**14-11-2024**

**CODING PRACTICE -4**

**1.Stock Buy and sell**

import java.util.\*;

class Problem1{

ArrayList<ArrayList<Integer>> stockBuySell(int A[], int n) {

ArrayList<ArrayList<Integer>> result = new ArrayList<>();

int i = 0;

while (i < n - 1) {

while (i < n - 1 && A[i] >= A[i + 1]) {

i++;

}

if (i == n - 1) {

break;

}

int buy = i++;

while (i < n && A[i] >= A[i - 1]) {

i++;

}

int sell = i - 1;

ArrayList<Integer> pair = new ArrayList<>();

pair.add(buy);

pair.add(sell);

result.add(pair);

}

if (result.isEmpty()) {

return new ArrayList<>();

}

return result;

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

int n = sc.nextInt();

int[] prices = new int[n];

for (int i = 0; i < n; i++) {

prices[i] = sc.nextInt();

}

Problem1 solution = new Problem1();

ArrayList<ArrayList<Integer>> result = solution.stockBuySell(prices, n);

if (result.isEmpty()) {

System.out.println("No Profit");

} else {

for (ArrayList<Integer> pair : result) {

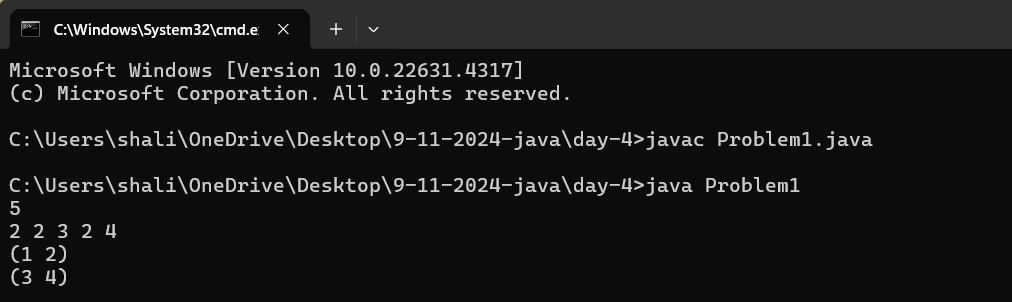
System.out.println("(" + pair.get(0) + " " + pair.get(1) + ")");

}

}

sc.close();

}

}

**2.Coin Change(Count ways)**

import java.util.\*;

class Problem2{

public int count(int coins[], int sum) {

int[] dp = new int[sum + 1];

dp[0] = 1; // There's one way to make sum 0, which is using no coins.

for (int coin : coins) {

for (int i = coin; i <= sum; i++) {

dp[i] += dp[i - coin];

}

}

return dp[sum];

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

// Reading the number of coins

System.out.print("Enter the number of coins: ");

int n = sc.nextInt();

// Reading the coin denominations

int[] coins = new int[n];

System.out.print("Enter the coin denominations: ");

for (int i = 0; i < n; i++) {

coins[i] = sc.nextInt();

}

// Reading the sum

System.out.print("Enter the sum: ");

int sum = sc.nextInt();

Problem2 solution = new Problem2();

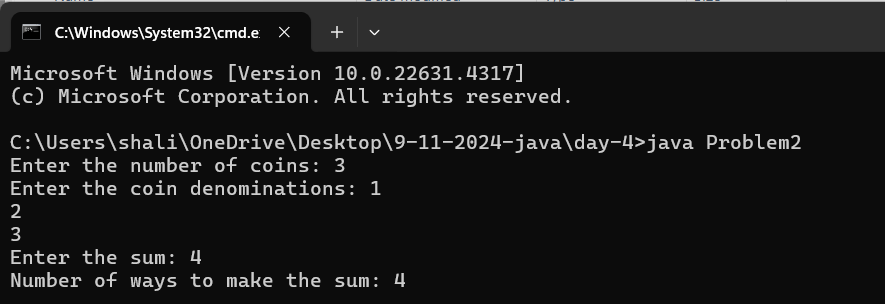
// Output the number of ways to make the sum

System.out.println("Number of ways to make the sum: " + solution.count(coins, sum));

sc.close();

}

}



**3.First and last Occurrence**

import java.util.\*;

class Problem3{

public int[] findFirstAndLast(int arr[], int n, int x) {

int[] result = new int[2];

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

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

return result;

}

private int findFirstOccurrence(int arr[], int n, int x) {

int low = 0, high = n - 1, first = -1;

while (low <= high) {

int mid = low + (high - low) / 2;

if (arr[mid] == x) {

first = mid;

high = mid - 1; // Search in the left half to find the first occurrence

} else if (arr[mid] < x) {

low = mid + 1;

} else {

high = mid - 1;

}

}

return first;

}

private int findLastOccurrence(int arr[], int n, int x) {

int low = 0, high = n - 1, last = -1;

while (low <= high) {

int mid = low + (high - low) / 2;

if (arr[mid] == x) {

last = mid;

low = mid + 1; // Search in the right half to find the last occurrence

} else if (arr[mid] < x) {

low = mid + 1;

