**Home Work Day 1**

1. **Implement Quick Sort**

class Solution {

public int[] sortArray(int[] nums) {

if (nums == null || nums.length <= 1) {

return nums;

}

mergeSort(nums, 0, nums.length - 1);

return nums;

}

private void mergeSort(int[] nums, int low, int high) {

if (low < high) {

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

mergeSort(nums, low, mid);

mergeSort(nums, mid + 1, high);

merge(nums, low, mid, high);

}

}

private void merge(int[] nums, int low, int mid, int high) {

int n1 = mid - low + 1;

int n2 = high - mid;

int[] left = new int[n1];

int[] right = new int[n2];

for (int i = 0; i < n1; ++i) {

left[i] = nums[low + i];

}

for (int j = 0; j < n2; ++j) {

right[j] = nums[mid + 1 + j];

}

int i = 0, j = 0, k = low;

while (i < n1 && j < n2) {

if (left[i] <= right[j]) {

nums[k++] = left[i++];

} else {

nums[k++] = right[j++];

}

}

while (i < n1) {

nums[k++] = left[i++];

}

while (j < n2) {

nums[k++] = right[j++];

}

}

}

**2. Implement Merge Sort**

/\*\*

\* Definition for singly-linked list.

\* public class ListNode {

\* int val;

\* ListNode next;

\* ListNode() {}

\* ListNode(int val) { this.val = val; }

\* ListNode(int val, ListNode next) { this.val = val; this.next = next; }

\* }

\*/

class Solution {

public ListNode sortList(ListNode head) {

if (head == null || head.next == null) {

return head;

}

ListNode middle = findMiddle(head);

ListNode nextToMiddle = middle.next;

middle.next = null;

ListNode left = sortList(head);

ListNode right = sortList(nextToMiddle);

return merge(left, right);

}

private ListNode findMiddle(ListNode head) {

if (head == null) {

return null;

}

ListNode slow = head;

ListNode fast = head.next;

while (fast != null && fast.next != null) {

slow = slow.next;

fast = fast.next.next;

}

return slow;

}

private ListNode merge(ListNode left, ListNode right) {

ListNode dummy = new ListNode(0);

ListNode current = dummy;

while (left != null && right != null) {

if (left.val < right.val) {

current.next = left;

left = left.next;

} else {

current.next = right;

right = right.next;

}

current = current.next;

}

if (left != null) {

current.next = left;

}

if (right != null) {

current.next = right;

}

return dummy.next;

}

}

1. **Implement Binary Search (Asked in Ninjacart)**

class Solution {

public int search(int[] nums, int target) {

int left = 0;

int right = nums.length - 1;

while (left <= right) {

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

if (nums[mid] == target) {

return mid;

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

left = mid + 1;

} else {

right = mid - 1;

}

}

return -1;

}

}

1. **Move All Zeros to end of Array (Asked in Ninjacart)**

class Solution {

void pushZerosToEnd(int[] arr, int n) {

int nonZeroIndex = 0;

// Traverse the array and move non-zero elements to the front

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

if (arr[i] != 0) {

arr[nonZeroIndex++] = arr[i];

}

}

// Fill the remaining positions with zeros

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

arr[i] = 0;

}

}

}

1. **Longest Palindromic Subsequence (Asked in Ninjacart)**

class Solution {

public int longestPalindromeSubseq(String s) {

int n = s.length();

// Create a 2D array to store the length of the longest palindromic subsequence

int[][] dp = new int[n][n];

// Every individual character is a palindrome of length 1

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

dp[i][i] = 1;

}

// Loop to fill the dp array

for (int len = 2; len <= n; len++) {

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

int j = i + len - 1;

if (s.charAt(i) == s.charAt(j)) {

dp[i][j] = 2 + dp[i + 1][j - 1];

} else {

dp[i][j] = Math.max(dp[i + 1][j], dp[i][j - 1]);

}

}

}

// The length of the longest palindromic subsequence in the entire string

return dp[0][n - 1];

}

}

1. **Two Sum**

class Solution {

public int maxProfit(int k, int[] prices) {

int n = prices.length;

// If k is large enough, it becomes a general stock transaction problem

if (k >= n / 2) {

int maxProfit = 0;

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

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

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

}

}

return maxProfit;

}

// Use dynamic programming to solve the problem

int[][] buy = new int[n][k + 1];

int[][] sell = new int[n][k + 1];

for (int j = 1; j <= k; j++) {

buy[0][j] = -prices[0];

sell[0][j] = 0;

}

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

buy[i][0] = Math.max(buy[i - 1][0], sell[i - 1][0] - prices[i]);

for (int j = 1; j <= k; j++) {

buy[i][j] = Math.max(buy[i - 1][j], sell[i - 1][j] - prices[i]);

sell[i][j] = Math.max(sell[i - 1][j], buy[i - 1][j - 1] + prices[i]);

}

}

return sell[n - 1][k];

}

}

**Home Work Day 2:**

1. **Big Countries**

SELECT name, population, area

FROM World

WHERE area >= 3000000 OR population >= 25000000;

1. **Swap Salary**

UPDATE Salary

SET sex = CASE

WHEN sex = 'm' THEN 'f'

