

LAB # 05

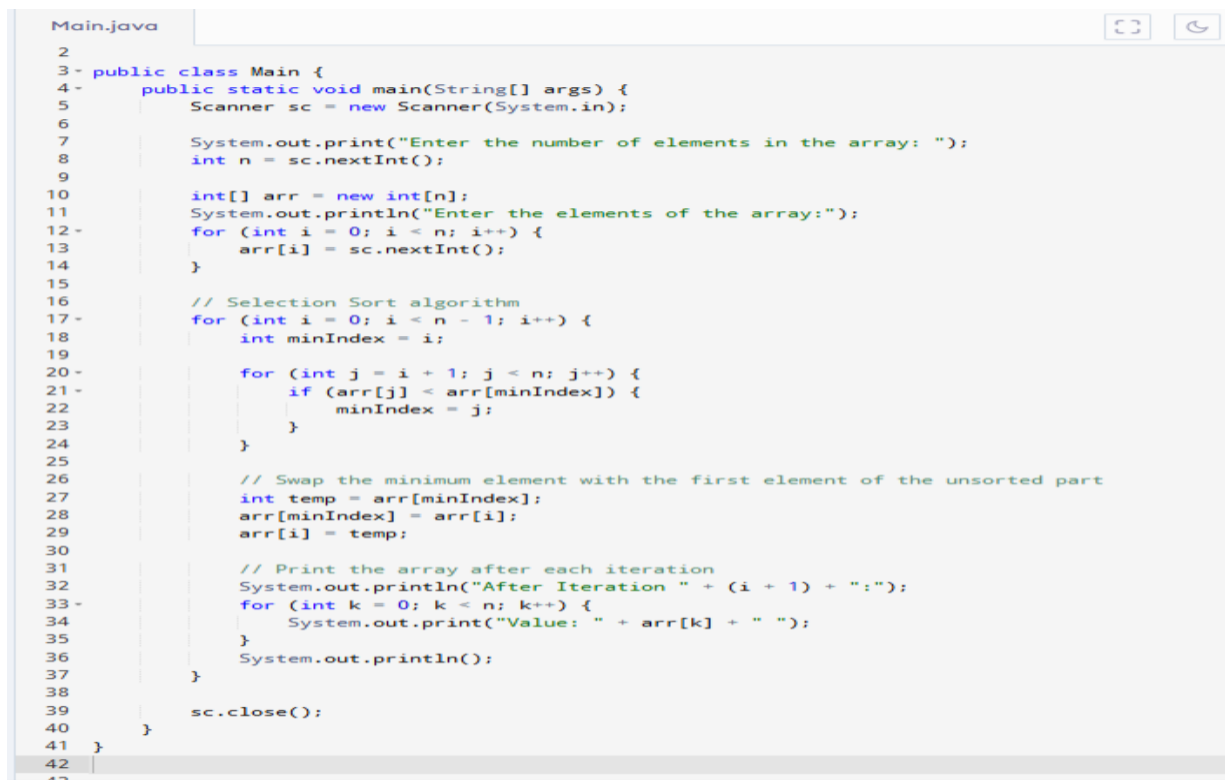
Sorting on Linear Array

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.



```
Main.java
2
3- public class Main {
4-     public static void main(String[] args) {
5         Scanner sc = new Scanner(System.in);
6
7         System.out.print("Enter the number of elements in the array: ");
8         int n = sc.nextInt();
9
10        int[] arr = new int[n];
11        System.out.println("Enter the elements of the array:");
12-        for (int i = 0; i < n; i++) {
13            arr[i] = sc.nextInt();
14        }
15
16        // Selection Sort algorithm
17-        for (int i = 0; i < n - 1; i++) {
18            int minIndex = i;
19
20            for (int j = i + 1; j < n; j++) {
21-                if (arr[j] < arr[minIndex]) {
22                    minIndex = j;
23                }
24            }
25
26            // Swap the minimum element with the first element of the unsorted part
27            int temp = arr[minIndex];
28            arr[minIndex] = arr[i];
29            arr[i] = temp;
30
31            // Print the array after each iteration
32            System.out.println("After Iteration " + (i + 1) + ":");
33-            for (int k = 0; k < n; k++) {
34                System.out.print("Value: " + arr[k] + " ");
35            }
36            System.out.println();
37        }
38
39        sc.close();
40    }
41 }
42
43
```

Output:

```
Output
^ Enter the number of elements in the array: 2
Enter the elements of the array:
2 3
After Iteration 1:
Value: 2 Value: 3

--- Code Execution Successful ---|
```

