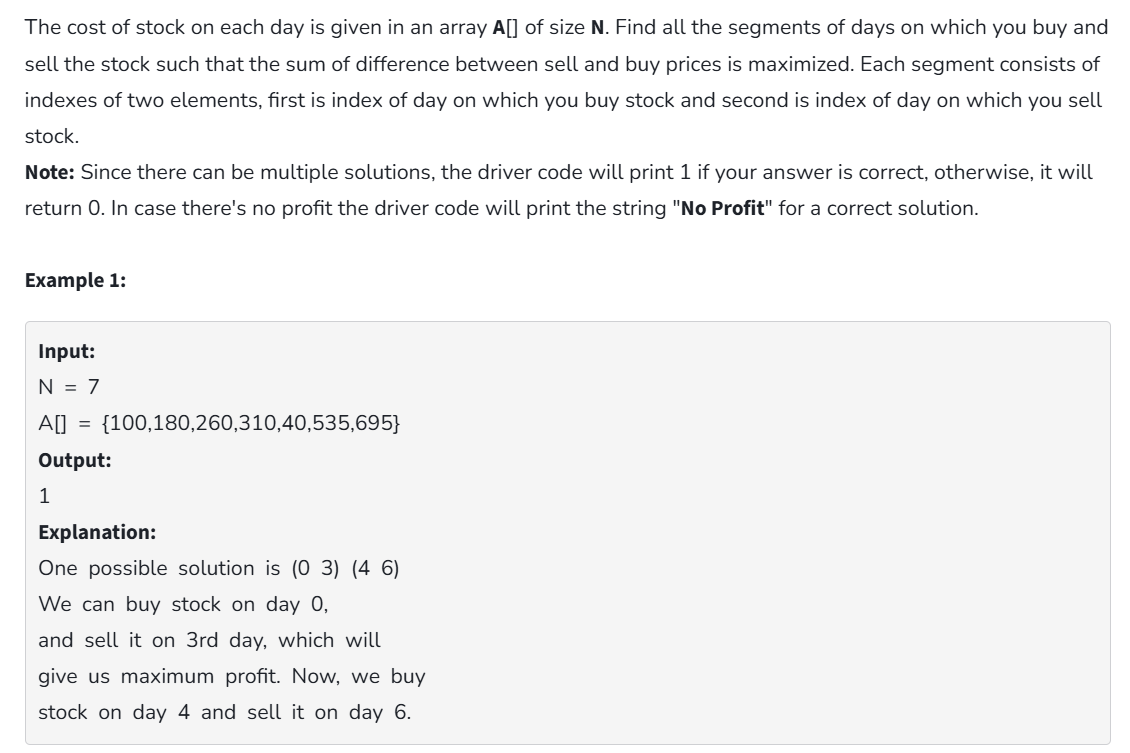
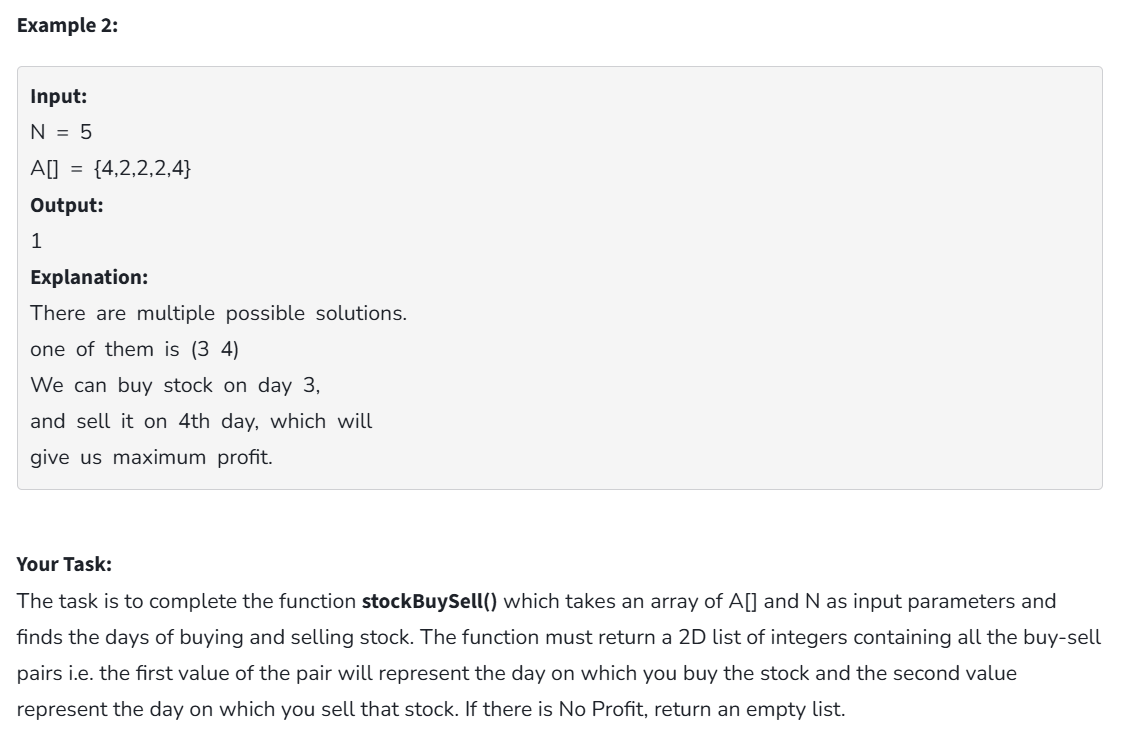
**DSA Practice Test – 5** 14th Nov 2024

