Ranjith R 22IT085 Day 5 DSA Practice

1. Buy And Sell Stock

Given an array prices[] of length n, representing the prices of the stocks on different days. The task is to find the maximum profit possible by buying and selling the stocks on different days when at most one transaction is allowed. Here one transaction means 1 buy + 1 Sell. If it is not possible to make a profit then return 0.

Note: Stock must be bought before being sold.

Examples:

Input: prices[] = [7, 10, 1, 3, 6, 9, 2]
Output: 8
Explanation: You can buy the stock on day 2 at price = 1 and sell it on day 5 at price = 9. Hence, the profit is 8.

PROGRAM:

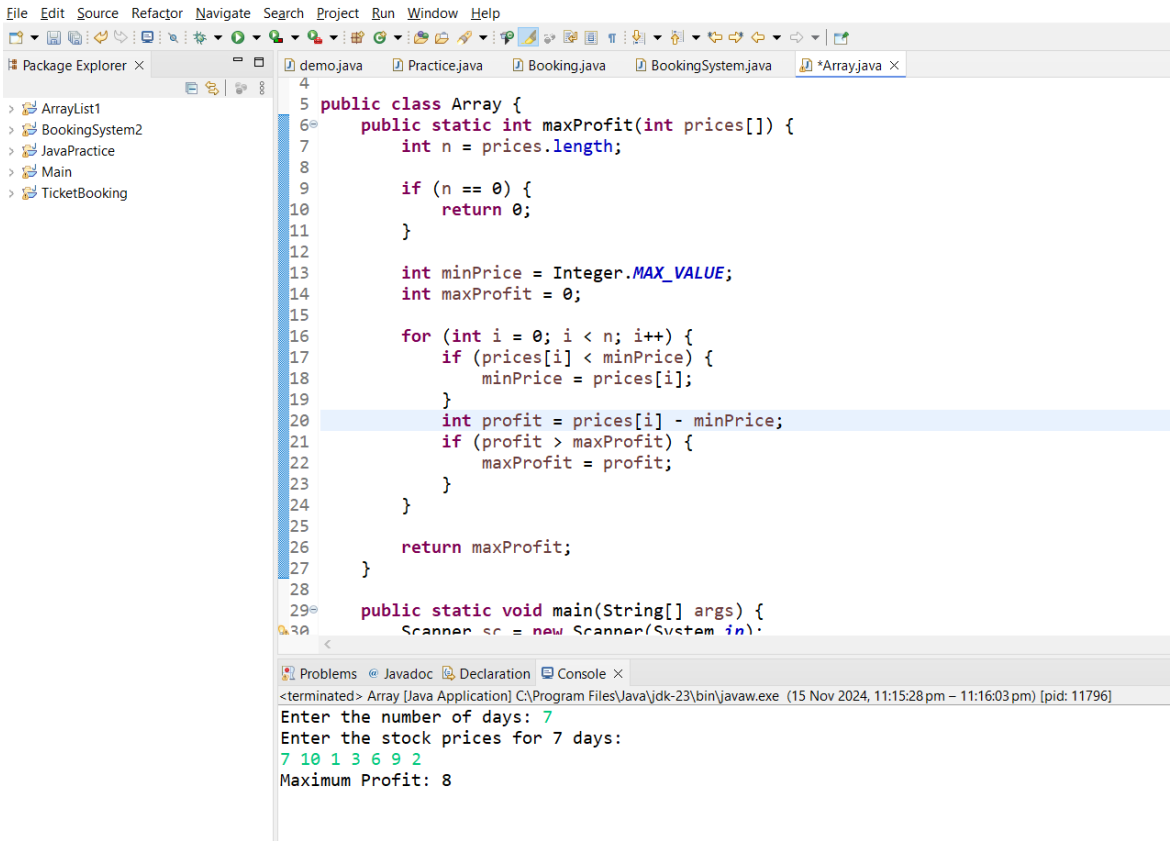
```
package JavaPractice;


import java.util.Scanner;


public class Array {
    public static int maxProfit(int prices[]) {
        int n = prices.length;


        if (n == 0) {
            return 0;
```

```
        }

        int minPrice = Integer.MAX_VALUE;
        int maxProfit = 0;

        for (int i = 0; i < n; i++) {
            if (prices[i] < minPrice) {
                minPrice = prices[i];
            }
            int profit = prices[i] - minPrice;
            if (profit > maxProfit) {
                maxProfit = profit;
            }
        }

        return maxProfit;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        System.out.print("Enter the number of days: ");
        int n = sc.nextInt();

        int[] prices = new int[n];
```

```
        System.out.println("Enter the stock prices for " + n + "
days:");
        for (int i = 0; i < n; i++) {
            prices[i] = sc.nextInt();
        }

        System.out.println("Maximum Profit: " + maxProfit(prices));
    }
}
```