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.

```
1- import java.util.Scanner;
2
3- public class Main {
4
5-     public static void bubbleSort() {
6         Scanner sc = new Scanner(System.in);
7
8         // Create an array of 10 integers
9         int[] arr = new int[10];
10
11        // Input 10 integers from the user
12        System.out.println("Enter 10 numbers:");
13-        for (int i = 0; i < 10; i++) {
14            arr[i] = sc.nextInt(); // Read each integer
15        }
16
17        // Bubble Sort: Iterate and compare adjacent elements
18-        for (int i = 0; i < arr.length - 1; i++) {
19-            for (int j = 0; j < arr.length - 1 - i; j++) {
20                // If the current element is greater than the next element, swap them
21-                if (arr[j] > arr[j + 1]) {
22                    int temp = arr[j];
23                    arr[j] = arr[j + 1];
24                    arr[j + 1] = temp;
25                }
26            }
27            // Print the array after each iteration of bubble sort
28            System.out.println("After Iteration " + (i + 1) + ":");
29-            for (int num : arr) {
30                System.out.print(num + " ");
31            }
32            System.out.println(); // New line after each iteration
33        }
34        sc.close();
35    }
36
37-    public static void main(String[] args) {
38        bubbleSort(); // Call bubbleSort method
39    }
40 }
41
```

Output:

```

Output
Enter 10 numbers:
2 33 66 22 99 2 1 4 6 9
After Iteration 1:
2 33 22 66 2 1 4 6 9 99
After Iteration 2:
2 22 33 2 1 4 6 9 66 99
After Iteration 3:
2 22 2 1 4 6 9 33 66 99
After Iteration 4:
2 2 1 4 6 9 22 33 66 99
After Iteration 5:
2 1 2 4 6 9 22 33 66 99
After Iteration 6:
1 2 2 4 6 9
=== Code Exited With Errors ===

```

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.

```

1- import java.util.Random;
2
3- public class Main {
4
5    // Function to generate 10 random numbers and perform Merge Sort
6-    public static void mergeSort() {
7        int[] arr = new int[10];
8        Random random = new Random();
9
10       // Generating 10 random numbers
11       System.out.println("Original Array:");
12-       for (int i = 0; i < arr.length; i++) {
13           arr[i] = random.nextInt(100); // Random numbers between 0 and 99
14           System.out.print(arr[i] + " ");
15       }
16       System.out.println();
17
18       // Call recursive merge sort
19       mergeSortRec(arr, 0, arr.length - 1);
20   }
21
22   // Recursive Merge Sort function
23-   private static void mergeSortRec(int[] arr, int left, int right) {
24-       if (left < right) {
25           int mid = (left + right) / 2;
26
27           // Recursively divide the array into halves
28           mergeSortRec(arr, left, mid);
29           mergeSortRec(arr, mid + 1, right);
30
31           // Merge the two halves
32           merge(arr, left, mid, right);
33
34           // Print the array after each merge step
35           System.out.println("After Merging (" + left + " to " + right + ")");
36-           for (int i = 0; i < arr.length; i++) {
37               System.out.print(arr[i] + " ");
38           }
39           System.out.println();
40       }
41   }

```

```
41     }
42
43     // Merge function to combine two halves of the array
44     private static void merge(int[] arr, int left, int mid, int right) {
45         // Find the sizes of the two sub-arrays to be merged
46         int n1 = mid - left + 1;
47         int n2 = right - mid;
48
49         // Create temporary arrays
50         int[] leftArray = new int[n1];
51         int[] rightArray = new int[n2];
52
53         // Copy data to temporary arrays
54         System.arraycopy(arr, left, leftArray, 0, n1);
55         System.arraycopy(arr, mid + 1, rightArray, 0, n2);
56
57         // Merge the temporary arrays back into the original array
58         int i = 0, j = 0, k = left;
59         while (i < n1 && j < n2) {
60             if (leftArray[i] <= rightArray[j]) {
61                 arr[k] = leftArray[i];
62                 i++;
63             } else {
64                 arr[k] = rightArray[j];
65                 j++;
66             }
67             k++;
68         }
69
70         // Copy remaining elements of leftArray (if any)
71         while (i < n1) {
72             arr[k] = leftArray[i];
73             i++;
74             k++;
75         }
76
77         // Copy remaining elements of rightArray (if any)
78         while (j < n2) {
79             arr[k] = rightArray[j];
80             j++;
81             k++;
82         }
83     }
84
85     // Main method to start the program
86     public static void main(String[] args) {
87         mergeSort(); // Call the mergeSort function
88     }
89 }
90
```