} else {

high = mid - 1;

}

}

return last;

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

int n = sc.nextInt();

int[] arr = new int[n];

for (int i = 0; i < n; i++) {

arr[i] = sc.nextInt();

}

int x = sc.nextInt();

Problem3 solution = new Problem3();

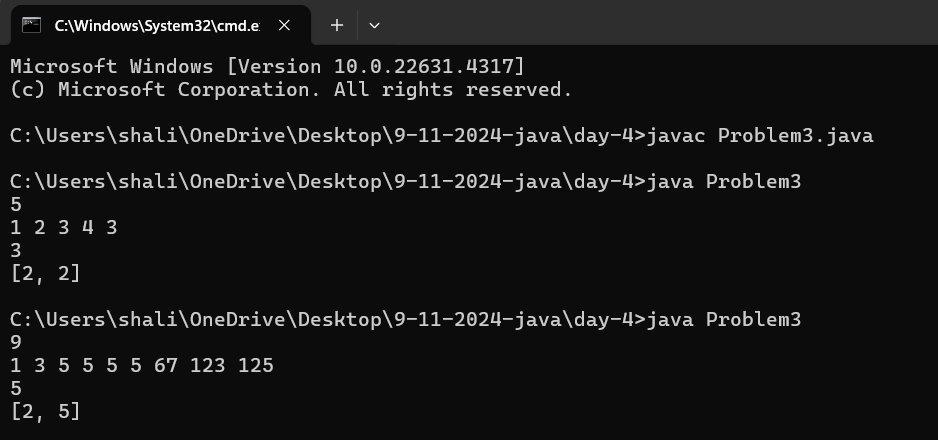
int[] result = solution.findFirstAndLast(arr, n, x);

System.out.println(Arrays.toString(result));

sc.close();

}

}

****

**4.Find Transition Point**

import java.util.Scanner;

class Problem4 {

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 sc = new Scanner(System.in);

int n = sc.nextInt(); // Read the size of the array

int[] arr = new int[n];

for (int i = 0; i < n; i++) {

arr[i] = sc.nextInt(); // Read the elements of the array

}

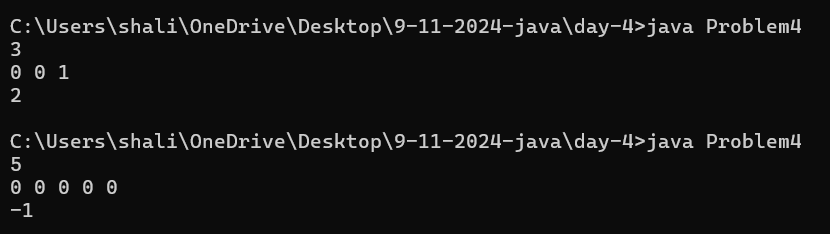
Problem4 solution = new Problem4();

System.out.println(solution.transitionPoint(arr)); // Print the result

sc.close();

}

}



**5.First Repeating element**

import java.util.\*;

class Problem5 {

int firstRepeatingElement(int arr[]) {

Set<Integer> seen = new HashSet<>();

for (int i = 0; i < arr.length; i++) {

if (seen.contains(arr[i])) {

return i + 1;

}

seen.add(arr[i]);

}

return -1;

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

int n = sc.nextInt();

int[] arr = new int[n];

for (int i = 0; i < n; i++) {

arr[i] = sc.nextInt();

}

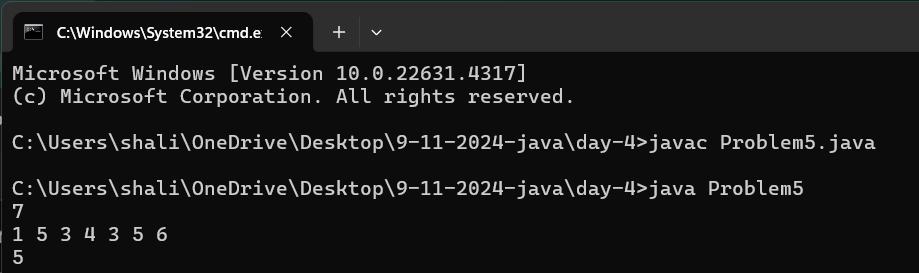
Problem5 solution = new Problem5();

System.out.println(solution.firstRepeatingElement(arr));

sc.close();

}

}



**6.Remove Duplicates Sorted array**

import java.util.\*;

class Problem6 {

int removeDuplicates(int arr[]) {

if (arr.length == 0) {

return 0;

}

int uniqueIndex = 1;

for (int i = 1; i < arr.length; i++) {

if (arr[i] != arr[i - 1]) {

arr[uniqueIndex] = arr[i];

uniqueIndex++;

}

}

return uniqueIndex;

