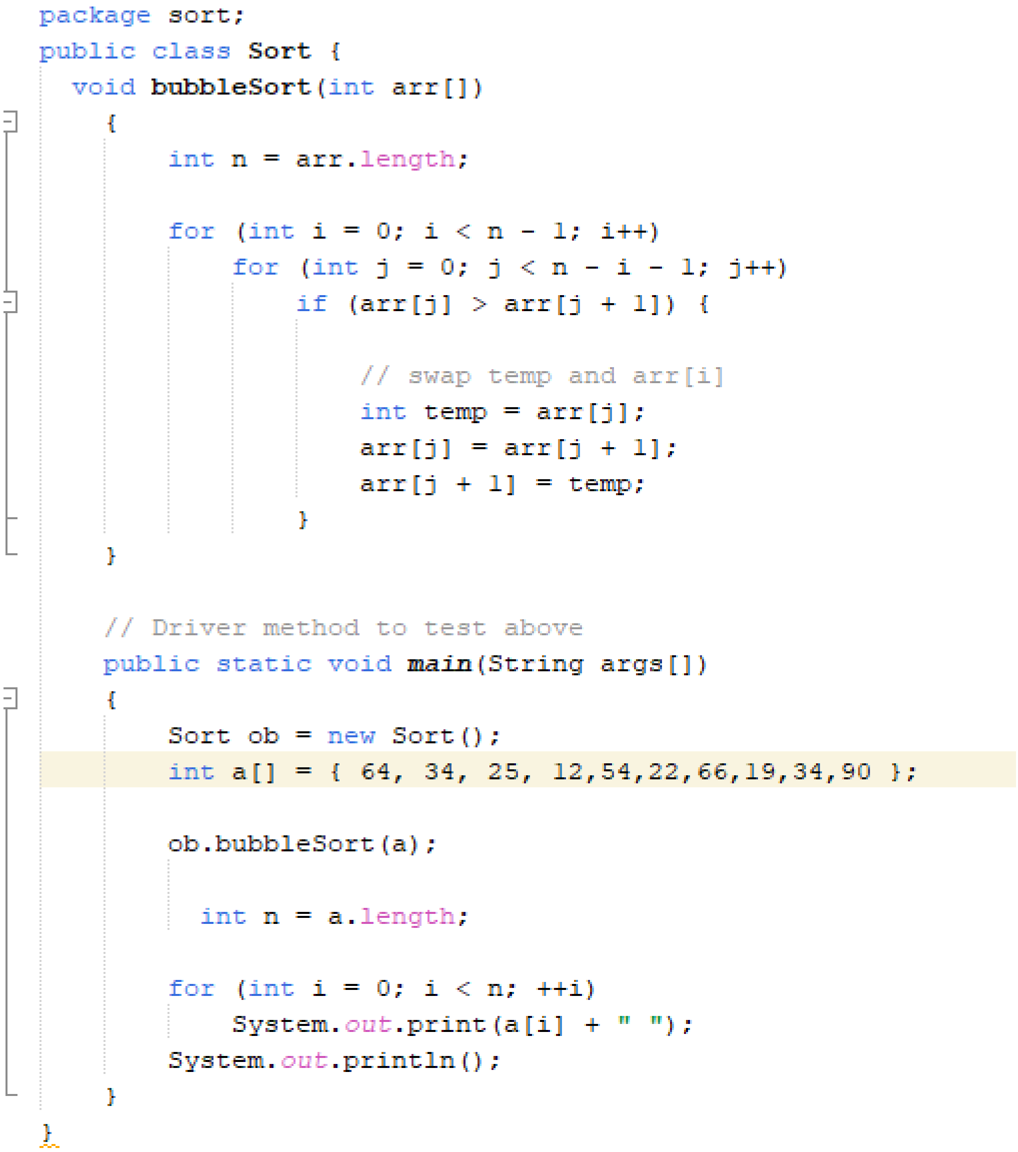
**OUTPUT:**



```
run:
11 12 22 25 64 BUILD SUCCESSFUL (total time: 0 seconds)
```

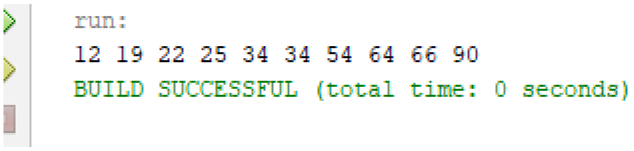
2. Write a program that takes 10 numbers as input in an array. Sort the elements of array by using Bubble sort. Print each iteration of the sorting process.

**CODE:**



```
package sort;
public class Sort {
  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]) {

                    // swap temp and arr[i]
                    int temp = arr[j];
                    arr[j] = arr[j + 1];
                    arr[j + 1] = temp;
                }
    }

    // Driver method to test above
    public static void main(String args[])
    {
        Sort ob = new Sort();
        int a[] = { 64, 34, 25, 12,54,22,66,19,34,90 };

        ob.bubbleSort(a);

          int n = a.length;

        for (int i = 0; i < n; ++i)
            System.out.print(a[i] + " ");
        System.out.println();
    }
}
```