Output:

```
Original Array:
38 60 51 94 27 55 16 18 50 72
After Merging (0 to 1):
38 60 51 94 27 55 16 18 50 72
After Merging (0 to 2):
38 51 60 94 27 55 16 18 50 72
After Merging (3 to 4):
38 51 60 27 94 55 16 18 50 72
After Merging (0 to 4):
27 38 51 60 94 55 16 18 50 72
After Merging (5 to 6):
27 38 51 60 94 16 55 18 50 72
After Merging (5 to 7):
27 38 51 60 94 16 18 55 50 72
After Merging (8 to 9):
27 38 51 60 94 16 18 55 50 72
After Merging (5 to 9):
27 38 51 60 94 16 18 50 55 72
After Merging (0 to 9):
16 18 27 38 50 51 55 60 72 94
=== Code Execution Successful ===
```

Home Tasks

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

```
1- import java.util.Random;
2
3- class Account {
4     int accountNo;
5     int balance;
6
7-     public Account(int accountNo, int balance) {
8         this.accountNo = accountNo;
9         this.balance = balance;
10    }
11
12    @Override
13-    public String toString() {
14        return "Account No. " + accountNo + " Balance " + balance;
15    }
16 }
17
18- public class Main {
19-     public static void quickSort(Account[] accounts, int low, int high) {
20-         if (low < high) {
21             int pi = partition(accounts, low, high);
22             quickSort(accounts, low, pi - 1);
23             quickSort(accounts, pi + 1, high);
24         }
25     }
26
27-     private static int partition(Account[] accounts, int low, int high) {
28         int pivot = accounts[high].balance;
29         int i = low - 1;
30         for (int j = low; j < high; j++) {
31             if (accounts[j].balance >= pivot) {
32                 i++;
33                 Account temp = accounts[i];
34                 accounts[i] = accounts[j];
35                 accounts[j] = temp;
36             }
37         }
38         Account temp = accounts[i + 1];
39         accounts[i + 1] = accounts[high];
40         accounts[high] = temp;
41         return i + 1;
42     }
43
44-     public static void main(String[] args) {
45         Random random = new Random();
46         int n = 10;
47         Account[] accounts = new Account[n];
48
49         for (int i = 0; i < n; i++) {
50             int accountNo = 1000 + random.nextInt(9000);
51             int balance = random.nextInt(100001);
52             accounts[i] = new Account(accountNo, balance);
53         }
54
55         System.out.println("Original Account Balances:");
56         for (Account account : accounts) {
57             System.out.println(account);
58         }
59
60         quickSort(accounts, 0, n - 1);
61
62         System.out.println("\nSorted Account Balances (Descending Order):");
63         for (Account account : accounts) {
64             System.out.println(account);
65         }
66     }
67 }
```

Output:

```

- Original Account Balances:
Account No. 8704 Balance 79489
Account No. 9623 Balance 6019
Account No. 3217 Balance 5824
Account No. 7439 Balance 21732
Account No. 1251 Balance 17973
Account No. 3316 Balance 75460
Account No. 4315 Balance 19447
Account No. 9165 Balance 53726
Account No. 4103 Balance 85728
Account No. 8334 Balance 21704

Sorted Account Balances (Descending Order):
Account No. 4103 Balance 85728
Account No. 8704 Balance 79489
Account No. 3316 Balance 75460
Account No. 9165 Balance 53726
Account No. 7439 Balance 21732
Account No. 8334 Balance 21704
Account No. 4315 Balance 19447
Account No. 1251 Balance 17973
Account No. 9623 Balance 6019
Account No. 3217 Balance 5824

=== Code Execution Successful ===

```

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.

```

1- import java.util.Arrays;
2
3- public class Main {
4
5-     public static void mergeSort(int[] arr) {
6-         if (arr.length < 2) {
7-             return;
8-         }
9-         int mid = arr.length / 2;
10-        int[] left = Arrays.copyOfRange(arr, 0, mid);
11-        int[] right = Arrays.copyOfRange(arr, mid, arr.length);
12
13-        mergeSort(left);
14-        mergeSort(right);
15
16-        merge(arr, left, right);
17-    }
18
19-    private static void merge(int[] arr, int[] left, int[] right) {
20-        int i = 0, j = 0, k = 0;
21-        while (i < left.length && j < right.length) {
22-            if (left[i] <= right[j]) {
23-                arr[k++] = left[i++];
24-            } else {
25-                arr[k++] = right[j++];
26-            }
27-        }
28-        while (i < left.length) {
29-            arr[k++] = left[i++];
30-        }
31-        while (j < right.length) {
32-            arr[k++] = right[j++];
33-        }
34-    }
35
36-    public static void main(String[] args) {
37-        int[] arr = { 11, 12, 13, 5, 6, 9 };
38
39-        System.out.println("Original Array:");
40-        System.out.println(Arrays.toString(arr));
41
42-        mergeSort(arr);
43
44-        System.out.println("Sorted Array:");
45-        System.out.println(Arrays.toString(arr));
46-    }
47- }
48