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

int n = sc.nextInt();

int[] arr = new int[n];

for (int i = 0; i < n; i++) {

arr[i] = sc.nextInt();

}

Problem6 solution = new Problem6();

int newSize = solution.removeDuplicates(arr);

System.out.println(newSize);

for (int i = 0; i < newSize; i++) {

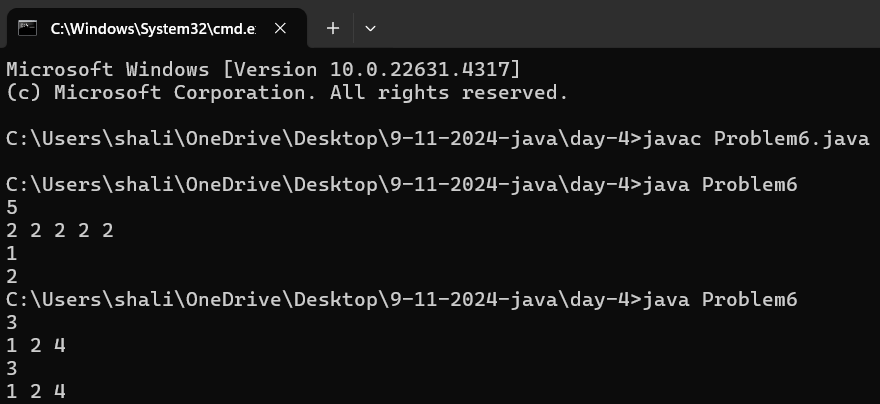
System.out.print(arr[i] + " ");

}

sc.close();

}

}



**7.Maximum Index**

import java.util.\*;

class Problem7 {

int maxIndexDiff(int arr[]) {

int n = arr.length;

int[] leftMin = new int[n];

int[] rightMax = new int[n];

leftMin[0] = arr[0];

for (int i = 1; i < n; i++) {

leftMin[i] = Math.min(arr[i], leftMin[i - 1]);

}

rightMax[n - 1] = arr[n - 1];

for (int i = n - 2; i >= 0; i--) {

rightMax[i] = Math.max(arr[i], rightMax[i + 1]);

}

int i = 0, j = 0, maxDiff = -1;

while (i < n && j < n) {

if (leftMin[i] < rightMax[j]) {

maxDiff = Math.max(maxDiff, j - i);

j++;

} else {

i++;

}

}

return maxDiff;

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

int n = sc.nextInt();

int[] arr = new int[n];

for (int i = 0; i < n; i++) {

arr[i] = sc.nextInt();

}

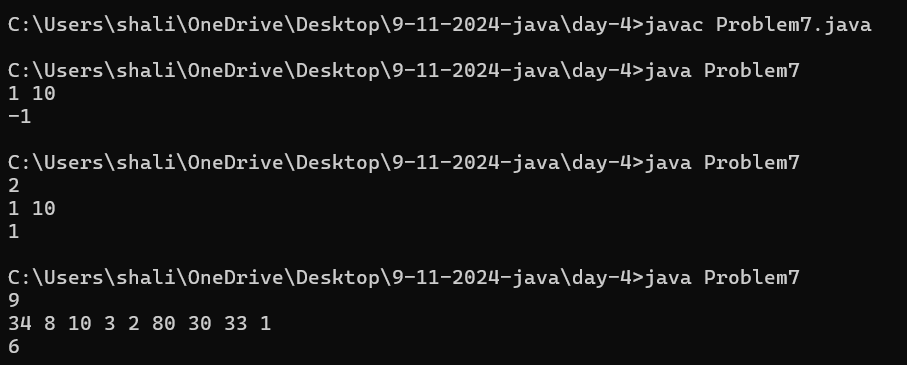
Problem7 solution = new Problem7();

System.out.println(solution.maxIndexDiff(arr));

sc.close();

}

}



**8.Wave array**

import java.util.Scanner;

class Problem8 {

void waveArray(int arr[]) {

int n = arr.length;

// Traverse the array in steps of 2, and swap adjacent elements

for (int i = 0; i < n - 1; i += 2) {

// Swap arr[i] and arr[i+1] to satisfy the condition for wave-like array

if (arr[i] < arr[i + 1]) {

int temp = arr[i];

arr[i] = arr[i + 1];

arr[i + 1] = temp;

}

// If i > 0 and arr[i-1] is smaller than arr[i], swap them again

if (i > 0 && arr[i] < arr[i - 1]) {

int temp = arr[i];

arr[i] = arr[i - 1];

arr[i - 1] = temp;

}

}

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

int n = sc.nextInt();

int[] arr = new int[n];

for (int i = 0; i < n; i++) {

arr[i] = sc.nextInt();

}

Problem8 solution = new Problem8();

solution.waveArray(arr);

// Print the modified array

for (int i = 0; i < n; i++) {

System.out.print(arr[i] + " ");

}

sc.close();

}

}