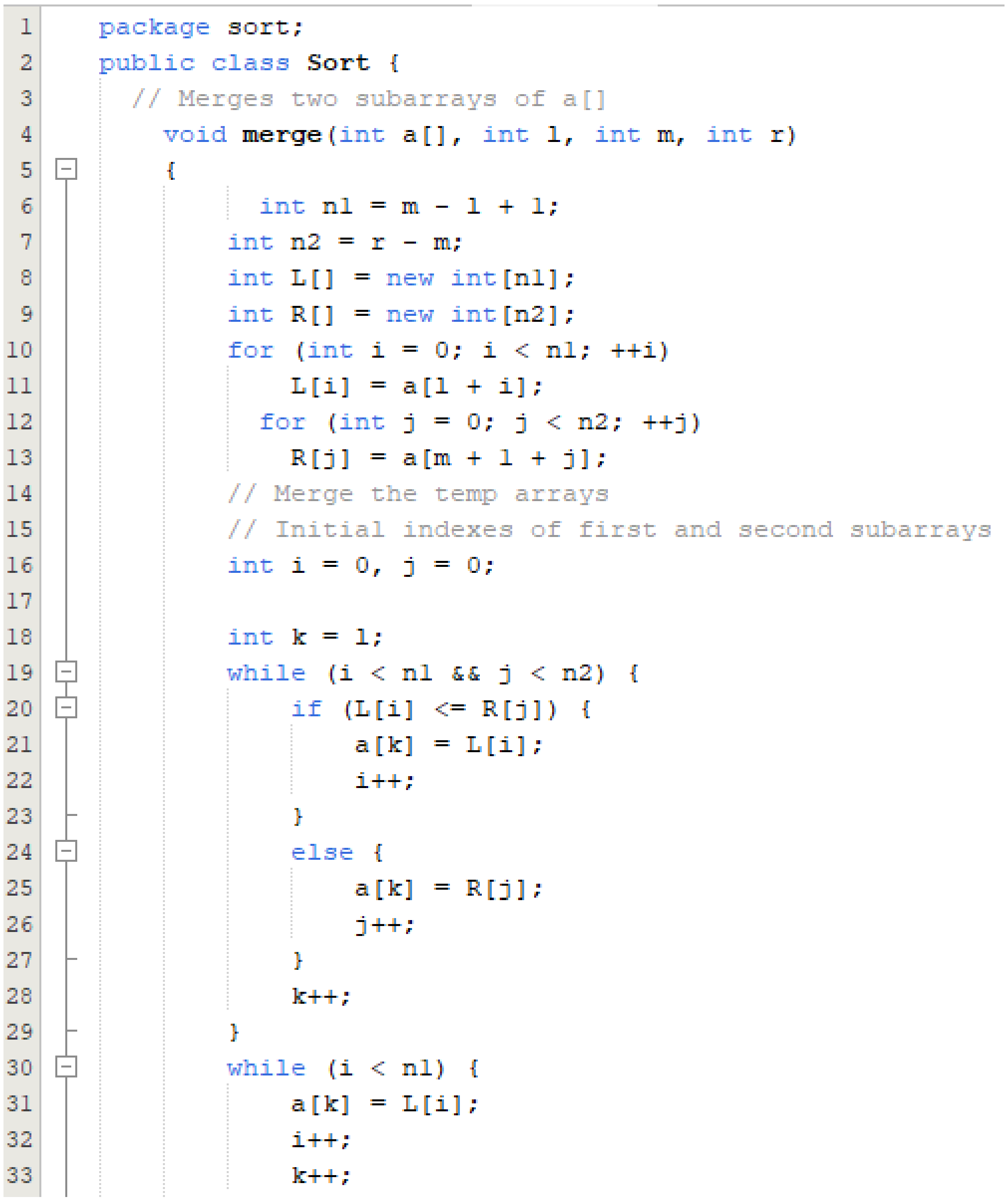
**OUTPUT:**



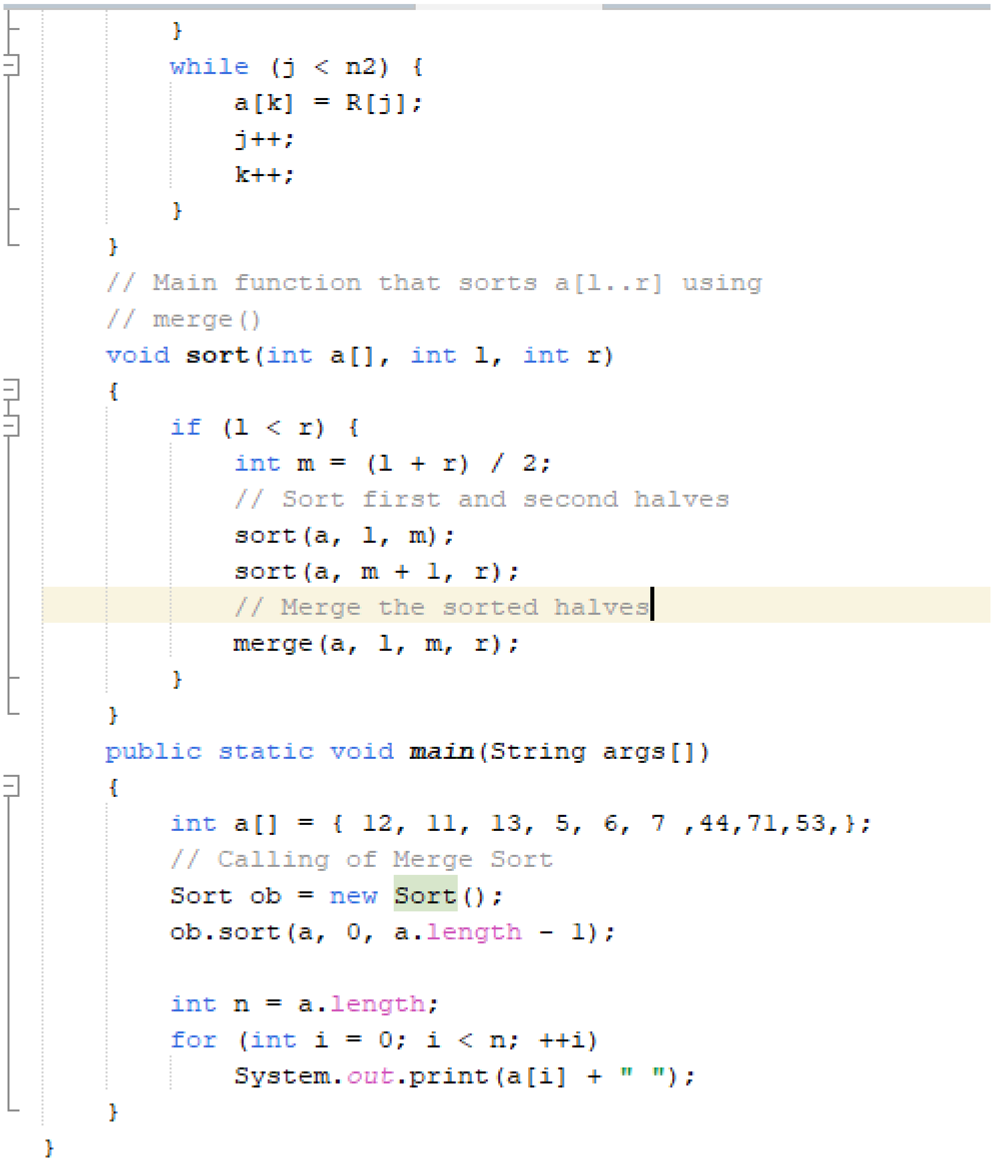
```
run:
12 19 22 25 34 34 54 64 66 90
BUILD SUCCESSFUL (total time: 0 seconds)
```

3. Write a program that takes 10 random numbers in an array. Sort the elements of array by using Merge sort applying recursive technique. Print each iteration of the sorting process.

**CODE:**

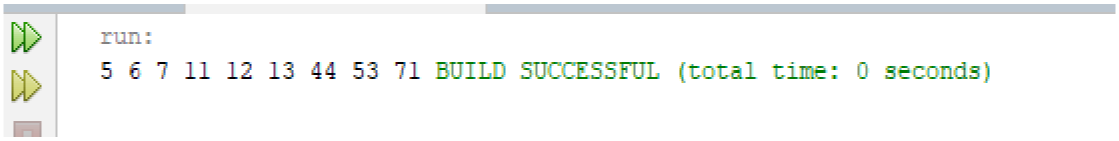


```java
package sort;
public class Sort {
  // Merges two subarrays of a[]
    void merge(int a[], int l, int m, int r)
    {
          int n1 = m - l + 1;
        int n2 = r - m;
        int L[] = new int[n1];
        int R[] = new int[n2];
        for (int i = 0; i < n1; ++i)
            L[i] = a[l + i];
          for (int j = 0; j < n2; ++j)
            R[j] = a[m + 1 + j];
        // Merge the temp arrays
        // Initial indexes of first and second subarrays
        int i = 0, j = 0;

        int k = l;
        while (i < n1 && j < n2) {
            if (L[i] <= R[j]) {
                a[k] = L[i];
                i++;
            }
            else {
                a[k] = R[j];
                j++;
            }
            k++;
        }
        while (i < n1) {
            a[k] = L[i];
            i++;
            k++;
```



```
        }
        while (j < n2) {
            a[k] = R[j];
            j++;
            k++;
        }
    }
    // Main function that sorts a[l..r] using
    // merge()
    void sort(int a[], int l, int r)
    {
        if (l < r) {
            int m = (l + r) / 2;
            // Sort first and second halves
            sort(a, l, m);
            sort(a, m + 1, r);
            // Merge the sorted halves
            merge(a, l, m, r);
        }
    }
    public static void main(String args[])
    {
        int a[] = { 12, 11, 13, 5, 6, 7 ,44,71,53,};
        // Calling of Merge Sort
        Sort ob = new Sort();
        ob.sort(a, 0, a.length - 1);

        int n = a.length;
        for (int i = 0; i < n; ++i)
            System.out.print(a[i] + " ");
    }
}
```

