# **LAB # 06**

# Searching in a Linear Array

**OBJECTIVE:** To find an element in linear array using Linear Search and Binary Search.

#### Lab Task

1. Declare an array of size 10 to store account balances. Initialize with values 0 to 1000000. Check all array if any value is less than 10000. Show message:

Account No. Low Balance Account No. Low Balance

#### CODE:

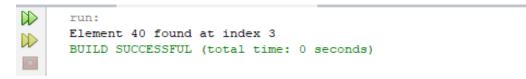
```
package search.sort;
  import java.util.Scanner;
     public class SearchSort {
4 –
        public static void main(String[] args) {
5
             // Declare and initialize the array
             int[] accountBalances = {0, 5000, 15000, 25000, 10000, 8000, 45000, 6000, 120000, 700};
6
7
             // Check for low balances and display the message
  自
8
             for (int i = 0; i < accountBalances.length; i++) {</pre>
9
                 if (accountBalances[i] < 10000) {
10
                     System.out.println("Account No. " + (i + 1) + " Low Balance");
11
                 1
12
13
14
```

# **OUTPUT:**

```
run:
Account No. 1 Low Balance
Account No. 6 Low Balance
Account No. 8 Low Balance
Account No. 10 Low Balance
Account No. 10 Low Balance
BUILD SUCCESSFUL (total time: 0 seconds)
```

2. Write a program to search in array using Array built-in class.

```
package search.sort;
2 - import java.util.Arrays;
3
    public class SearchSort {
4 -
        public static void main(String[] args) {
5
             // Declare and initialize the array
6
             int[] numbers = {10, 20, 30, 40, 50, 60, 70, 80};
7
             // Element to search
8
             int searchElement = 40;
             // Sort the array (required for binarySearch)
9
10
             Arrays.sort(numbers);
             // Search for the element using binarySearch
11
12
             int index = Arrays.binarySearch(numbers, searchElement);
13
             // Check if the element was found
14
             if (index >= 0) {
15
                System.out.println("Element " + searchElement + " found at index " + index);
16
17
                 System.out.println("Element " + searchElement + " not found");
18
19
         }
```



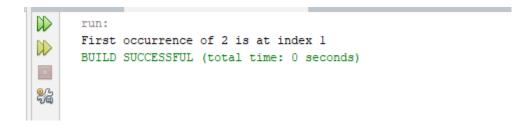
3. Given an unsorted array arr of integers, find the smallest positive integer that is **missing** from the array. You need to implement this using **binary search**. The array can contain both negative numbers and positive numbers, and you can assume that the array does not have duplicates.

```
package search.sort;
∃ import java.util.Arrays;
  public class SearchSort {
     public static void main(String[] args) {
         int[] arr = {3, 4, -1, 1};
          // Step 1: Sort the array
          Arrays.sort(arr);
          // Step 2: Perform binary search for the smallest missing positive
          int low = 0, high = arr.length - 1;
          int smallestPositive = 1; // Start with the smallest positive integer
          while (low <= high) {
              int mid = (low + high) / 2;
              // Check if the current element matches the smallest positive integer
              if (arr[mid] == smallestPositive) {
                 // If match found, increment smallestPositive and search the right half
                  smallestPositive++;
                  low = mid + 1;
              } else if (arr[mid] < smallestPositive) {</pre>
                  // If arr[mid] is smaller, search the right half
                  low = mid + 1;
              } else {
                  // If arr[mid] is larger, search the left half
                  high = mid - 1;
          // Print the smallest missing positive integer
          System.out.println("Smallest missing positive integer is at index : " + smallestPositive);
```

```
run:
Smallest missing positive integer is at index : 2
BUILD SUCCESSFUL (total time: 0 seconds)
```

4. You are given a sorted array arr[] and a target element target. Your task is to find the **first** occurrence of the target in the array using binary search. If the target is not found, return -1. You are given a sorted array arr[] and a target element target. Your task is to find the **first** occurrence of the target in the array using binary search. If the target is not found, return -1.

```
package search.sort;
🔒 🖯 import java.util.Arrays;
    public class SearchSort {
4
       public static void main(String[] args) {
5
           int[] arr = {1, 2, 2, 2, 3, 4, 5};
            int target = 2; // Target value to search for
7
            // Perform binary search to find the first occurrence of the target
8
            int low = 0;
9
            int high = arr.length - 1;
.0
            int result = -1; // Default value if the target is not found
.1 🛱
            while (low <= high) {
.2
                int mid = (low + high) / 2;
4
                if (arr[mid] == target) {
.5
                    result = mid; // Found the target, store the index
.6
.7
                    high = mid - 1; // Continue to search on the left side for the first occurrence
                 } else if (arr[mid] > target) {
.8
                    high = mid - 1; // Target is smaller, search the left half
9 🖨
                } else {
:0
                    low = mid + 1;  // Target is larger, search the right half
1
2
3 🖨
            if (result != -1) {
                System.out.println("First occurrence of " + target + " is at index " + result);
5 🖨
6
                System.out.println("Target not found in the array.");
:7
             }
:8
```