**1. Stock Buy and Sell**

****

****

**Code:**

import java.util.ArrayList;

class Solution {

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

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

int i = 0;

while (i < n - 1) {

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

i++;

}

if (i == n - 1) break;

int buy = i++;

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

i++;

}

int sell = i - 1;

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

segment.add(buy);

segment.add(sell);

result.add(segment);

}

return result;

}

public static void main(String[] args) {

Solution solution = new Solution();

int[] prices1 = {100, 180, 260, 310, 40, 535, 695};

int n1 = prices1.length;

System.out.println(solution.stockBuySell(prices1, n1));

int[] prices2 = {4, 2, 2, 2, 4};

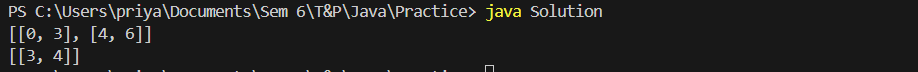
int n2 = prices2.length;

System.out.println(solution.stockBuySell(prices2, n2));

}

}

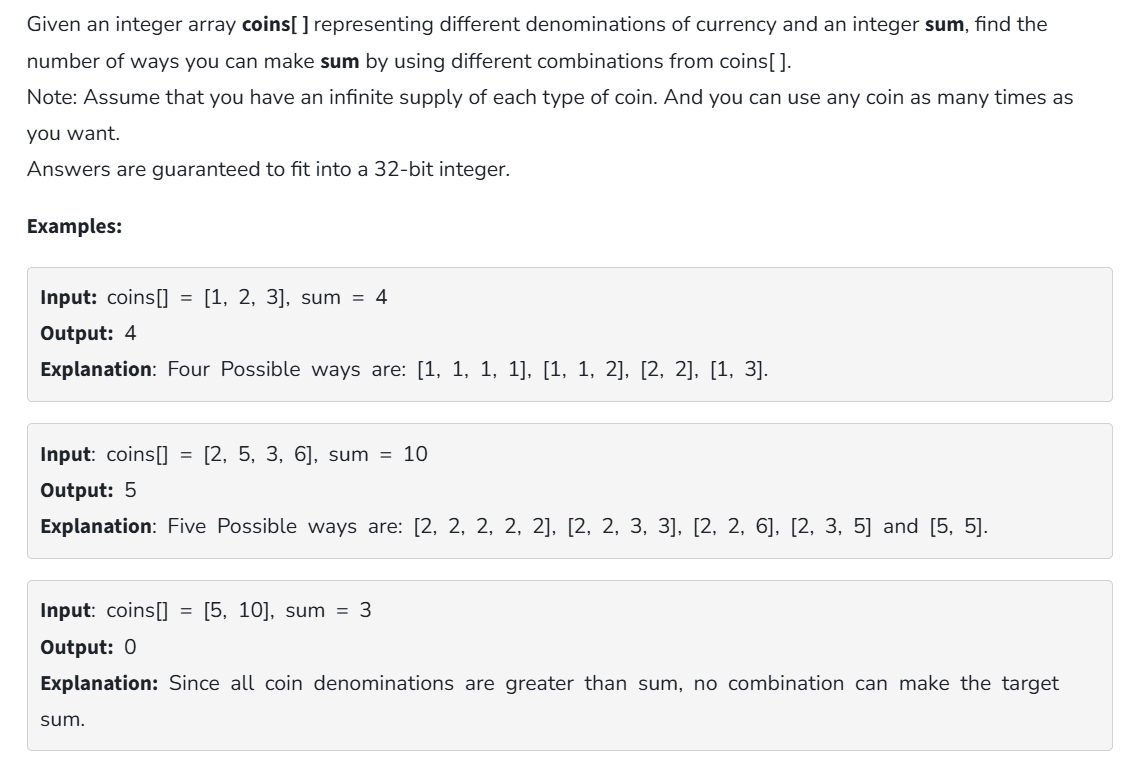