**OUTPUT:**



```
run:
5 6 7 11 12 13 44 53 71 BUILD SUCCESSFUL (total time: 0 seconds)
```

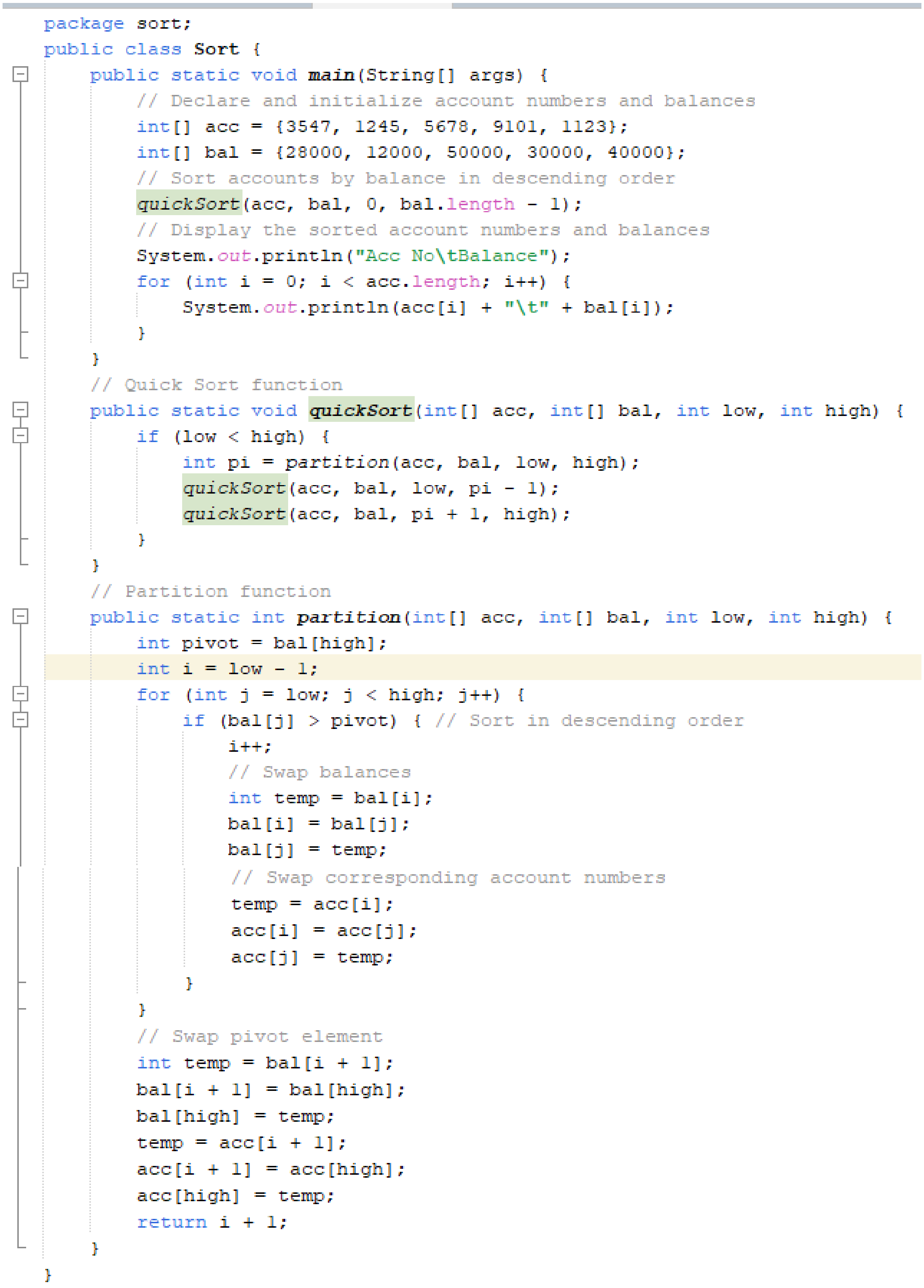
## Home Task

1. Declare an array of size n to store account balances. Initialize with values 0 to 100000 and sort Account No's according to highest balance values by using Quick sort, For e.g.:

   Account No. 3547 Balance 28000

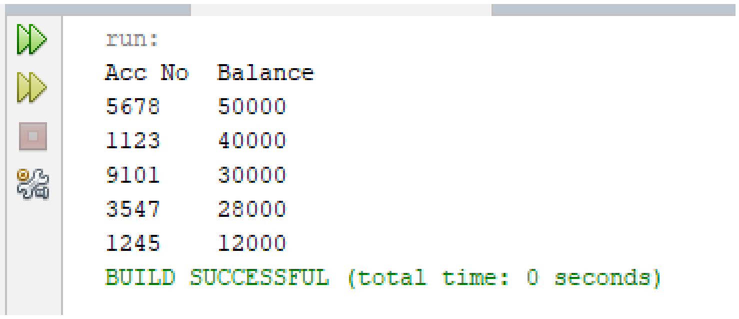
   Account No. 1245 Balance 12000

## CODE:



```
package sort;
public class Sort {
    public static void main(String[] args) {
        // Declare and initialize account numbers and balances
        int[] acc = {3547, 1245, 5678, 9101, 1123};
        int[] bal = {28000, 12000, 50000, 30000, 40000};
        // Sort accounts by balance in descending order
        quickSort(acc, bal, 0, bal.length - 1);
        // Display the sorted account numbers and balances
        System.out.println("Acc No\tBalance");
        for (int i = 0; i < acc.length; i++) {
            System.out.println(acc[i] + "\t" + bal[i]);
        }
    }
    // Quick Sort function
    public static void quickSort(int[] acc, int[] bal, int low, int high) {
        if (low < high) {
            int pi = partition(acc, bal, low, high);
            quickSort(acc, bal, low, pi - 1);
            quickSort(acc, bal, pi + 1, high);
        }
    }
    // Partition function
    public static int partition(int[] acc, int[] bal, int low, int high) {
        int pivot = bal[high];
        int i = low - 1;
        for (int j = low; j < high; j++) {
            if (bal[j] > pivot) { // Sort in descending order
                i++;
                // Swap balances
                int temp = bal[i];
                bal[i] = bal[j];
                bal[j] = temp;
                // Swap corresponding account numbers
                temp = acc[i];
                acc[i] = acc[j];
                acc[j] = temp;
            }
        }
        // Swap pivot element
        int temp = bal[i + 1];
        bal[i + 1] = bal[high];
        bal[high] = temp;
        temp = acc[i + 1];
        acc[i + 1] = acc[high];
        acc[high] = temp;
        return i + 1;
    }
}
```