# **Home Task**

1. Write a program initializing array of size 20 and search an element using binary search.

```
package search.sort;
🛚 🗆 import java.util.Arrays;
    public class SearchSort {
     public static void main(String[] args) {
            // Initialize an unsorted array
           int[] arr = {3, 1, 5, 2, 44, 23, 134, 10, 33, 23, 90, 234, 333, 44, 123, 78, 23, 24, 22};
           int target = 10; // Element to search for
            // Sort the array before applying binary search
            Arrays.sort(arr);
            System.out.println("SORTED ARRAY"+Arrays.toString(arr));
            // Binary search logic inside the main method
            int low = 0;
            int high = arr.length - 1;
            int result = -1;
 阜
            while (low <= high) {
               int mid = (low + high) / 2;
if (arr[mid] == target) {
                   result = mid; // Element found
                   break;
 阜
                } else if (arr[mid] < target) {
                   low = mid + 1; // Search the right half
 阜
                } else {
                   high = mid - 1; // Search the left half
            if (result == -1) {
 System.out.println("Element " + target + " not found.");
            } else {
                System.out.println("Element " + target + " found at index " + result);
```

```
run:
SORTED ARRAY[1, 2, 3, 5, 10, 22, 23, 23, 24, 33, 44, 44, 78, 90, 123, 134, 234, 333]
Element 10 found at index 4
BUILD SUCCESSFUL (total time: 0 seconds)
```

2. Write a function called occurrences that, given an array of numbers A, prints all the distinct values in A each followed by its number of occurrences. For example, if A = (28, 1, 0, 1, 0, 3, 4, 0, 0, 3), the function should output the following five lines (here separated by a semicolon) "28 1; 1 2; 0 4; 3 2; 4 1".

### **CODE**

```
package search.sort;
import java.util.Arrays;
  public class SearchSort {
    public static void occurrences(int[] A) {
          // Iterate through the array to find distinct elements
          for (int i = 0; i < A.length; i++) {
              boolean found = false;
              // Check if the current element has already been counted
               for (int j = 0; j < i; j++) {
                  if (A[i] == A[j]) {
                       found = true;
                      break;
                  }
              // If the element is not counted yet, count its occurrences
              if (!found) {
                  int count = 0;
                   for (int k = 0; k < A.length; k++) {
                      if (A[k] == A[i]) {
                          count++;
                       1
                   // Print the element and its count
                  System.out.print(A[i] + " " + count + "; ");
          }
      public static void main(String[] args) {
          int[] A = \{28, 1, 4, 45, 44, 45, 45, 3, 12, 12, 34\};
          occurrences (A);
      }
  }
```

# **OUTPUT**



```
run:
28 1; 1 1; 4 1; 45 3; 44 1; 3 1; 12 2; 34 1; BUILD SUCCESSFUL (total time: 0 seconds)
```

3. Assume a bank's system needs to identify accounts with critically low balances and alert the user. Test the function with various balance values to ensure it correctly identifies all accounts below the threshold.

### **CODE**

```
package search.sort;
import java.util.Arrays;
  public class SearchSort {
      public static String checkBalance(double accountBalance, double threshold) {
if (accountBalance < threshold) { // Function to check if balance is critically low
              return "Alert: Balance is critically low!";
阜
          } else {
             return "Balance is okay.";
          1
      1
      // Function to search for a specific balance in the account list
早早早
      public static String searchBalance(double[] balances, double targetBalance) {
          for (double balance : balances) {
              if (balance == targetBalance) {
                  return "Balance " + targetBalance + " found in the system.";
          return "Balance " + targetBalance + " not found in the system.";
public static void main(String[] args) {
          double[] testBalances = {100, 45, 30, 10, 60};
                                                          // Test data for account balances
          double threshold = 50;
                                  // Define the threshold for low balance
          for (double balance : testBalances) {
              System.out.println("Balance: " + balance + " - " + checkBalance(balance, threshold));
          // Test searching for specific balance
          double targetBalance = 30; // Change this value to test different balances
          System.out.println(searchBalance(testBalances, targetBalance));
          targetBalance = 200; // Test for a balance that is not in the list
          System.out.println(searchBalance(testBalances, targetBalance));
```

#### **OUTPUT:**

```
Cutput - Searchsort (run) ×

run:
Balance: 100.0 - Balance is okay.
Balance: 45.0 - Alert: Balance is critically low!
Balance: 30.0 - Alert: Balance is critically low!
Balance: 10.0 - Alert: Balance is critically low!
Balance: 60.0 - Balance is okay.
Balance 30.0 found in the system.
Balance 200.0 not found in the system.
BUILD SUCCESSFUL (total time: 0 seconds)
```