Time Complexity: O(n)

OUTPUT:



2. Coin Exchange

Given an integer array of coins[ ] representing different denominations of currency and an integer sum, find the number of ways you can make a sum by using different combinations from coins[ ].

Note: Assume that you have an infinite supply of each type of coin. And you can use any coin as many times as you want.

Answers are guaranteed to fit into a 32-bit integer.

Examples:

Input: coins[] = [1, 2, 3], sum = 4

Output: 4

Explanation: Four Possible ways are: [1, 1, 1, 1], [1, 1, 2], [2, 2], [1, 3].

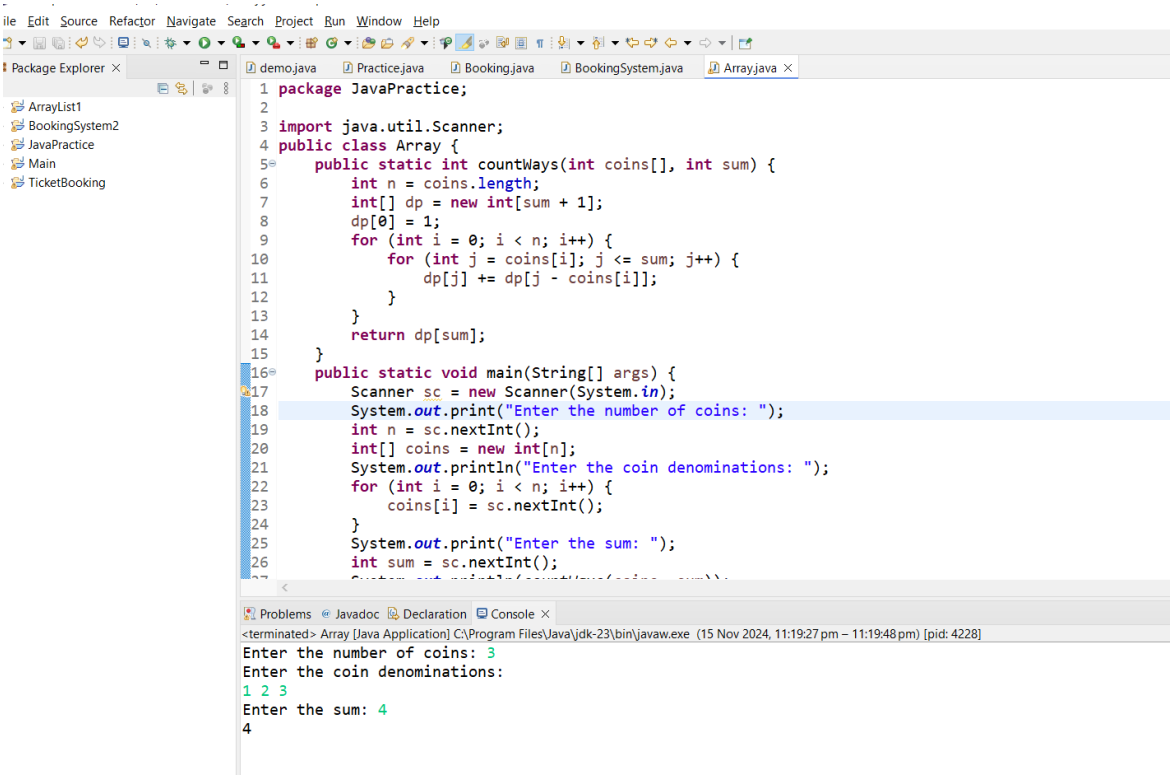
PROGRAM:

```
package JavaPractice;


import java.util.Scanner;
public class Array {
    public static int countWays(int coins[], int sum) {
        int n = coins.length;
        int[] dp = new int[sum + 1];
        dp[0] = 1;
        for (int i = 0; i < n; i++) {
            for (int j = coins[i]; j <= sum; j++) {
                dp[j] += dp[j - coins[i]];
            }
```

```
        }
        return dp[sum];
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the number of coins: ");
        int n = sc.nextInt();
        int[] coins = new int[n];
        System.out.println("Enter the coin denominations: ");
        for (int i = 0; i < n; i++) {
            coins[i] = sc.nextInt();
        }
        System.out.print("Enter the sum: ");
        int sum = sc.nextInt();
        System.out.println(countWays(coins, sum));
    }
}
```

