**LAVANYAA A AIML**

1. **First and Last Occurrence**

import java.util.Arrays;

public class FirstAndLastOccurrence {

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

int first = -1, last = -1;

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

if (arr[i] == target) {

if (first == -1) first = i;

last = i;}

}

return new int[]{first, last};

}

public static void main(String[] args) {

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

int target = 2;

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

System.out.println("First Occurrence: " + result[0] + ", Last Occurrence: " + result[1]);

}

}

TIME COMPLEXITY: O(n)

SPACE COMPLEXITY: O(1)

1. **Find transition point**

public class TransitionPoint {

public static int findTransitionPoint(int[] arr) {

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

if (arr[i] == 1) {

return i;

}

}

return -1;

}

public static void main(String[] args) {

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

System.out.println("Transition Point: " + findTransitionPoint(arr));

}

}

TIME COMPLEXITY: O(n)

SPACE COMPLEXITY: O(1)

1. **First Repeating Element**

import java.util.HashSet;

public class FirstRepeatingElement {

public static int firstRepeating(int[] arr) {

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

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

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

return arr[i];

}

set.add(arr[i]);

}

return -1;

}

public static void main(String[] args) {

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

System.out.println("First Repeating Element: " + firstRepeating(arr));

}

}

TIME COMPLEXITY: O(n)

SPACE COMPLEXITY: O(n)

1. **Remove Duplicates from Sorted Array**

public class RemoveDuplicates {

public static int removeDuplicates(int[] arr) {

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

int index = 1;

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

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

arr[index++] = arr[i];

}

}

return index;

}

public static void main(String[] args) {

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

int newLength = removeDuplicates(arr);

System.out.println("Array after removing duplicates: ");

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

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

}

}

}

TIME COMPLEXITY: O(n)

SPACE COMPLEXITY: O(1)

1. **Maximum index**

public class MaximumIndex {

public static int findMaxIndexDifference(int[] arr) {

int maxDiff = -1;

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

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

if (arr[i] <= arr[j]) {

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

}

}

}

return maxDiff;

}

public static void main(String[] args) {

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

System.out.println("Maximum Index Difference: " + findMaxIndexDifference(arr));

}

}

TIME COMPLEXITY: O(n2)

SPACE COMPLEXITY: O(1)

1. **Wave Array**

import java.util.Arrays;

public class WaveArray {

public static void waveArray(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 = {1, 2, 3, 4, 5, 6};

waveArray(arr);

System.out.println("Wave Array: " + Arrays.toString(arr)); }

}

TIME COMPLEXITY: O(n log n)

SPACE COMPLEXITY: O(n)

1. **Coin change**

import java.util.Arrays;

public class CoinChange {

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

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

Arrays.fill(dp, amount + 1);

dp[0] = 0;

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

for (int coin : coins) {

if (i - coin >= 0) {

dp[i] = Math.min(dp[i], dp[i - coin] + 1);

}

}

}

return dp[amount] > amount ? -1 : dp[amount]; }

public static void main(String[] args) {

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

int amount = 11;

System.out.println("Minimum coins needed: " + coinChange(coins, amount));

}

}

1. Stock buy and sell

public class StockBuyAndSell {

public static int maxProfit(int[] prices) {

int maxProfit = 0;

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

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

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

}

}

return maxProfit;

}

public static void main(String[] args) {

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

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

}

}

TIME COMPLEXITY: O(n)

SPACE COMPLEXITY: O(1)