**Output:**



**Time Complexity:** O (n)

**Space Complexity:** O (n)

**2. Coin Change**

****

**Code:**

class Solution {

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

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

dp[0] = 1;

for (int coin : coins) {

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

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

}

}

return dp[sum];

}

public static void main(String[] args) {

Solution solution = new Solution();

int[] coins1 = {1, 2, 3};

int sum1 = 4;

System.out.println(solution.countWaysToMakeSum(coins1, sum1));

int[] coins2 = {2, 5, 3, 6};

int sum2 = 10;

System.out.println(solution.countWaysToMakeSum(coins2, sum2));

int[] coins3 = {5, 10};

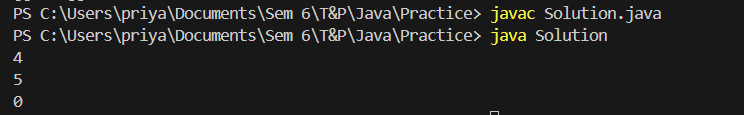
int sum3 = 3;

System.out.println(solution.countWaysToMakeSum(coins3, sum3));

}

}

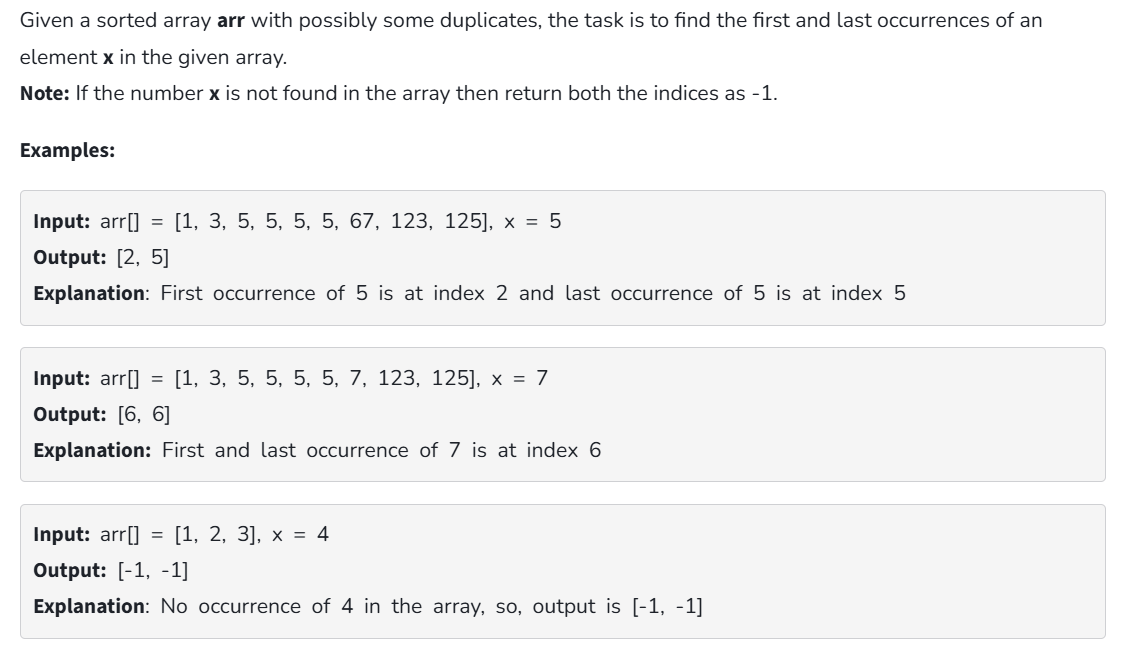
**Output:**



**Time Complexity:** O (n\*sum)

**Space Complexity:** O (sum)

**3. First and Last Occurrences:**



**Code:**

import java.util.ArrayList;

