DSA PRACTICE DAY 5 DHANUSHSHRUTHI S T AI&DS

1. **Remove Duplicates Elements in the List:**

**CODE:**

import java.util.ArrayList;

import java.util.HashSet;

import java.util.List;

public class RemoveDuplicates {

public static List<Integer> removeDuplicates(List<Integer> list) {

return new ArrayList<>(new HashSet<>(list));

}

public static void main(String[] args) {

List<Integer> list = new ArrayList<>();

list.add(1);

list.add(2);

list.add(2);

list.add(3);

list.add(3);

list.add(4);

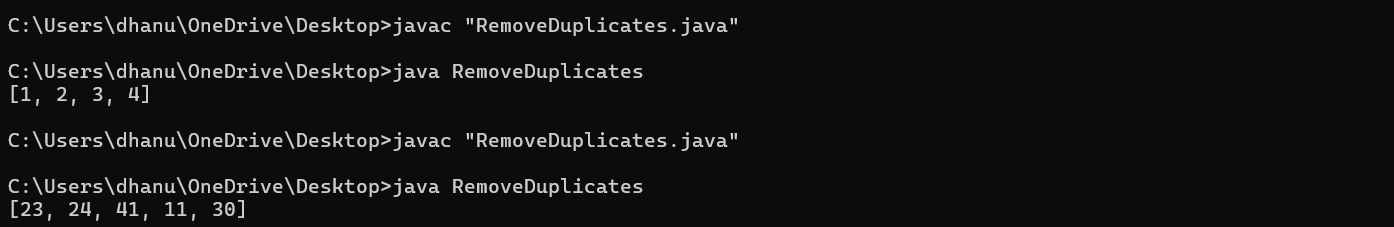
List<Integer> uniqueList = removeDuplicates(list);

System.out.println(uniqueList);

}

}

**OUTPUT:**



**TIME COMPLEXITY: O(n)**

1. **First Repeating Element:**

**CODE:**

import java.util.HashSet;

public class FirstRepeatingElement {

public static int findFirstRepeating(int[] arr) {

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

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

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

return arr[i];

}

seen.add(arr[i]);

}

return -1; // Return -1 if no repeating element is found

}

public static void main(String[] args) {

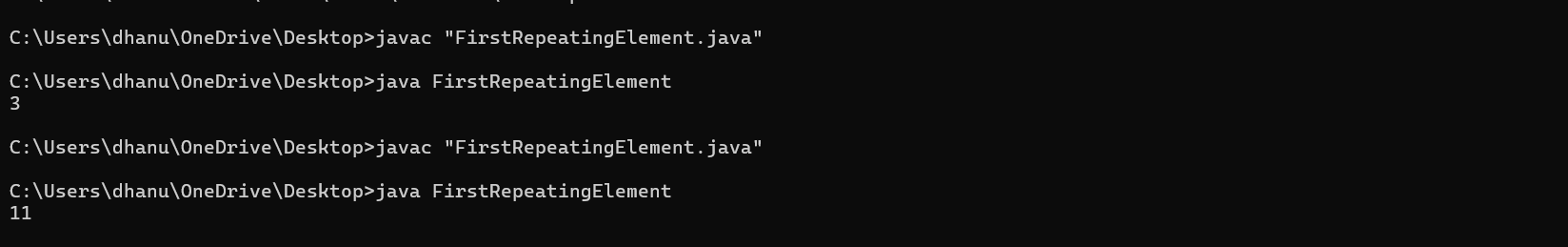
int[] arr = {10, 5, 3, 4, 3, 5, 6};

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

}

}

**OUTPUT:**



**TIME COMPLEXITY: O(n)**

1. **Find Transition Point:**

**CODE:**

public class TransitionPoint {

public static int findTransitionPoint(int[] arr) {

int left = 0, right = arr.length - 1;

int transitionPoint = -1;

while (left <= right) {

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

if (arr[mid] == 1) {

transitionPoint = mid;

right = mid - 1;

} else {

left = mid + 1;

}

}

return transitionPoint;

}

public static void main(String[] args) {

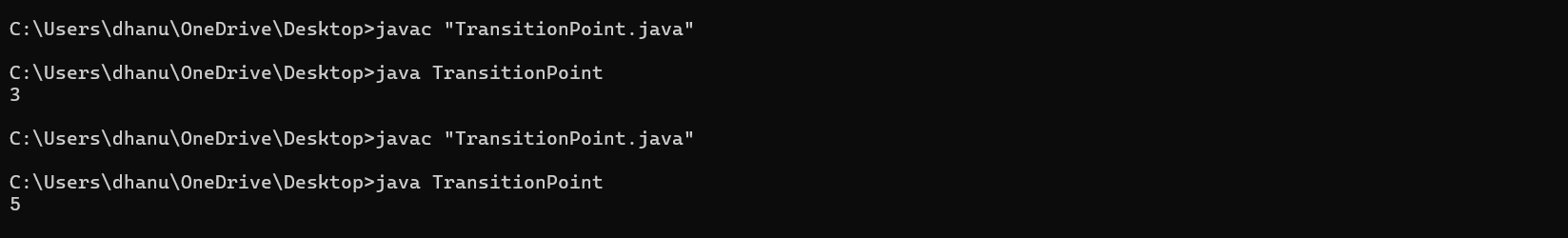
int[] arr = {0, 0, 0, 1, 1, 1};

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

}

}

**OUTPUT:**



**TIME COMPLEXITY: O(log n)**

1. **Wave Array:**

**CODE:**

import java.util.Arrays;

public class WaveArray {

public static void convertToWave(int[] arr) {

Arrays.sort(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 = {10, 5, 6, 3, 2, 20, 100, 80};

convertToWave(arr);