Time Complexity: O(n * sum)

OUTPUT:



3. First and Last Occurrences

Given a sorted array arr with possibly some duplicates, the task is to find the first and last occurrences of an element x in the given array.

Note: If the number x is not found in the array then return both the indices as -1.

Examples:

Input: arr[] = [1, 3, 5, 5, 5, 5, 67, 123, 125], x = 5

Output: [2, 5]

Explanation: First occurrence of 5 is at index 2 and last occurrence of 5 is at index 5

PROGRAM:

```
package JavaPractice;
```

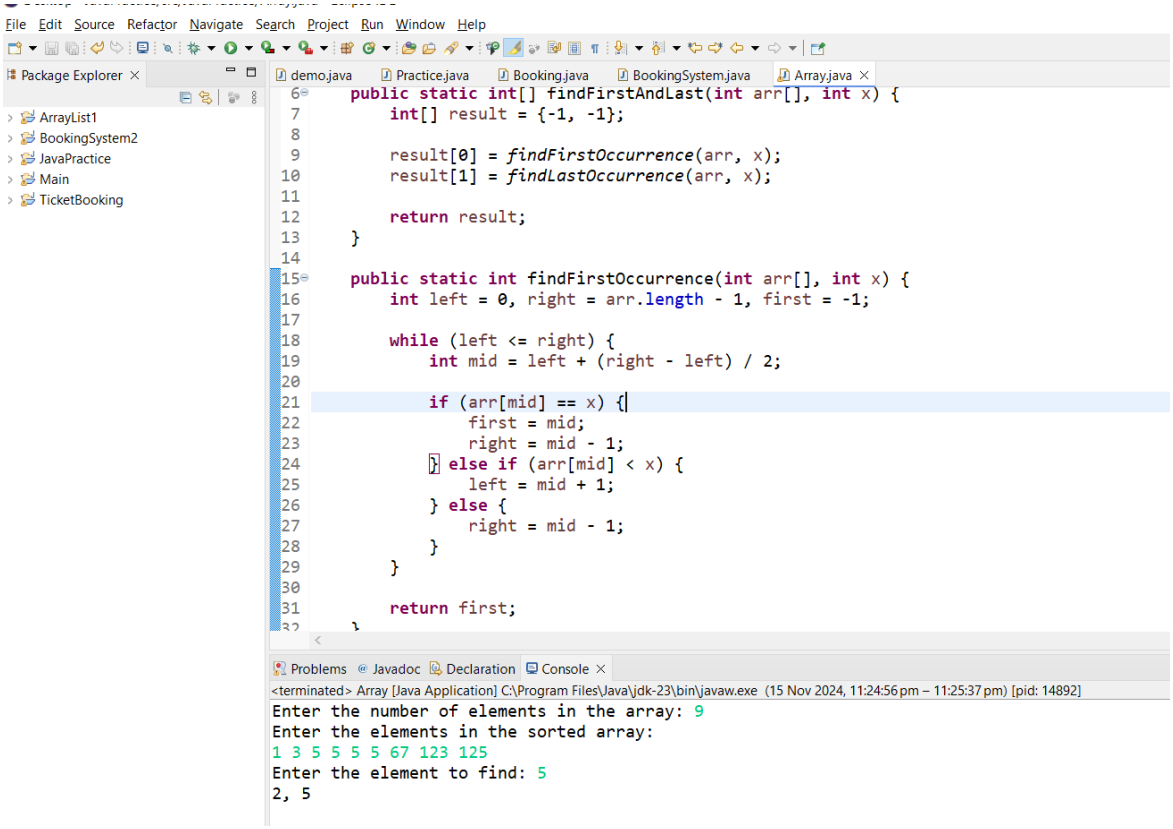
```java
import java.util.Scanner;

public class Array {
    public static int[] findFirstAndLast(int arr[], int x) {
        int[] result = {-1, -1};

        result[0] = findFirstOccurrence(arr, x);
        result[1] = findLastOccurrence(arr, x);

        return result;
    }

    public static int findFirstOccurrence(int arr[], int x) {
        int left = 0, right = arr.length - 1, first = -1;

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

            if (arr[mid] == x) {
                first = mid;
                right = mid - 1;
            } else if (arr[mid] < x) {
                left = mid + 1;
            } else {
                right = mid - 1;
            }
        }
```

```
    return first;
}

public static int findLastOccurrence(int arr[], int x) {
    int left = 0, right = arr.length - 1, last = -1;

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

        if (arr[mid] == x) {
            last = mid;
            left = mid + 1;
        } else if (arr[mid] < x) {
            left = mid + 1;
        } else {
            right = mid - 1;
        }
    }

    return last;
}

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 in the sorted array:
");
        for (int i = 0; i < n; i++) {
            arr[i] = sc.nextInt();
        }

        System.out.print("Enter the element to find: ");
        int x = sc.nextInt();

        int[] result = findFirstAndLast(arr, x);

        System.out.println(result[0] + ", " + result[1]);
    }
}
```

