# LAB # 05

## Sorting on Linear Array

**OBJECTIVE:** To sort a linear array using Selection Sort, Bubble Sort and Merge Sort.

### 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.

```
1   package selectionsort;
2   public class SelectionSort {
3       public static void main(String[] args) {
4           int[] array = {64, 25, 12, 22, 11};
5           System.out.println("Original Array:");
6           printArray(array);
7           for (int i = 0; i < array.length - 1; i++) {
8               int minIndex = i;
9               for (int j = i + 1; j < array.length; j++) {
10                  if (array[j] < array[minIndex]) {
11                      minIndex = j;
12                  }
13              }
14              int temp = array[minIndex];
15              array[minIndex] = array[i];
16              array[i] = temp;
17              System.out.println("Array after step " + (i + 1) + ":");
18              printArray(array);
19          }
20      }
21      public static void printArray(int[] array) {
22          for (int num : array) {
23              System.out.print(num + " ");
24          }
25          System.out.println();
26      }
27  }
```

selectionsort.SelectionSort > main > for (int i = 0; i < array.length - 1; i++)

Output - SelectionSort (run)

```
run:
Original Array:
64 25 12 22 11
Array after step 1:
11 25 12 22 64
Array after step 2:
11 12 25 22 64
Array after step 3:
11 12 22 25 64
Array after step 4:
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.

```
package bubblesort;
import java.util.Scanner;
public class BubbleSort {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int[] array = new int[10];

        System.out.println("Enter 10 numbers:");
        for (int i = 0; i < 10; i++) {
            array[i] = scanner.nextInt();
        }

        System.out.println("Original Array:");
        printArray(array);
        for (int i = 0; i < array.length - 1; i++) {
            for (int j = 0; j < array.length - i - 1; j++) {
                if (array[j] > array[j + 1]) {
                    int temp = array[j];
                    array[j] = array[j + 1];
                    array[j + 1] = temp;
                }
            }
            System.out.println("Array after step " + (i + 1) + ":");
            printArray(array);
        }
    }
    public static void printArray(int[] array) {
        for (int num : array) {
            System.out.print(num + " ");
        }
        System.out.println();
    }
}
```

OUTPUT:

```
run:
Enter 10 numbers:
45 12 78 4 56 20 9 34 90 2
Original Array:
45 12 78 4 56 20 9 34 90 2
Array after step 1:
12 45 4 56 20 9 34 78 2 90
Array after step 2:
12 4 45 20 9 34 56 2 78 90
Array after step 3:
4 12 20 9 34 45 2 56 78 90
Array after step 4:
4 12 9 20 34 2 45 56 78 90
Array after step 5:
4 9 12 20 2 34 45 56 78 90
Array after step 6:
4 9 12 2 20 34 45 56 78 90
Array after step 7:
4 9 2 12 20 34 45 56 78 90
Array after step 8:
4 2 9 12 20 34 45 56 78 90
Array after step 9:
2 4 9 12 20 34 45 56 78 90
BUILD SUCCESSFUL (total time: 32 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.

```
package mergesort;
public class MergeSort {
    public static void main(String[] args) {
        int[] array = {38, 27, 43, 3, 9, 82, 10};
        System.out.println("Original Array:");
        printArray(array);
        mergeSort(array, 0, array.length - 1);
        System.out.println("Sorted Array:");
        printArray(array);
    }
    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);
            System.out.println("Array after merge:");
            printArray(array);
        }
    }
    public static void merge(int[] array, int left, int mid, int right) {
        int[] temp = new int[right - left + 1];
        int i = left, j = mid + 1, k = 0;
        while (i <= mid && j <= right) {
            temp[k++] = (array[i] < array[j]) ? array[i++] : array[j++];
        }
        while (i <= mid) temp[k++] = array[i++];
        while (j <= right) temp[k++] = array[j++];
        System.arraycopy(temp, 0, array, left, temp.length);
    }
    public static void printArray(int[] array) {
        for (int num : array) {
            System.out.print(num + " ");
        }
        System.out.println();
    }
}
```

OUTPUT:

```
run:
Original Array:
38 27 43 3 9 82 10
Array after merge:
27 38 43 3 9 82 10
Array after merge:
27 38 3 43 9 82 10
Array after merge:
3 27 38 43 9 82 10
Array after merge:
3 27 38 43 9 82 10
Array after merge:
3 27 38 43 9 10 82
Array after merge:
3 9 10 27 38 43 82
Sorted Array:
3 9 10 27 38 43 82
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

```
package accountbalancesorter;
class Account {
    int accountNo;
    int balance;
    Account(int accountNo, int balance) {
        this.accountNo = accountNo;
        this.balance = balance;
    }
}
public class AccountBalanceSorter {
    public static void quickSort(Account[] accounts, int low, int high) {
        if (low < high) {
            int pivot = accounts[high].balance;
            int i = low - 1;
            for (int j = low; j < high; j++) {
                if (accounts[j].balance > pivot) {
                    i++;
                    Account temp = accounts[i];
                    accounts[i] = accounts[j];
                    accounts[j] = temp;
                }
            }
            Account temp = accounts[i + 1];
            accounts[i + 1] = accounts[high];
            accounts[high] = temp;
            quickSort(accounts, low, i);
            quickSort(accounts, i + 2, high);
        }
    }
    public static void main(String[] args) {
```

