GATE TECHNICAL TRAINING – DSA CODING PRACTICE PROBLEMS 2026

DATE: 14-11-24 **NAME: Jenifa S Joanna – CSBS**

1. STOCK BUY AND SELL

The cost of stock on each day is given in an array A[] of size N. Find all the segments of days on which you buy and sell the stock such that the sum of difference between sell and buy prices is maximized. Each segment consists of indexes of two elements, first is index of day on which you buy stock and second is index of day on which you sell stock.

```
import java.util.Scanner;
public class BuyandSellStocks {
public static int maximumProfit(int nums[]) {
int lMin=nums[0];
int lMax=nums[0];
int i=0;
int res=0;
int n=nums.length;
while(i<n-1) {
while(i<n-1 && nums[i]>=nums[i+1]) {
i++;
}
lMin = nums[i];
while(i<n-1 && nums[i]<=nums[i+1]) {
i++;
}
lMax = nums[i];
res+=(lMax - lMin);
}
return res;
}
public static void main(String args[]) {
Scanner sc = new Scanner(System.in);
System.out.println("Enter the no. of elements: ");
int n = sc.nextInt();
int nums[] = new int[n];

System.out.println("Enter the elements in the array: ");
for(int i=0; i<n; i++) {
nums[i]= sc.nextInt();
}
System.out.println("The Maximum Profit is " + maximumProfit(nums));
}
}
```

```
Enter the no. of elements:
5
Enter the elements in the array:
4 9 -3 5 6
The Maximum Profit is 14
```

2. COIN CHANGE (COUNT WAYS)

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

```
import java.util.Scanner;
public class CountChange {
static long count(int coins[], int n, int sum) {
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 scanner = new Scanner(System.in);
System.out.print("Enter the number of coins: ");
int n = scanner.nextInt();
int[] coins = new int[n];
System.out.println("Enter the values of the coins: ");
for (int i = 0; i < n; i++) {
coins[i] = scanner.nextInt();
}
System.out.print("Enter the target sum: ");
int sum = scanner.nextInt();
long ways = count(coins, n, sum);
System.out.println("Number of ways to make the sum " + sum + " is: " + ways);
}
}
```

```
Enter the size of the array: 5
Enter the elements of the array:
6 3 6 9 10
Array after removing duplicates:
6 3 9 10
PS D:\Coding\Java Programming\13-11 pr
```

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 anelement x in the given array.

```
import java.util.ArrayList;
import java.util.Scanner;
public class FirstLastOccurance {
public static ArrayList<Integer> findFirstAndLast(int[] arr, int x) {
ArrayList<Integer> result = new ArrayList<>();
int n = arr.length;
int first = -1, last = -1;
for (int i = 0; i < n; i++) {
if (x != arr[i])
```

```
continue;
if (first == -1)
first = i;
last = i;
}
result.add(first);
result.add(last);
return result;
}
public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    System.out.print("Enter the number of elements in the array: ");
    int n = scanner.nextInt();
    int[] arr = new int[n];
    System.out.println("Enter the elements of the array: ");
    for (int i = 0; i < n; i++) {
    arr[i] = scanner.nextInt();
    }
    System.out.print("Enter the value of x to search for: ");
    int x = scanner.nextInt();
    ArrayList<Integer> result = findFirstAndLast(arr, x);
    System.out.println("First occurrence: " + result.get(0));
    System.out.println("Last occurrence: " + result.get(1));
    }
    }
```

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 is observed, and before that, only 0 was observed. If arr does not have any 1, return -1. If array does not have any 0, return 0.

```
import java.util.Scanner;
public class TransitionPoint {
public static int transitionPoint(int arr[]) {
for (int i = 0; i < arr.length; i++) {
if (arr[i] == 1) {
return i;
}
}
return -1;
}
public static void main(String[] args) {
Scanner scanner = new Scanner(System.in);
System.out.print("Enter the number of elements in the array: ");
int n = scanner.nextInt();
int[] arr = new int[n];
System.out.println("Enter the elements of the array:");
for (int i = 0; i < n; i++) {
arr[i] = scanner.nextInt();
}
int result = transitionPoint(arr);
if (result != -1) {
System.out.println("The transition point is at index: " + result);
} else {
System.out.println("There is no transition point.");
}
```

```
}
}
```

```
Enter the number of elements in the array: 5
Enter the elements of the array:
3 6 1 -8 7
The transition point is at index: 2
PS D:\Coding\Java Programming\13-11 practice problem>
```