class Solution {

ArrayList<Integer> find(int arr[], int x) {

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

int first = -1, last = -1;

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

if (arr[i] == x) {

if (first == -1) {

first = i;

}

last = i;

}

}

result.add(first);

result.add(last);

return result;

}

public static void main(String[] args) {

Solution gfg = new Solution();

int[] arr = {1, 2, 3, 4, 2, 5, 2};

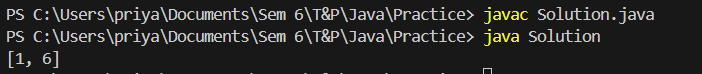
int x = 2;

System.out.println(gfg.find(arr, x));

}

}

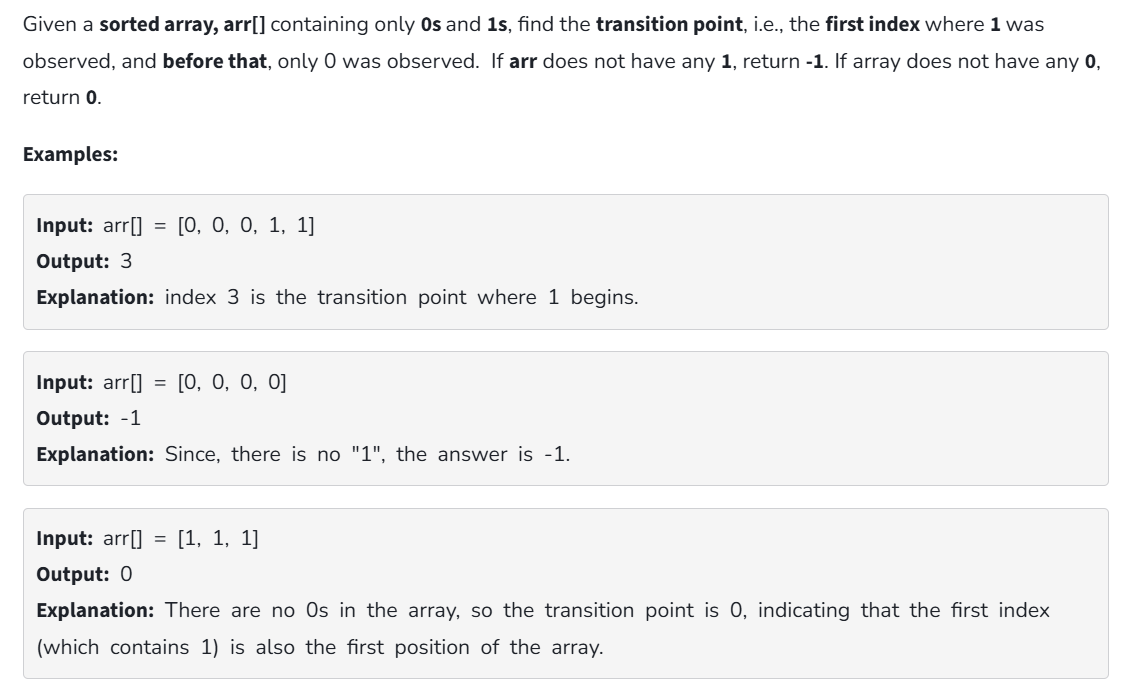
**Output:**



**Time Complexity:** O (n)

**Space Complexity:** O (1)

**4. Find Transition Point**

****

**Code:**

class Solution {

static int transitionPoint(int arr[]) {

int p=-1;

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

if(arr[i]==1) return i;

}

return p;

}

public static void main(String[] args) {

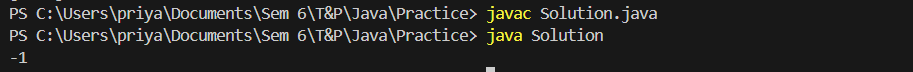
int[] arr = {0,0,0,0};