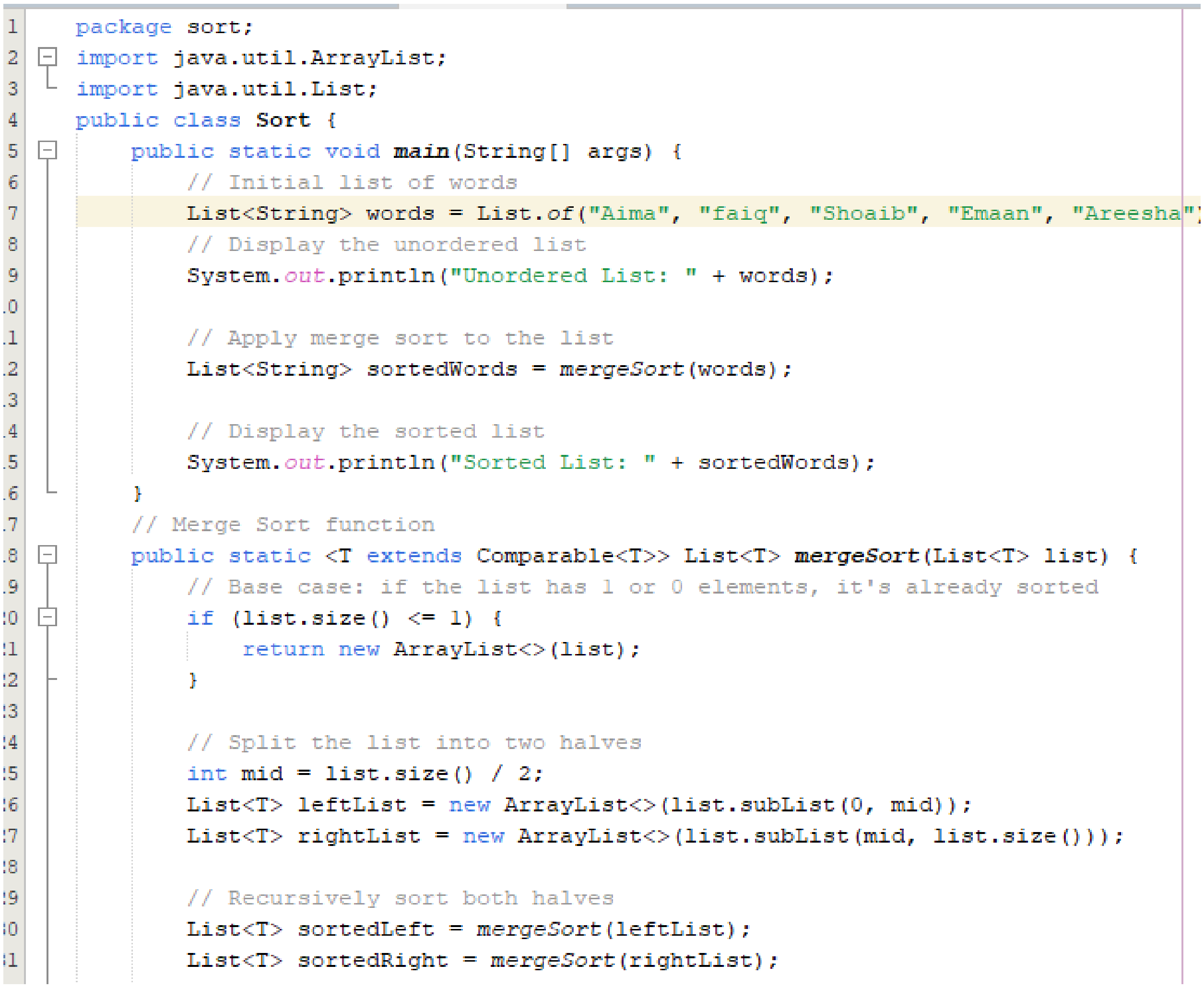
**OUTPUT:**



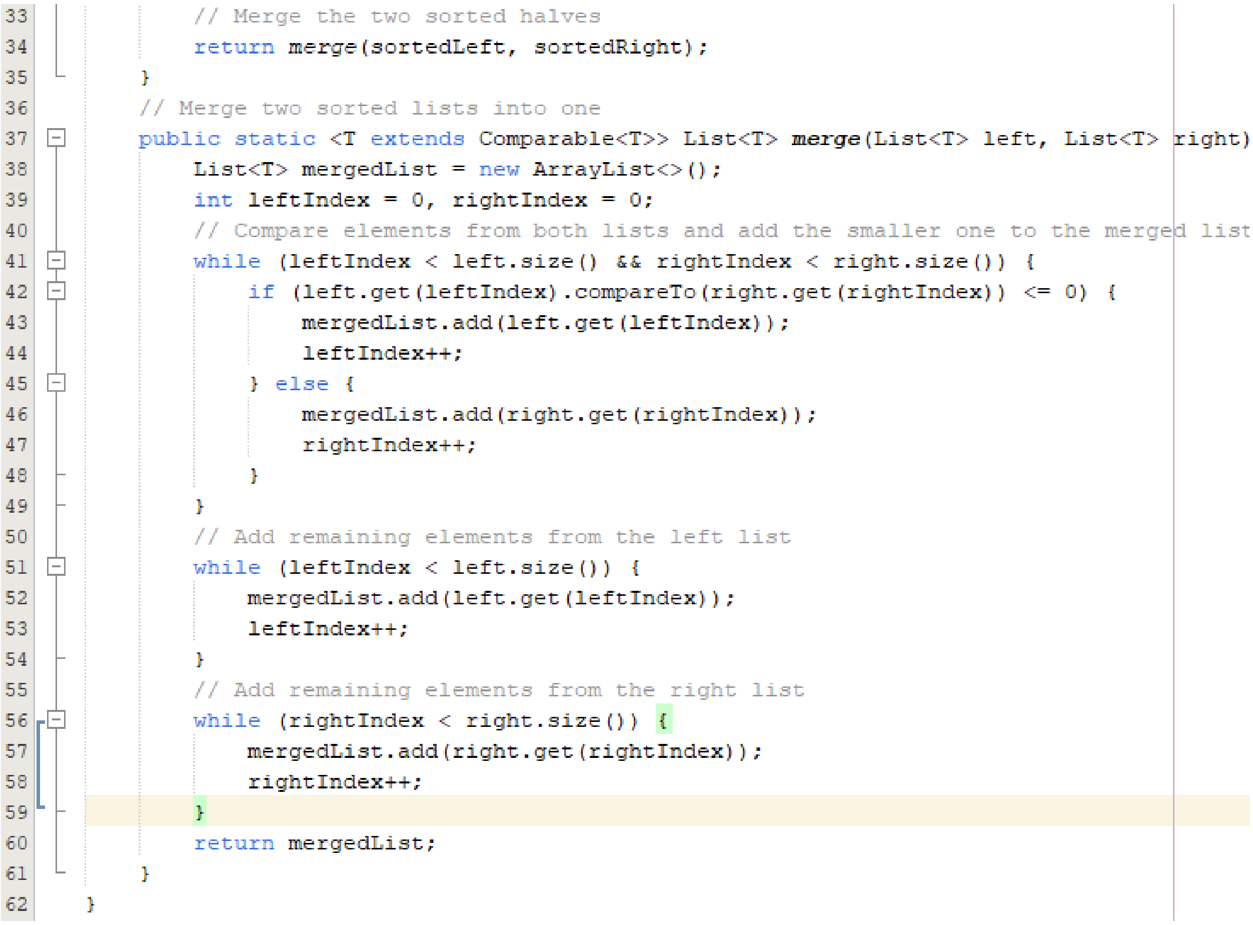
```
run:
Acc No   Balance
5678     50000
1123     40000
9101     30000
3547     28000
1245     12000
BUILD SUCCESSFUL (total time: 0 seconds)
```

2. Write a program which takes an unordered list of integers (or any other objects e.g. String), you have to rearrange the list in their natural order using merge sort.