WHEN sex = 'f' THEN 'm'

END;

1. **Not Boring Movies**

SELECT id, movie, description, rating

FROM Cinema

WHERE id % 2 = 1 AND description != 'boring'

ORDER BY rating DESC;

1. **Duplicate Emails**

SELECT email

FROM Person

GROUP BY email

HAVING COUNT(email) > 1;

1. **Combine Two Tables**

SELECT

p.firstName,

p.lastName,

a.city,

a.state

FROM Person p

LEFT JOIN Address a ON p.personId = a.personId;

1. **Emp earning more than their Managers**

SELECT e.name AS Employee

FROM Employee e

JOIN Employee m ON e.managerId = m.id

WHERE e.salary > m.salary;

1. **Customers who never order**

SELECT c.name AS Customers

FROM Customers c

LEFT JOIN Orders o ON c.id = o.customerId

WHERE o.id IS NULL OR o.customerId IS NULL;

1. **Delete Duplicate Emails**

DELETE p1

FROM Person p1, Person p2

WHERE p1.email = p2.email AND p1.id > p2.id;

1. **Rising Temperature**

SELECT id

FROM (

SELECT id,

LAG(temperature) OVER (ORDER BY recordDate) AS prev\_temperature

FROM Weather

) AS temp\_info

WHERE prev\_temperature IS NOT NULL AND temperature > prev\_temperature;

1. **Class More than 5 Students**

SELECT class

FROM Courses

GROUP BY class

HAVING COUNT(DISTINCT student) >= 5;

1. **Second Highest Salary**

SELECT

MAX(salary) AS SecondHighestSalary

FROM Employee

WHERE salary < (SELECT MAX(salary) FROM Employee);

**Home Work Day 3:**

**SQL :**

1.

SELECT e1.name

FROM Employee e1

JOIN Employee e2 ON e1.id = e2.managerId

GROUP BY e1.id, e1.name

HAVING COUNT(e2.id) >= 5;

2.

SELECT DISTINCT c.customer\_id

FROM Customer c

WHERE NOT EXISTS (

SELECT p.product\_key

FROM Product p

WHERE NOT EXISTS (

SELECT 1

FROM Customer c1

WHERE c1.customer\_id = c.customer\_id

AND c1.product\_key = p.product\_key

)

);

3.

SELECT us.product\_id,

ROUND(SUM(price \* units) / SUM(units), 2) AS average\_price

FROM UnitsSold us

JOIN Prices p ON us.product\_id = p.product\_id

AND us.purchase\_date BETWEEN p.start\_date AND p.end\_date

GROUP BY us.product\_id;

**Questions :**

import java.util.Scanner;

public class Main {

public static int PassingGame(int N, int[] A) {

int result = -404;

int[] playerEnergy = new int[N];

System.arraycopy(A, 0, playerEnergy, 0, N);

int currentPlayer = 0;

int duration = 0;

while (true) {

if (playerEnergy[currentPlayer] > 0) {

int nextPlayer = (currentPlayer + 1) % N;

playerEnergy[currentPlayer] -= 1;

currentPlayer = nextPlayer;

duration += 1;

} else {

break;

}

}

result = duration;

return result;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

int N = scanner.nextInt();

// Input: Array of player energies

int[] A = new int[N];

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

A[i] = scanner.nextInt();

}

// Output: Maximum duration of the game

int output = PassingGame(N, A);

System.out.println(output);

}

}

**2. Sort Array**

import java.util.Scanner;

public class SortArray {

public static int sortArray(int N, int K, int[] P) {

int result = -404; // Default value

// WRITE YOUR LOGIC HERE

int operations = 0;

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

if (P[i] != i + 1) {

// Find the index of the correct element

int correctIndex = -1;

for (int j = i; j < N; j++) {

if (P[j] == i + 1) {

correctIndex = j;

break;

}

}

// Check if the segment length is within K

if (correctIndex - i + 1 > K) {

operations++;

i = correctIndex - K + 1; // Move to the next segment

} else {

// Sort the segment

for (int j = correctIndex; j > i; j--) {

int temp = P[j];

P[j] = P[j - 1];

P[j - 1] = temp;

}

operations++;

}

}

}

result = operations;

return result;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

int N = scanner.nextInt();

int K = scanner.nextInt();

int[] P = new int[N];

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

P[i] = scanner.nextInt();

}

int output = sortArray(N, K, P);

System.out.println(output);

}

}

**3. Sort the Array**

import java.util.Scanner;

public class SortArray {

public static int minOperationsToSort(int N, int[] A, String S) {

int result = -404; // Default value

// WRITE YOUR LOGIC HERE

int operations = 0;

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

if (S.charAt(i) != S.charAt(i + 1)) {

// If the magic bits are different, we can perform one operation

operations++;

i++; // Skip the next element as it is part of the subarray

}

}

result = operations;

return result;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

int N = scanner.nextInt();

int[] A = new int[N];

String S = scanner.next();

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

A[i] = scanner.nextInt();

}

int output = minOperationsToSort(N, A, S);

System.out.println(output);

}

}