System.out.println(transitionPoint(arr));

}

}

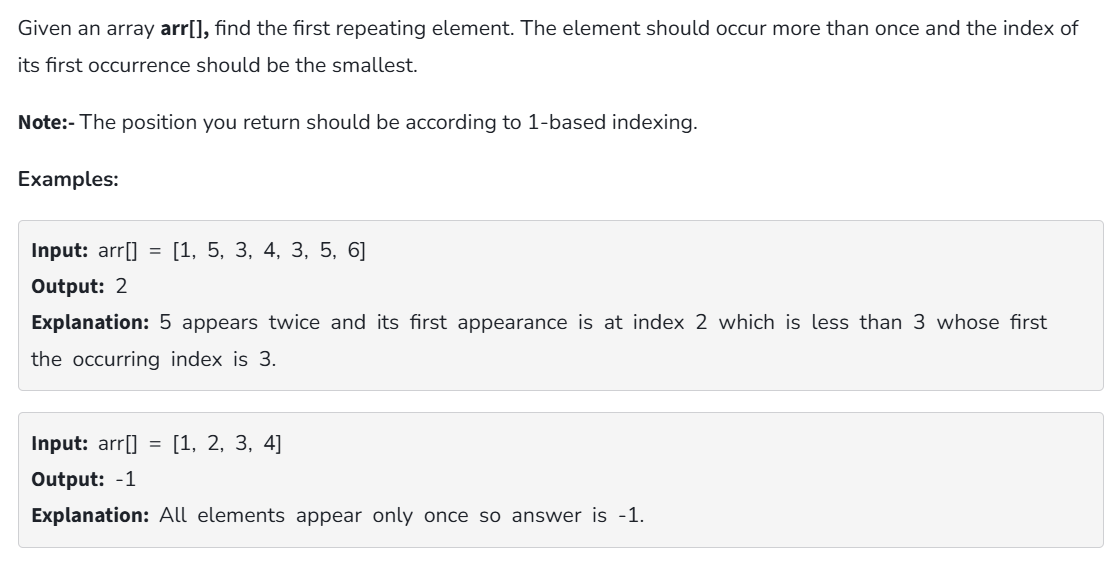
**Output:**



**Time Complexity:** O (n)

**Space Complexity:** O (1)

**5. First Repeating Element**



**Code:**

import java.util.HashMap;

class Solution {

public static int firstRepeated(int[] arr) {

HashMap<Integer, Integer> map = new HashMap<>();

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

map.put(arr[i], map.getOrDefault(arr[i], 0) + 1);

}

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

if (map.get(arr[i]) > 1) {

return i + 1;

}

}

return -1;

}

public static void main(String[] args) {

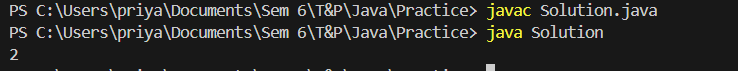
int[] arr = {1, 5, 3, 4, 3, 5, 6};

System.out.println(firstRepeated(arr));