CODE:

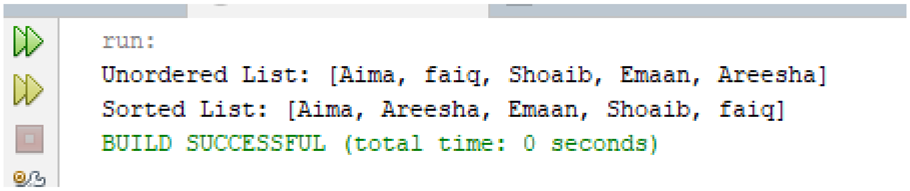


```java
package sort;
import java.util.ArrayList;
import java.util.List;
public class Sort {
    public static void main(String[] args) {
        // Initial list of words
        List<String> words = List.of("Aima", "faiq", "Shoaib", "Emaan", "Areesha"
        // Display the unordered list
        System.out.println("Unordered List: " + words);

        // Apply merge sort to the list
        List<String> sortedWords = mergeSort(words);

        // Display the sorted list
        System.out.println("Sorted List: " + sortedWords);
    }
    // Merge Sort function
    public static <T extends Comparable<T>> List<T> mergeSort(List<T> list) {
        // Base case: if the list has 1 or 0 elements, it's already sorted
        if (list.size() <= 1) {
            return new ArrayList<>(list);
        }

        // Split the list into two halves
        int mid = list.size() / 2;
        List<T> leftList = new ArrayList<>(list.subList(0, mid));
        List<T> rightList = new ArrayList<>(list.subList(mid, list.size()));

        // Recursively sort both halves
        List<T> sortedLeft = mergeSort(leftList);
        List<T> sortedRight = mergeSort(rightList);
```



```java
        // Merge the two sorted halves
        return merge(sortedLeft, sortedRight);
    }
    // Merge two sorted lists into one
    public static <T extends Comparable<T>> List<T> merge(List<T> left, List<T> right)
        List<T> mergedList = new ArrayList<>();
        int leftIndex = 0, rightIndex = 0;
        // Compare elements from both lists and add the smaller one to the merged list
        while (leftIndex < left.size() && rightIndex < right.size()) {
            if (left.get(leftIndex).compareTo(right.get(rightIndex)) <= 0) {
                mergedList.add(left.get(leftIndex));
                leftIndex++;
            } else {
                mergedList.add(right.get(rightIndex));
                rightIndex++;
            }
        }
        // Add remaining elements from the left list
        while (leftIndex < left.size()) {
            mergedList.add(left.get(leftIndex));
            leftIndex++;
        }
        // Add remaining elements from the right list
        while (rightIndex < right.size()) {
            mergedList.add(right.get(rightIndex));
            rightIndex++;
        }
        return mergedList;
    }
}
```