Time Complexity: O(log n)

OUTPUT:



4.Find Transition Point

Given a sorted array, arr[] containing only 0s and 1s, find the transition point, i.e., the first index where 1 was observed, and before that, only 0 was observed. If arr does not have any 1, return -1. If the array does not have any 0, return 0.

Examples:

Input: arr[] = [0, 0, 0, 1, 1]

Output: 3

Explanation: index 3 is the transition point where 1 begins.

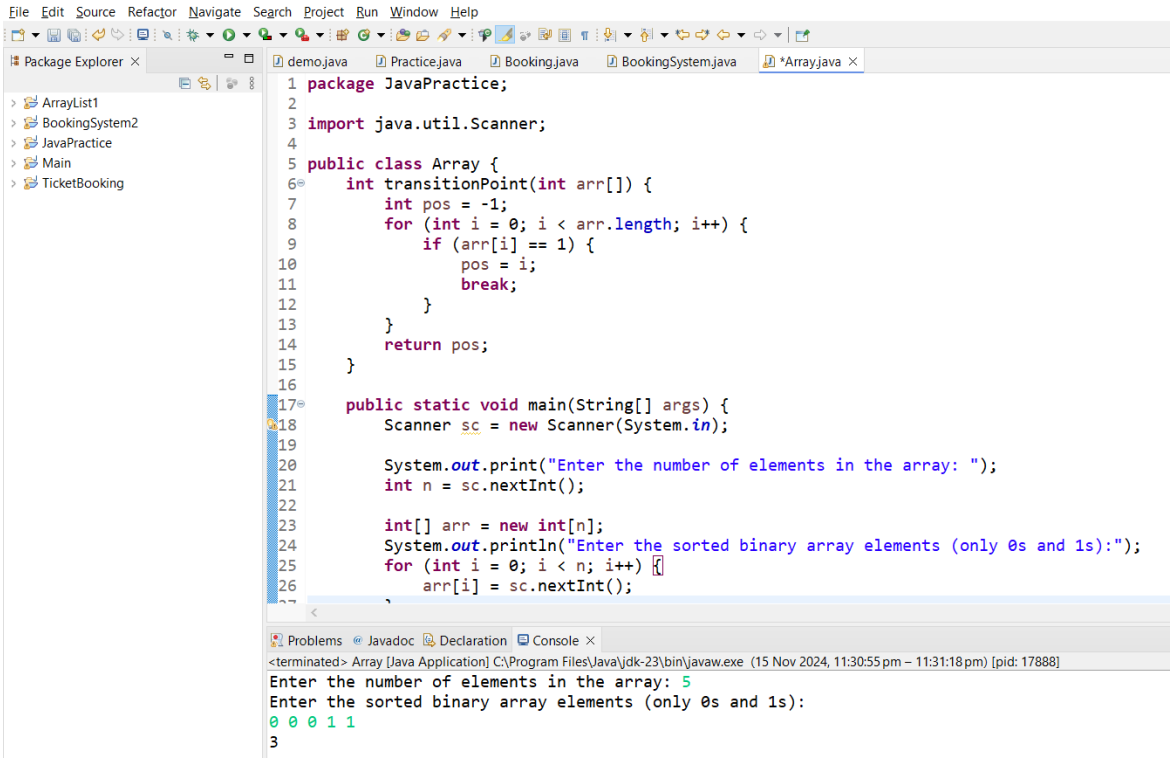
PROGRAM:

```
package JavaPractice;

import java.util.Scanner;

public class Array {
    int transitionPoint(int arr[]) {
        int pos = -1;
```

```
        for (int i = 0; i < arr.length; i++) {
            if (arr[i] == 1) {
                pos = i;
                break;
            }
        }
        return pos;
    }

    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 sorted binary array elements
(only 0s and 1s):");
        for (int i = 0; i < n; i++) {
            arr[i] = sc.nextInt();
        }
        Array sol=new Array();
        int result = sol.transitionPoint(arr);
        System.out.println(result);
    }
}
```

Time Complexity: O(n)

OUTPUT:



6. Remove Duplicates Sorted Array

Given a sorted array arr. Return the size of the modified array which contains only distinct elements.

Note:

1. Don't use set or HashMap to solve the problem.
2. You must return the modified array size only where distinct elements are present and modify the original array such that all the distinct elements come at the beginning of the original array.

Examples :

Input: arr = [2, 2, 2, 2, 2]
Output: [2]
Explanation: After removing all the duplicates only one instance of 2 will remain i.e. [2] so modified array will contains 2 at first position and you should return 1 after modifying the array, the driver code will print the modified array elements.