}

}

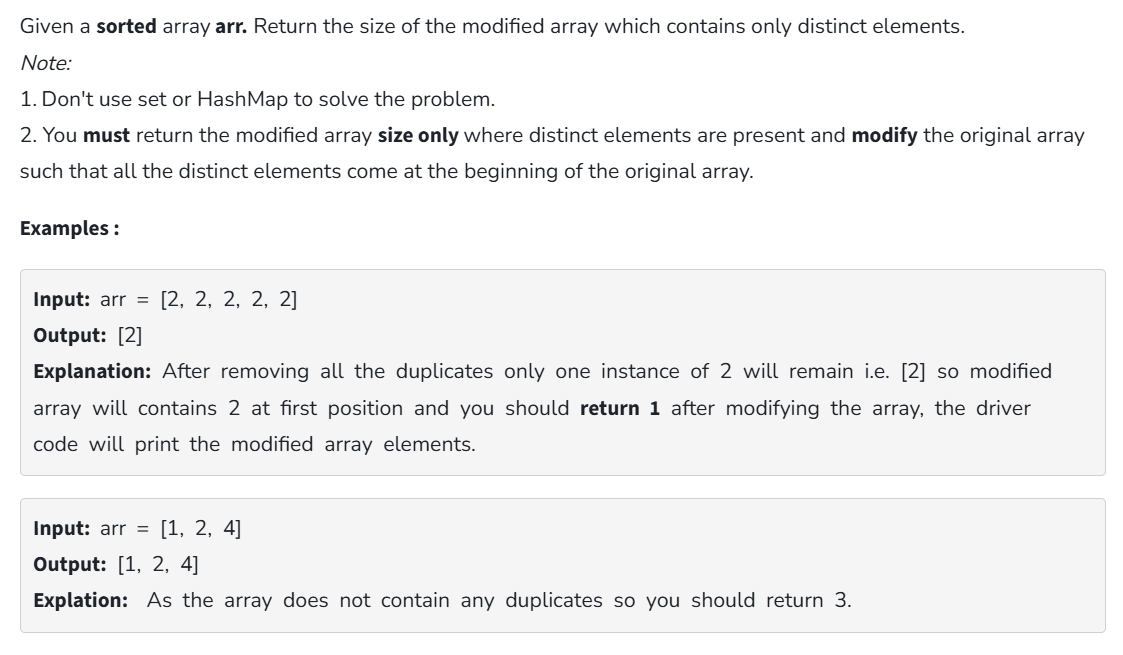
**Output:**



**Time Complexity:** O (n2)

**Space complexity:** O (1)

**6. Remove Duplicates in a Sorted Array**



**Code:**

class Solution {

public int remove\_duplicate(List<Integer> arr) {

if (arr.size() == 0) return 0;

int j = 1;

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

if (!arr.get(i).equals(arr.get(i - 1))) {

arr.set(j, arr.get(i));

j++;

}

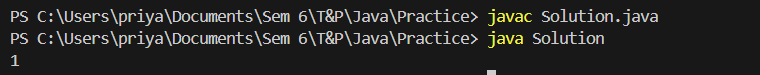
}

return j;

}

}

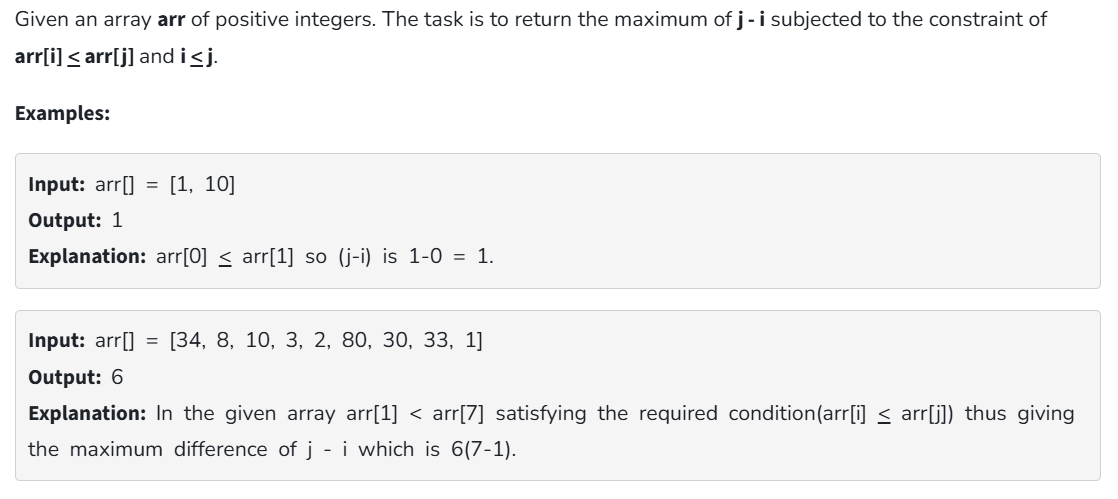
**Output:**



**Time Complexity:** O (n)

**Space Complexity:** O (1)

**7. Maximum Index**



**Code:**

public class Solution {

static int max(int x, int y) {

return x > y ? x : y;

}

static int min(int x, int y) {

return x < y ? x : y;

}

static int maxIndexDiff(int[] arr, int n) {

int[] LMin = new int[n];

int[] RMax = new int[n];

LMin[0] = arr[0];