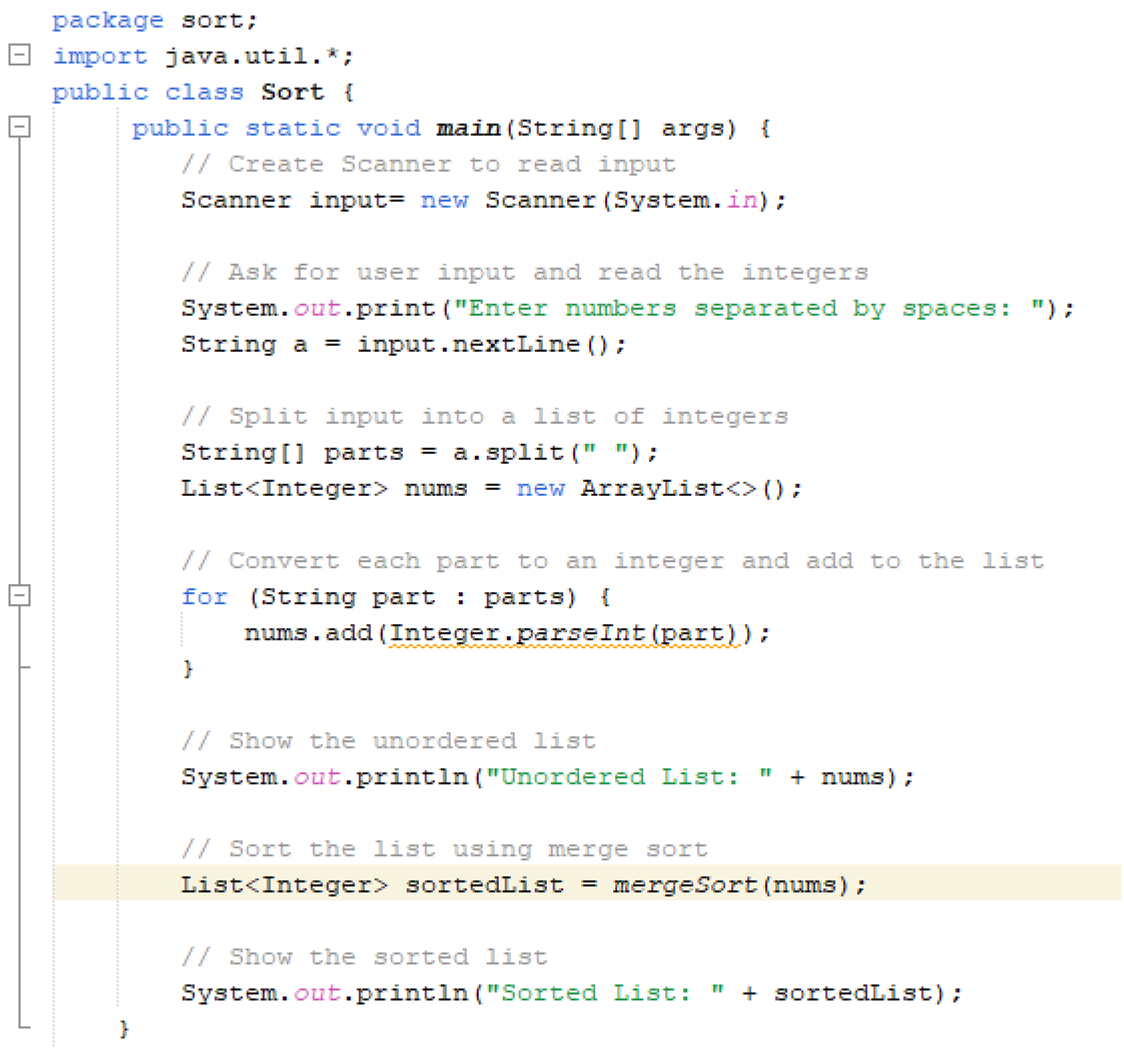
**OUTPUT:**



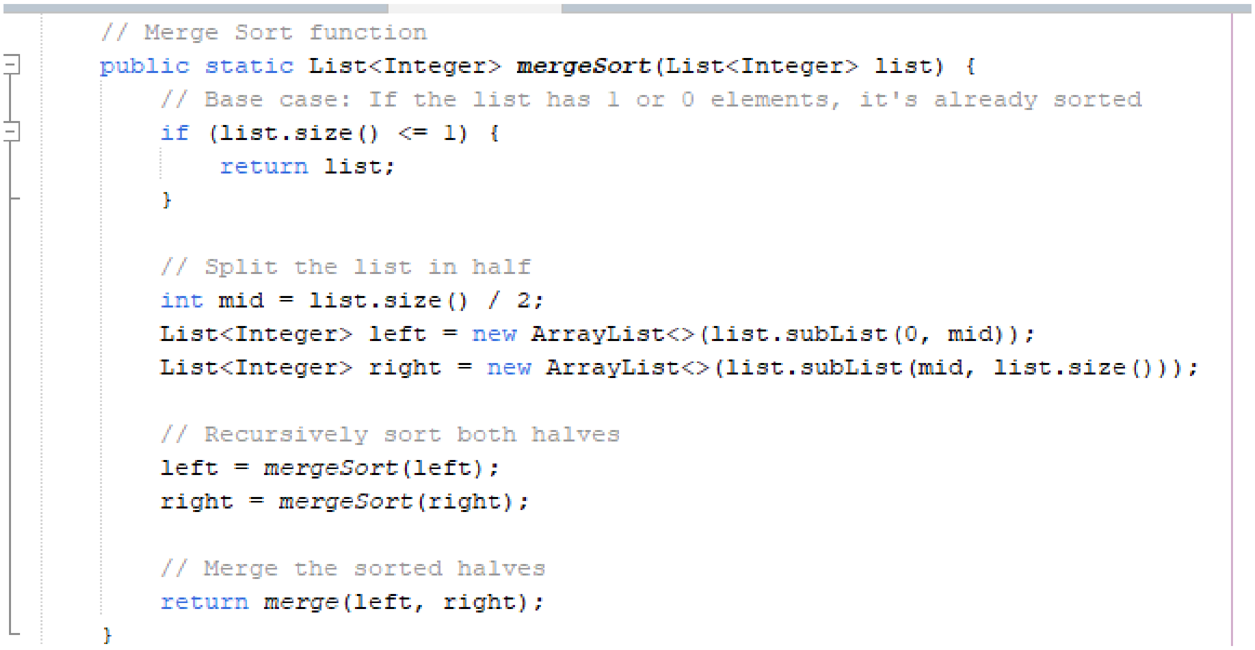
```
run:
Unordered List: [Aima, faiq, Shoaib, Emaan, Areesha]
Sorted List: [Aima, Areesha, Emaan, Shoaib, faiq]
BUILD SUCCESSFUL (total time: 0 seconds)
```

3. You are given an unordered list of integers or strings. Write a program to Take this list as input. Sort it in **natural order** using Merge Sort. For integers, this means ascending order. For strings, this means alphabetical order. Print the sorted list.

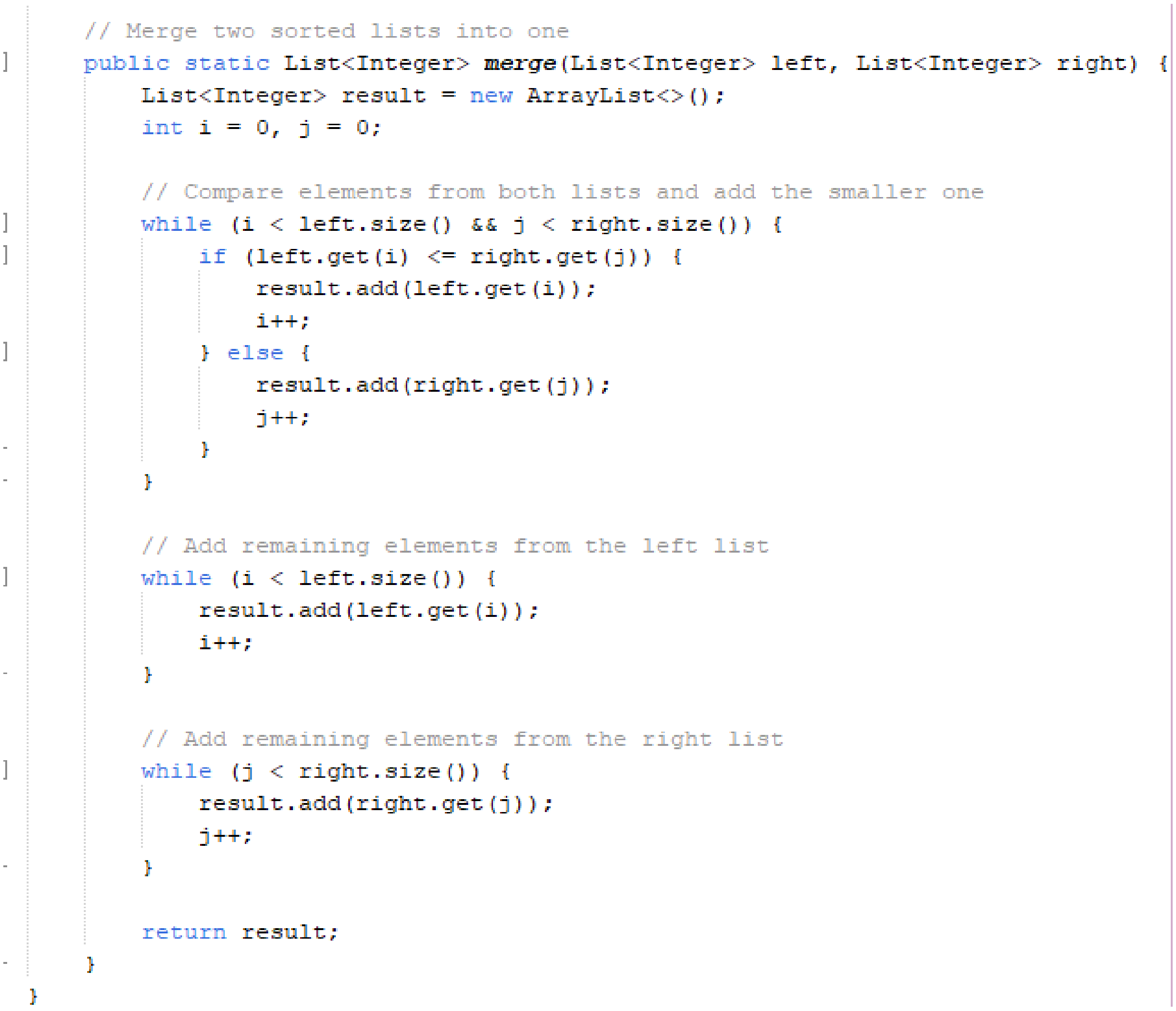
**CODE:**



```
package sort;
import java.util.*;
public class Sort {
    public static void main(String[] args) {
        // Create Scanner to read input
        Scanner input= new Scanner(System.in);

        // Ask for user input and read the integers
        System.out.print("Enter numbers separated by spaces: ");
        String a = input.nextLine();

        // Split input into a list of integers
        String[] parts = a.split(" ");
        List<Integer> nums = new ArrayList<>();

        // Convert each part to an integer and add to the list
        for (String part : parts) {
            nums.add(Integer.parseInt(part));
        }

        // Show the unordered list
        System.out.println("Unordered List: " + nums);

        // Sort the list using merge sort
        List<Integer> sortedList = mergeSort(nums);

        // Show the sorted list
        System.out.println("Sorted List: " + sortedList);
    }
```