```

Output:

```

-----
Original Array:
[11, 12, 13, 5, 6, 9]
Sorted Array:
[5, 6, 9, 11, 12, 13]

=== Code Execution Successful ===

```

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.

```

1 import java.util.Arrays;
2 import java.util.Scanner;
3
4 public class Main {
5
6     public static <T extends Comparable<T>> void mergeSort(T[] arr) {
7         if (arr.length < 2) {
8             return;
9         }
10        int mid = arr.length / 2;
11        T[] left = Arrays.copyOfRange(arr, 0, mid);
12        T[] right = Arrays.copyOfRange(arr, mid, arr.length);
13
14        mergeSort(left);
15        mergeSort(right);
16
17        merge(arr, left, right);
18    }
19
20    private static <T extends Comparable<T>> void merge(T[] arr, T[] left, T[] right) {
21        int i = 0, j = 0, k = 0;
22        while (i < left.length && j < right.length) {
23            if (left[i].compareTo(right[j]) <= 0) {
24                arr[k++] = left[i++];
25            } else {
26                arr[k++] = right[j++];
27            }
28        }
29        while (i < left.length) {
30            arr[k++] = left[i++];
31        }
32        while (j < right.length) {
33            arr[k++] = right[j++];
34        }
35    }
36
37    public static void main(String[] args) {
38        Scanner scanner = new Scanner(System.in);
39        System.out.println("Enter 10 numbers:");
40        Integer[] arr = new Integer[10];
41        for (int i = 0; i < 10; i++) {
42            arr[i] = scanner.nextInt();
43        }
44
45        System.out.println("Original List:");
46        System.out.println(Arrays.toString(arr));
47
48        mergeSort(arr);
49
50        System.out.println("Sorted List:");
51        System.out.println(Arrays.toString(arr));
52
53        scanner.close();
54    }
55 }

```


Output:

```
Output
Enter 10 numbers:
1
2
3
4
5
6
7
8
9
0
Original List:
[1, 2, 3, 4, 5, 6, 7, 8, 9, 0]
Sorted List:
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

=== Code Execution Successful ===
```

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

```
1- import java.util.Random;
2
3- class Account {
4     int accountNo;
5     int balance;
6
7-     public Account(int accountNo, int balance) {
8         this.accountNo = accountNo;
9         this.balance = balance;
10    }
11
12    @Override
13-    public String toString() {
14        return "Account No. " + accountNo + " Balance " + balance;
15    }
16 }
17
18- public class Main {
19-     public static void quickSort(Account[] accounts, int low, int high) {
20-         if (low < high) {
21             int pi = partition(accounts, low, high);
22             quickSort(accounts, low, pi - 1);
23             quickSort(accounts, pi + 1, high);
24         }
25     }
26
27-     private static int partition(Account[] accounts, int low, int high) {
28         int pivot = accounts[high].balance;
29         int i = low - 1;
30         for (int j = low; j < high; j++) {
31             if (accounts[j].balance >= pivot) {
32                 i++;
33                 Account temp = accounts[i];
34                 accounts[i] = accounts[j];
35                 accounts[j] = temp;
36             }
37         }
38         Account temp = accounts[i + 1];
39         accounts[i + 1] = accounts[high];
40         accounts[high] = temp;
41         return i + 1;
42     }
43
44-     public static void main(String[] args) {
45         Random random = new Random();
46         int n = 10;
47         Account[] accounts = new Account[n];
48
49         for (int i = 0; i < n; i++) {
50             int accountNo = 1000 + random.nextInt(9000);
51             int balance = random.nextInt(100001);
52             accounts[i] = new Account(accountNo, balance);
53         }
54
55         System.out.println("Original Account Balances:");
56         for (Account account : accounts) {
57             System.out.println(account);
58         }
59
60         quickSort(accounts, 0, n - 1);
61
62         System.out.println("\nSorted Account Balances (Descending Order):");
63         for (Account account : accounts) {
64             System.out.println(account);
65         }
66     }
67 }
```

Output:

```
Original Account Balances:
Account No. 8704 Balance 79489
Account No. 9623 Balance 6019
Account No. 3217 Balance 5824
Account No. 7439 Balance 21732
Account No. 1251 Balance 17973
Account No. 3316 Balance 75460
Account No. 4315 Balance 19447
Account No. 9165 Balance 53726
Account No. 4103 Balance 85728
Account No. 8334 Balance 21704

Sorted Account Balances (Descending Order):
Account No. 4103 Balance 85728
Account No. 8704 Balance 79489
Account No. 3316 Balance 75460
Account No. 9165 Balance 53726
Account No. 7439 Balance 21732
Account No. 8334 Balance 21704
Account No. 4315 Balance 19447
Account No. 1251 Balance 17973
Account No. 9623 Balance 6019
Account No. 3217 Balance 5824

=== Code Execution Successful ===
```