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

LMin[i] = min(arr[i], LMin[i - 1]);

}

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

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

RMax[j] = max(arr[j], RMax[j + 1]);

}

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

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

if (LMin[i] <= RMax[j]) {

maxDiff = max(maxDiff, j - i);

j++;

} else {

i++;

}

}

return maxDiff;

}

public static void main(String[] args) {

int arr[] = {34, 8, 10, 3, 2, 80, 30, 33, 1};

int n = arr.length;

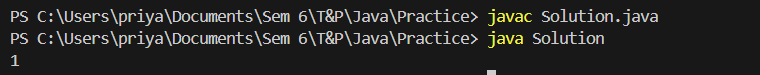
int maxDiff = maxIndexDiff(arr, n);

System.out.println( maxDiff);

}

}

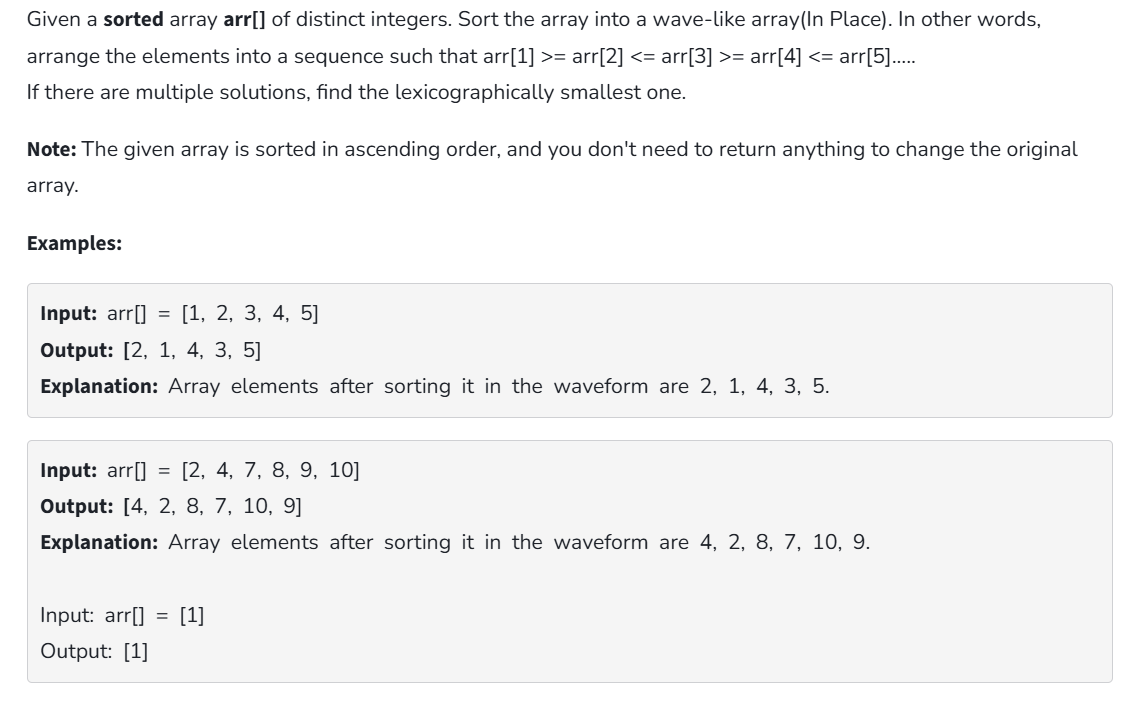
**Output:**



**Time Complexity:** O (n)

**Space Complexity:** O (n)

**8. Wave Array**

****

**Code:**

public class Solution {

public static void convertToWave(int[] arr) {

for (int i = 0; i < arr.length - 1; i += 2) {

int temp = arr[i];

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

arr[i + 1] = temp;

}

}

public static void main(String[] args) {

int[] arr = {1, 2, 3, 4, 5};

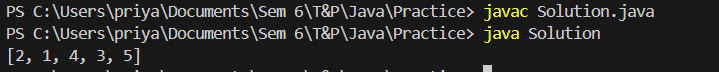
convertToWave(arr);

System.out.println(java.util.Arrays.toString(arr));

}

}

**Output:**



**Time Complexity:** O (n)

**Space Complexity:** O (1)