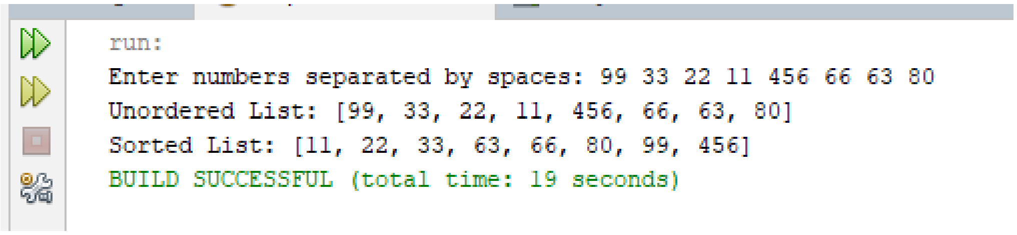


```
    // Merge Sort function
    public static List<Integer> mergeSort(List<Integer> list) {
        // Base case: If the list has 1 or 0 elements, it's already sorted
        if (list.size() <= 1) {
            return list;
        }

        // Split the list in half
        int mid = list.size() / 2;
        List<Integer> left = new ArrayList<>(list.subList(0, mid));
        List<Integer> right = new ArrayList<>(list.subList(mid, list.size()));

        // Recursively sort both halves
        left = mergeSort(left);
        right = mergeSort(right);

        // Merge the sorted halves
        return merge(left, right);
    }
```



```java
    // Merge two sorted lists into one
    public static List<Integer> merge(List<Integer> left, List<Integer> right) {
        List<Integer> result = new ArrayList<>();
        int i = 0, j = 0;

        // Compare elements from both lists and add the smaller one
        while (i < left.size() && j < right.size()) {
            if (left.get(i) <= right.get(j)) {
                result.add(left.get(i));
                i++;
            } else {
                result.add(right.get(j));
                j++;
            }
        }

        // Add remaining elements from the left list
        while (i < left.size()) {
            result.add(left.get(i));
            i++;
        }

        // Add remaining elements from the right list
        while (j < right.size()) {
            result.add(right.get(j));
            j++;
        }

        return result;
    }
}
```