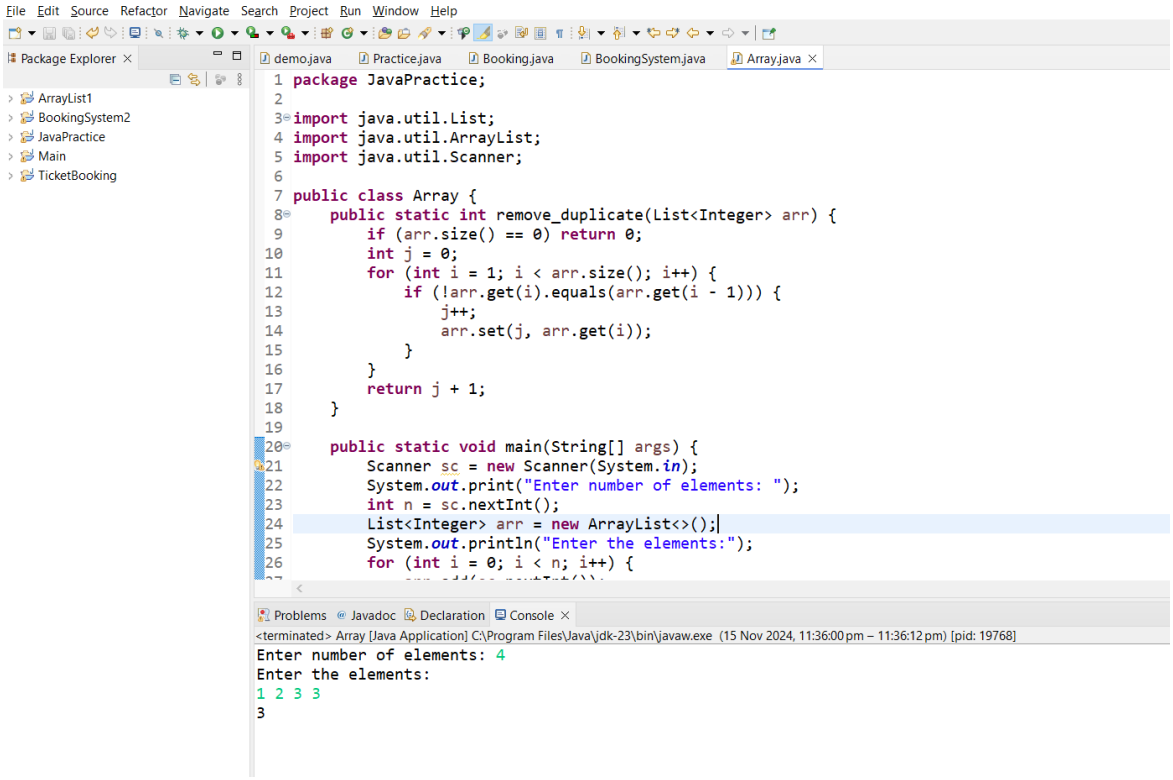
PROGRAM:

```
package JavaPractice;
```

```java
import java.util.List;
import java.util.ArrayList;
import java.util.Scanner;

public class Array {
    public static int remove_duplicate(List<Integer> arr) {
        if (arr.size() == 0) return 0;
        int j = 0;
        for (int i = 1; i < arr.size(); i++) {
            if (!arr.get(i).equals(arr.get(i - 1))) {
                j++;
                arr.set(j, arr.get(i));
            }
        }
        return j + 1;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter number of elements: ");
        int n = sc.nextInt();
        List<Integer> arr = new ArrayList<>();
        System.out.println("Enter the elements:");
        for (int i = 0; i < n; i++) {
            arr.add(sc.nextInt());
        }
        int size = Array.remove_duplicate(arr);
        System.out.println(size);
    }
}
```

Time Complexity: O(n)

OUTPUT:



7. Maximum Index

Given an array of positive integers. The task is to return the maximum of j - i subjected to the constraint of arr[i] < arr[j] and i < j.

Examples:

Input: arr[] = [1, 10]

Output: 1

Explanation: arr[0] < arr[1] so (j-i) is 1-0 = 1.

PROGRAM:

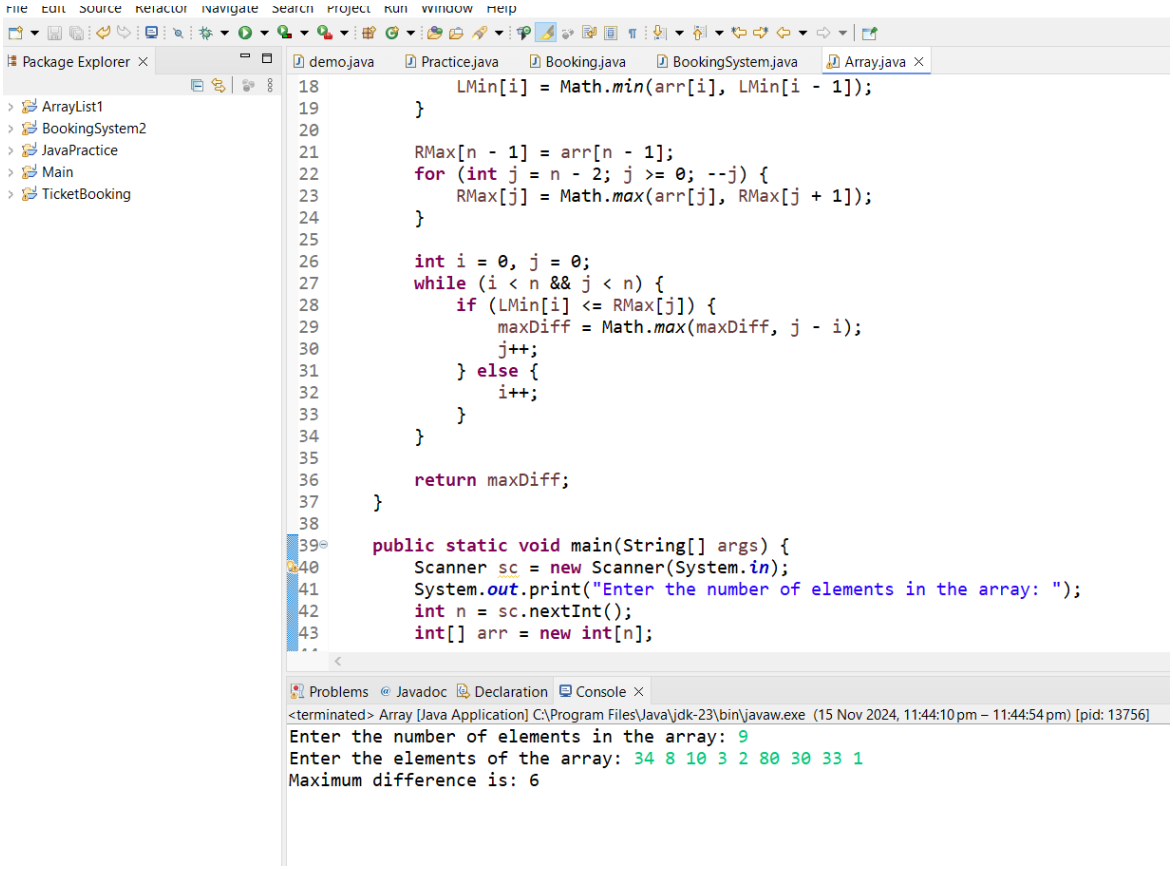
```
package JavaPractice;

import java.util.Scanner;

public class Array {

    int maxIndexDiff(int[] arr) {
```

```
int n = arr.length;
if (n == 1) {
    return 0;
}

int maxDiff = -1;
int[] LMin = new int[n];
int[] RMax = new int[n];

LMin[0] = arr[0];
for (int i = 1; i < n; ++i) {
    LMin[i] = Math.min(arr[i], LMin[i - 1]);
}
RMax[n - 1] = arr[n - 1];
for (int j = n - 2; j >= 0; --j) {
    RMax[j] = Math.max(arr[j], RMax[j + 1]);
}
int i = 0, j = 0;
while (i < n && j < n) {
    if (LMin[i] <= RMax[j]) {
        maxDiff = Math.max(maxDiff, j - i);
        j++;
    } else {
        i++;
    }
```

```
        }
        return maxDiff;
    }
    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.print("Enter the elements of the array: ");
        for (int i = 0; i < n; i++) {
            arr[i] = sc.nextInt();
        }
        Array solution = new Array();
        System.out.println("Maximum difference is: " +
solution.maxIndexDiff(arr));
            }
}
```

Time Complexity: O(n)

OUTPUT:


File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
ArrayList1
BookingSystem2
JavaPractice
Main
TicketBooking
demo.java
Practice.java
Booking.java
BookingSystem.java
Array.java
LMin[i] = Math.min(arr[i], LMin[i - 1]);
}
RMax[n - 1] = arr[n - 1];
for (int j = n - 2; j >= 0; --j) {
RMax[j] = Math.max(arr[j], RMax[j + 1]);
}
int i = 0, j = 0;
while (i < n && j < n) {
if (LMin[i] <= RMax[j]) {
maxDiff = Math.max(maxDiff, j - i);
j++;
} else {
i++;
}
}
return maxDiff;
}
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];
Problems
Javadoc
Declaration
Console
<terminated> Array [Java Application] C:\Program Files\Java\jdk-23\bin\javaw.exe (15 Nov 2024, 11:44:10 pm – 11:44:54 pm) [pid: 13756]
Enter the number of elements in the array: 9
Enter the elements of the array: 34 8 10 3 2 80 30 33 1
Maximum difference is: 6