5. FIRST REPEATING ELEMENT

Given an array arr[], find the first repeating element. The element should occur more than once and the index of its first occurrence should be the smallest.

```
import java.util.HashSet;
import java.util.Scanner;
public class FirstRepeatingElement {

public static int firstRepeated(int[] arr) {
int min = -1;
HashSet<Integer> set = new HashSet<>();
for (int i = arr.length - 1; i >= 0; i--) {
if (set.contains(arr[i])) {
min = i;
} else {
set.add(arr[i]);
}
}
return (min == -1) ? -1 : min + 1;
}
public static void main(String[] args) {
Scanner scanner = new Scanner(System.in);
System.out.print("Enter the number of elements in the array: ");
int n = scanner.nextInt();
int[] arr = new int[n];
System.out.println("Enter the elements of the array:");
for (int i = 0; i < n; i++) {
arr[i] = scanner.nextInt();
}
int result = firstRepeated(arr);
if (result == -1) {
System.out.println("No repeating element found.");
} else {
System.out.println("The position of the first repeating element is: " +
result);
}
scanner.close();
}
}
```

```
Enter the number of elements in the array: 5
Enter the elements of the array:
4 -8 -8 3 7
The position of the first repeating element is: 2
```

6. REMOVES DUPLICATES SORTED ARRAY

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

```
import java.util.HashSet;
import java.util.Scanner;
public class RemoveDuplicates {
public static int remove_duplicate(int[] arr) {
HashSet<Integer> s = new HashSet<>();
int idx = 0;
for (int i = 0; i < arr.length; i++) {
if (!s.contains(arr[i])) {
s.add(arr[i]);
arr[idx++] = arr[i];
}
}
return idx;
}
public static void main(String[] args) {
Scanner scanner = new Scanner(System.in);
System.out.print("Enter the size of the array: ");
int size = scanner.nextInt();
int[] arr = new int[size];
System.out.println("Enter the elements of the array: ");
for (int i = 0; i < size; i++) {
arr[i] = scanner.nextInt();
}

int newLength = remove_duplicate(arr);
System.out.println("Array after removing duplicates:");
for (int i = 0; i < newLength; i++) {
System.out.print(arr[i] + " ");
}
}
}
```

```
Enter the size of the array: 5
Enter the elements of the array:
-1 0 23 32 -1
Array after removing duplicates:
-1 0 23 32
```

7. MAXIMUM INDEX

Given an array arr 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.

```
import java.util.Scanner;
public class FindMaximum {
int max(int x, int y) {
return x > y ? x : y;
}
int min(int x, int y) {
return x < y ? x : y;
}
int maxIndexDiff(int arr[], int n) {
```

```
int maxDiff;
int i, j;
int RMax[] = new int[n];
int LMin[] = new int[n];
LMin[0] = arr[0];
for (i = 1; i < n; ++i)
LMin[i] = min(arr[i], LMin[i - 1]);
RMax[n - 1] = arr[n - 1];
for (j = n - 2; j >= 0; --j)
RMax[j] = max(arr[j], RMax[j + 1]);
i = 0;
j = 0;
maxDiff = -1;
while (j < n && i < n) {
if (LMin[i] <= RMax[j]) {
maxDiff = max(maxDiff, j - i);
j = j + 1;
} else {
i = i + 1;
}
}
return maxDiff;
}
public static void main(String[] args) {
Scanner scanner = new Scanner(System.in);
System.out.print("Enter the number of elements in the array: ");
int n = scanner.nextInt();
int[] arr = new int[n];
System.out.println("Enter the elements of the array: ");
for (int i = 0; i < n; i++) {
arr[i] = scanner.nextInt();
}
FindMaximum max = new FindMaximum();
int maxDiff = max.maxIndexDiff(arr, n);
System.out.println("The maximum index difference is: " + maxDiff);
}
}
```

```
Enter the size of the array: 5
Enter the elements of the array:
-1 0 23 32 -1
Array after removing duplicates:
-1 0 23 32
```

8.Wave Array

Given a sorted array arr[] of distinct integers. Sort the array into a wave-like array(In Place). In other words, arrange the elements into a sequence such that arr[1] >= arr[2] <= arr[3] >= arr[4] <= arr[5].....

If there are multiple solutions, find the lexicographically smallest one.

```
import java.util.Arrays;
import java.util.Scanner;
public class WaveArray {
public static void convertToWave(int[] arr) {
```

```
sorting(arr);
}
private static void swap(int arr[], int a, int b) {
int temp = arr[a];
arr[a] = arr[b];
arr[b] = temp;

}
private static void sorting(int arr[]) {
int n = arr.length;
for (int i = 0; i < n; i += 2) {
if (i > 0 && arr[i - 1] > arr[i]) {
swap(arr, i, i - 1);
}
if (i < n - 1 && arr[i + 1] > arr[i]) {
swap(arr, i, i + 1);
}
}
}
public static void main(String[] args) {
Scanner scanner = new Scanner(System.in);
System.out.print("Enter the number of elements in the array: ");
int n = scanner.nextInt();
int[] arr = new int[n];
System.out.println("Enter the elements of the array: ");
for (int i = 0; i < n; i++) {
arr[i] = scanner.nextInt();
}
convertToWave(arr);
System.out.println("The Waved Array is: "+ Arrays.toString(arr));
}}
```

```
Enter the number of elements in the array: 6
Enter the elements of the array:
-2 -4 12 23 1 6
The Waved Array is: [-2, -4, 23, 1, 12, 6]
```