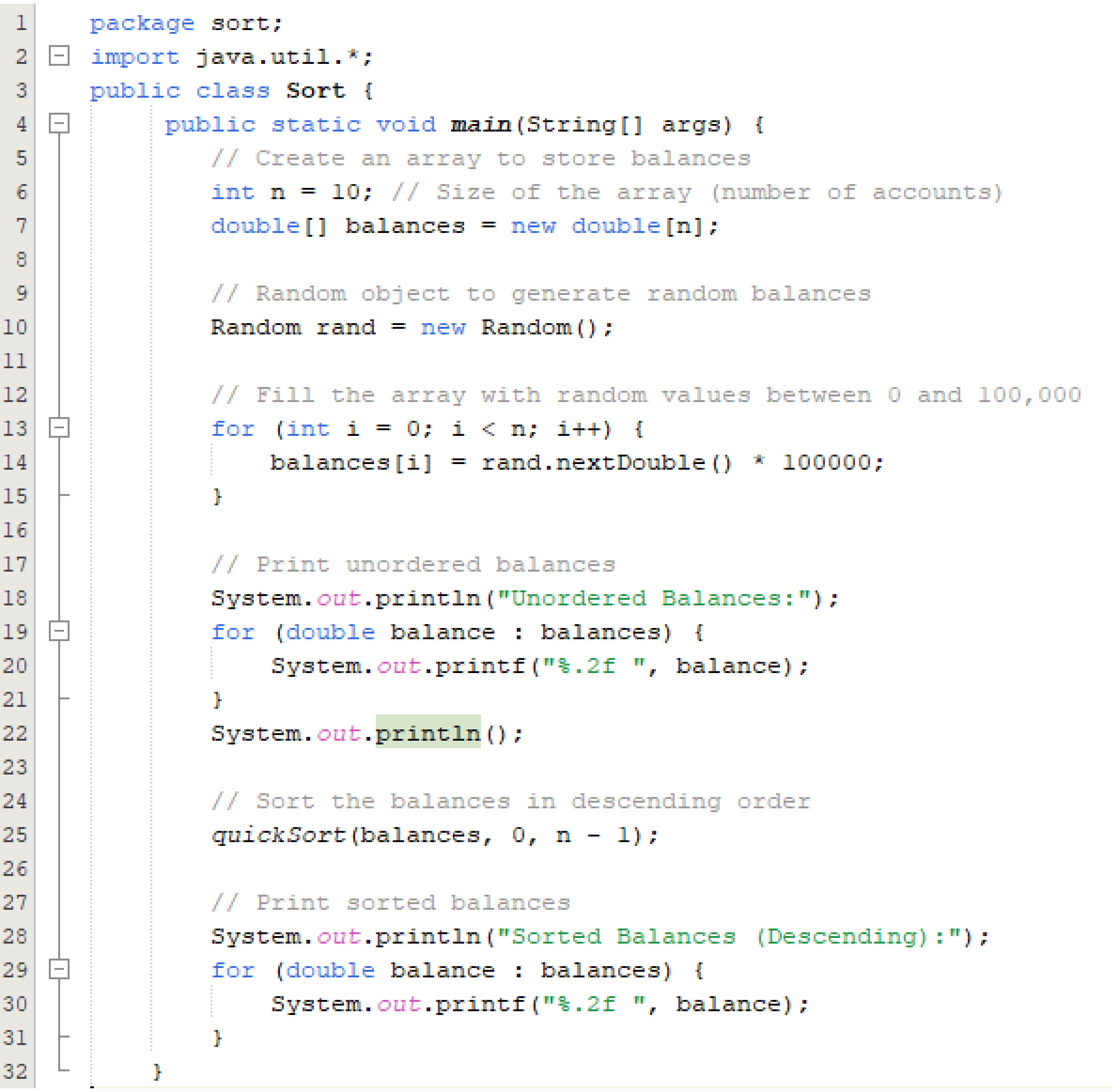
## OUTPUT



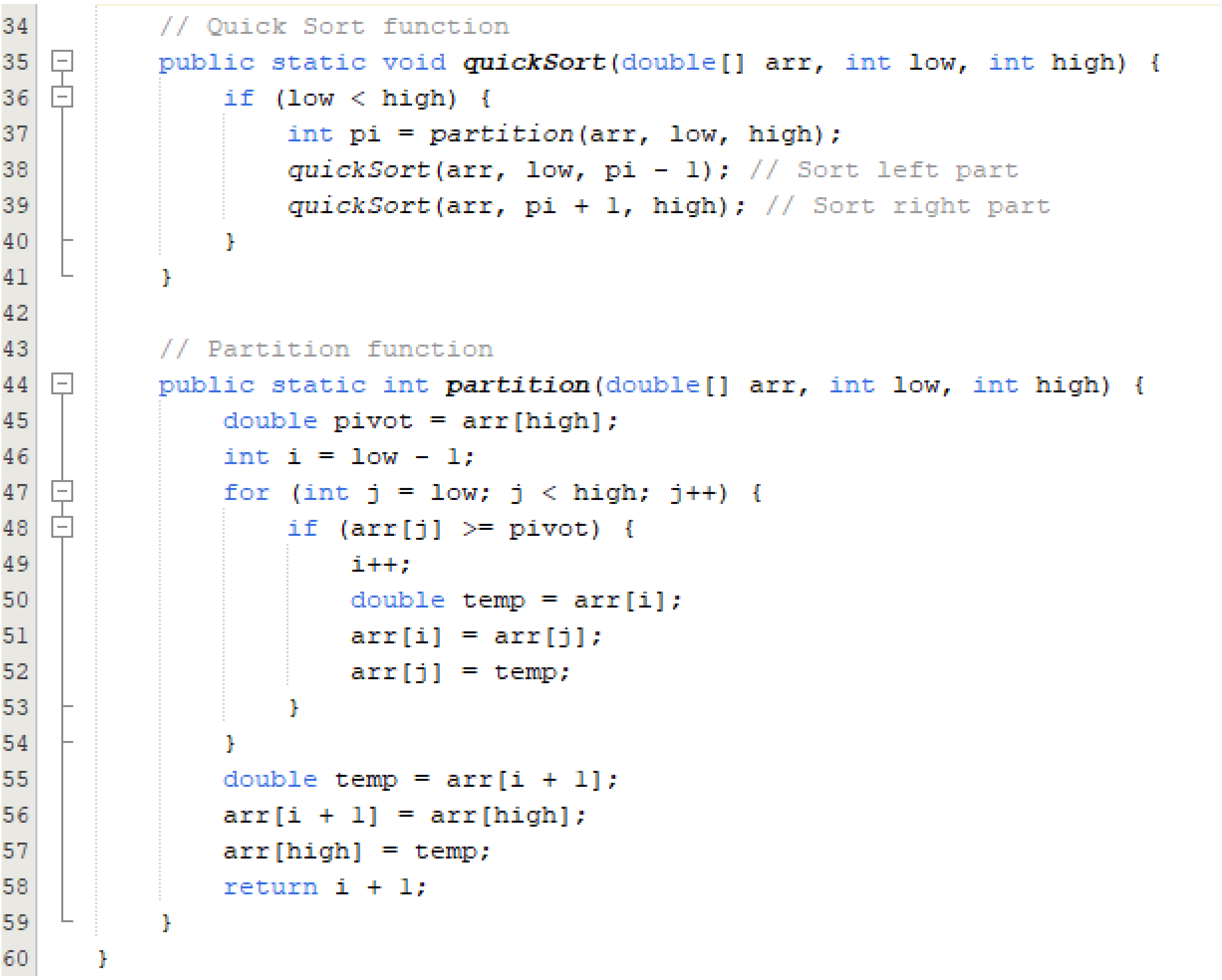
```
run:
Enter numbers separated by spaces: 99 33 22 11 456 66 63 80
Unordered List: [99, 33, 22, 11, 456, 66, 63, 80]
Sorted List: [11, 22, 33, 63, 66, 80, 99, 456]
BUILD SUCCESSFUL (total time: 19 seconds)
```

4. You are given a set of bank accounts, each with a unique account number and a balance. Write a Java program to Declare an array of size n to store account balances. Initialize each balance randomly with values between 0 and 100,000. Sort the accounts in **descending order** of their balances using Quick Sort. Print the sorted list in the format

**CODE:**

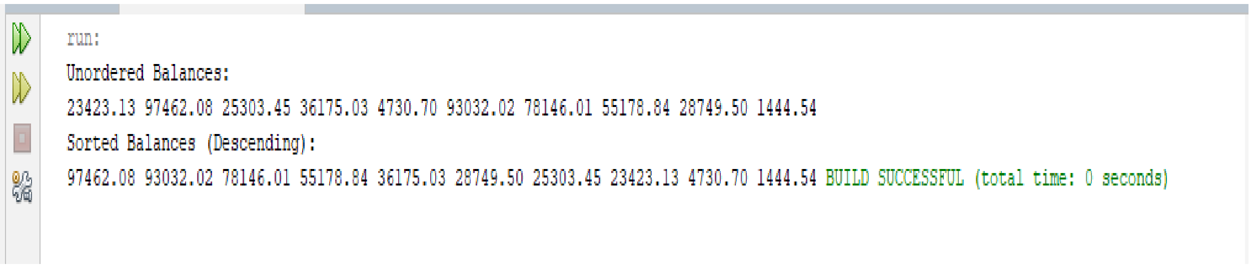


```java
package sort;
import java.util.*;
public class Sort {
    public static void main(String[] args) {
        // Create an array to store balances
        int n = 10; // Size of the array (number of accounts)
        double[] balances = new double[n];

        // Random object to generate random balances
        Random rand = new Random();

        // Fill the array with random values between 0 and 100,000
        for (int i = 0; i < n; i++) {
            balances[i] = rand.nextDouble() * 100000;
        }

        // Print unordered balances
        System.out.println("Unordered Balances:");
        for (double balance : balances) {
            System.out.printf("%.2f ", balance);
        }
        System.out.println();

        // Sort the balances in descending order
        quickSort(balances, 0, n - 1);

        // Print sorted balances
        System.out.println("Sorted Balances (Descending):");
        for (double balance : balances) {
            System.out.printf("%.2f ", balance);
        }
    }
```



```
    // Quick Sort function
    public static void quickSort(double[] arr, int low, int high) {
        if (low < high) {
            int pi = partition(arr, low, high);
            quickSort(arr, low, pi - 1); // Sort left part
            quickSort(arr, pi + 1, high); // Sort right part
        }
    }

    // Partition function
    public static int partition(double[] arr, int low, int high) {
        double pivot = arr[high];
        int i = low - 1;
        for (int j = low; j < high; j++) {
            if (arr[j] >= pivot) {
                i++;
                double temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            }
        }
        double temp = arr[i + 1];
        arr[i + 1] = arr[high];
        arr[high] = temp;
        return i + 1;
    }
}
```

## OUTPUT:



```
run:
Unordered Balances:
23423.13 97462.08 25303.45 36175.03 4730.70 93032.02 78146.01 55178.84 28749.50 1444.54
Sorted Balances (Descending):
97462.08 93032.02 78146.01 55178.84 36175.03 28749.50 25303.45 23423.13 4730.70 1444.54 BUILD SUCCESSFUL (total time: 0 seconds)
```