```
        Account[] accounts = {
            new Account(3547, 28000),
            new Account(1245, 12000),
            new Account(9876, 45000),
            new Account(5432, 10000),
            new Account(1122, 35000)
        };
        System.out.println("Before Sorting:");
        for (Account acc : accounts) {
            System.out.println("Account No: " + acc.accountNo + " Balance: " + acc.balance);
        }
        quickSort(accounts, 0, accounts.length - 1);
        System.out.println("\nAfter Sorting:");
        for (Account acc : accounts) {
            System.out.println("Account No: " + acc.accountNo + " Balance: " + acc.balance);
        }
    }
}
```

OUTPUT:

```
run:
Before Sorting:
Account No: 3547 Balance: 28000
Account No: 1245 Balance: 12000
Account No: 9876 Balance: 45000
Account No: 5432 Balance: 10000
Account No: 1122 Balance: 35000

After Sorting:
Account No: 9876 Balance: 45000
Account No: 1122 Balance: 35000
Account No: 3547 Balance: 28000
Account No: 1245 Balance: 12000
Account No: 5432 Balance: 10000
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.

```java
package mergesortexample;
import java.util.*;
public class MergeSortExample {
    public static <T extends Comparable<T>> void mergeSort(List<T> list) {
        if (list.size() <= 1) return;

        int mid = list.size() / 2;
        List<T> left = new ArrayList<>(list.subList(0, mid));
        List<T> right = new ArrayList<>(list.subList(mid, list.size()));
        mergeSort(left);
        mergeSort(right);
        merge(list, left, right);
    }
    private static <T extends Comparable<T>> void merge(List<T> list, List<T> left, List<T> right) {
        int i = 0, j = 0, k = 0;
        while (i < left.size() && j < right.size()) {
            if (left.get(i).compareTo(right.get(j)) <= 0) list.set(k++, left.get(i++));
            else list.set(k++, right.get(j++));
        }
        while (i < left.size()) list.set(k++, left.get(i++));
        while (j < right.size()) list.set(k++, right.get(j++));
    }
    public static void main(String[] args) {
        List<Integer> numbers = Arrays.asList(5, 2, 9, 1, 3, 6);
        mergeSort(numbers);
        System.out.println(numbers);
    }
}
```

```
nergesortexample.MergeSortExample > main > numbers >
t - MergeSortExample (run) ×
run:
[1, 2, 3, 5, 6, 9]
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.

```java
package mergesortstrings;
import java.util.*;
public class MergeSortStrings {
    public static void mergeSort(List<String> list) {
        if (list.size() <= 1) return;
        int mid = list.size() / 2;
        List<String> left = new ArrayList<>(list.subList(0, mid));
        List<String> right = new ArrayList<>(list.subList(mid, list.size()));

        mergeSort(left);
        mergeSort(right);
        merge(list, left, right);
    }
    private static void merge(List<String> list, List<String> left, List<String> right) {
        int i = 0, j = 0, k = 0;
        while (i < left.size() && j < right.size()) {
            if (left.get(i).compareTo(right.get(j)) <= 0) list.set(k++, left.get(i++));
            else list.set(k++, right.get(j++));
        }
        while (i < left.size()) list.set(k++, left.get(i++));
        while (j < right.size()) list.set(k++, right.get(j++));
    }
    public static void main(String[] args) {
        List<String> words = Arrays.asList("banana", "apple", "orange", "grape");
        mergeSort(words);
        System.out.println(words);
    }
}
```

```
MergeSortStrings (run) ×
run:
[apple, banana, grape, orange]
BUILD SUCCESSFUL (total time: 0 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.

```
package quicksortbankaccounts;
import java.util.*;
class BankAccount {
    int accountNumber;
    double balance;
    public BankAccount(int accountNumber, double balance) {
        this.accountNumber = accountNumber;
        this.balance = balance;
    }
    @Override
    public String toString() {
        return "Account " + accountNumber + ": $" + balance;
    }
}
public class QuickSortBankAccounts {
    public static void quickSort(BankAccount[] accounts, int low, int high) {
        if (low < high) {
            int pivotIndex = partition(accounts, low, high);
            quickSort(accounts, low, pivotIndex - 1);
            quickSort(accounts, pivotIndex + 1, high);
        }
    }
    private static int partition(BankAccount[] accounts, int low, int high) {
        BankAccount pivot = accounts[high];
        int i = low - 1;
        for (int j = low; j < high; j++) {
            if (accounts[j].balance > pivot.balance) { // Sort in descending order
                i++;
                BankAccount temp = accounts[i];
                accounts[i] = accounts[j];
                accounts[j] = temp;
            }
        }
        BankAccount temp = accounts[i + 1];
        accounts[i + 1] = accounts[high];
        accounts[high] = temp;
        return i + 1;
    }
    public static void main(String[] args) {
        Random rand = new Random();
        int n = 5; // Number of bank accounts
        BankAccount[] accounts = new BankAccount[n];

        for (int i = 0; i < n; i++) {
            accounts[i] = new BankAccount(i + 1, rand.nextInt(100001));
        }
        System.out.println("Original Bank Accounts:");
        for (BankAccount account : accounts) {
            System.out.println(account);
        }
        quickSort(accounts, 0, accounts.length - 1);
        System.out.println("\nSorted Bank Accounts in Descending Order:");
        for (BankAccount account : accounts) {
            System.out.println(account);
        }
    }
}
```

OUTPUT:

```
run:
Original Bank Accounts:
Account 1: $1932.0
Account 2: $42088.0
Account 3: $45838.0
Account 4: $18092.0
Account 5: $37186.0

Sorted Bank Accounts in Descending Order:
Account 3: $45838.0
Account 2: $42088.0
Account 5: $37186.0
Account 4: $18092.0
Account 1: $1932.0
BUILD SUCCESSFUL (total time: 0 seconds)
```