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

}

}

**OUTPUT:**



**TIME COMPLEXITY: O (n log n)**

1. **Stock buy and Sell:**

**CODE:**

public class StockBuySell {

public static int maxProfit(int[] prices) {

int profit = 0;

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

if (prices[i] > prices[i - 1]) {

profit += prices[i] - prices[i - 1];

}

}

return profit;

}

public static void main(String[] args) {

int[] prices = {7, 1, 5, 3, 6, 4};

System.out.println(maxProfit(prices));

}

}

**OUTPUT:**



**TIME COMPLEXITY: O(n)**

1. **Coin Change:**

**CODE:**

public class CoinChange {

public static int countWays(int[] coins, int amount) {

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

dp[0] = 1;

for (int coin : coins) {

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

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

}

}

return dp[amount];

}

public static void main(String[] args) {

int[] coins = {1, 2, 5};

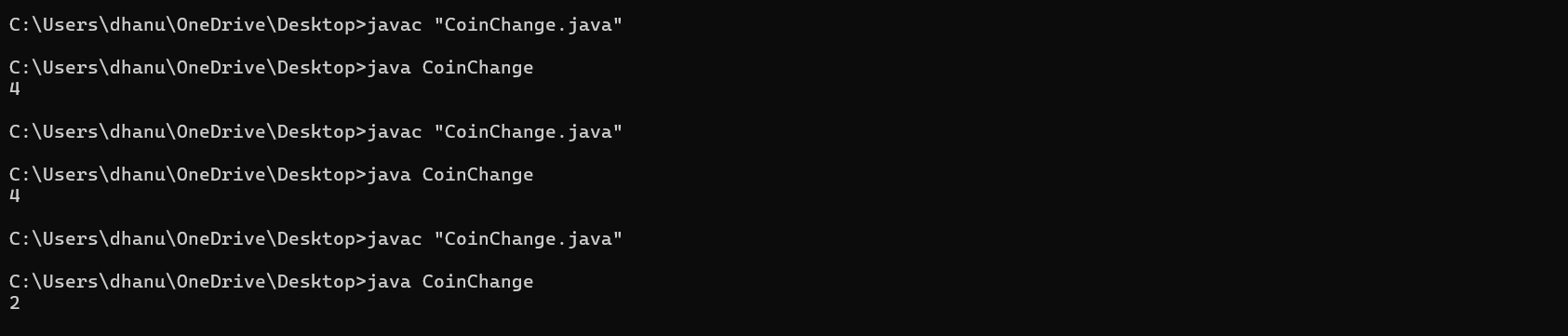
int amount = 5;

System.out.println(countWays(coins, amount));

}

}

**OUTPUT:**



**TIME COMPLEXITY:O(n\*m)**

1. **First and Last Occurences:**

**CODE:**

public class FirstLastOccurrence {

public static int findFirstOccurrence(int[] arr, int target) {

int left = 0, right = arr.length - 1, result = -1;

while (left <= right) {

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

if (arr[mid] == target) {

result = mid;

right = mid - 1;

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

left = mid + 1;

} else {

right = mid - 1;

}

}

return result;

}

public static int findLastOccurrence(int[] arr, int target) {

int left = 0, right = arr.length - 1, result = -1;

while (left <= right) {

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

if (arr[mid] == target) {

result = mid;

left = mid + 1;

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

left = mid + 1;

} else {

right = mid - 1;

}

}

return result;

}

public static void main(String[] args) {

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

int target = 2;

int first = findFirstOccurrence(arr, target);

int last = findLastOccurrence(arr, target);

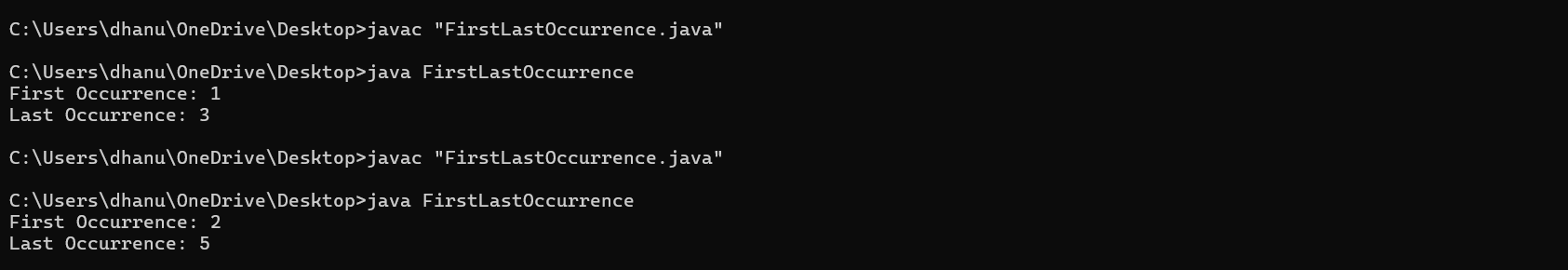
System.out.println("First Occurrence: " + first);

System.out.println("Last Occurrence: " + last);

}

}

**OUTPUT:**



**TIME COMPLEXITY:O(log n)**

1. **Maximum Index:**

**CODE:**

public class MaximumIndex {

public static 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(leftMin[i - 1], arr[i]);

}

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

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

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

}

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) {

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

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

}

}

**OUTPUT:**



**TIME COMPELXITY: O(n)**