[문제링크]

2017 10회 EPPER 2번

2. (10점) "OOXXOXXOOO"와 같은 OX퀴즈의 결과가 있다. O는 문제를 맞은 것이고, X는 문제를 틀린 것이다. 문제를 맞은 경우 그 문제의 점수는 그 문제까지 연속된 O의 개수가 된다. OX퀴즈의 결과가 주어졌을 때, 점수를 구하는 프로그램을 작성하시오.

예를 들어, "OOXXOXXOOO"의 점수는
1+2+0+0+1+0+0+1+2+3 = 10점이다.

[입력 형식]
- 문제에 대한 OX 결과를 입력한다.
  (문제의 수는 1000개를 넘지 않는다.)

[출력 형식]
- 점수를 출력한다.

[입력 예1]	[출력 예1]
OOXXOXXOOOO	14

[입력 예2]	[출력 예2]
OOXXOXXOOOXOXOXO	13

키워드: 연속된

2017 10회 EPPER 2번

▶ 문제 분석

난이도 하의 문제입니다. 난이도가 어려운 문제가 아니기 때문에, 문제를 보자마자 바로 푸는 것이 중요합니다. 보통 문제가 이렇게 짧으면 정청 어렵거나 엄청 쉽거나 둘 중 하나일 가능성이 높습니다. 어려운지 쉬운지는 읽어보면 바로 알 수 있는데요, 이 문제같은 경우에는 쉬운 문제입니다.
난이도 하의 문제들은 대개 5분내에 푸는 것이 좋습니다. 뒤에 나올 어려운 문제에 더 많은 시간을 쏟기 위함입니다.

▶ 접근 방법

이 문제에서는 문자열에서 'O'가 나오는 동안의 점수를 누적하고, 'X'를 만나면 점수를 0으로 설정해야합니다. 매 문제마다 해당하는 점수를 누적하여 더하고, 총점수를 구합니다.

▶ TIP

연속된 이라는 단어가 나왔을 때는 '뭘 언제까지 누적하지?' 라는 생각으로 접근하면 좋은데요, 하 난이도의 문제를 모아서 풀어보면 이런 유형의 문제에 대한 감이 오실거예요! 하 문제들은 양치기가 통하는 문제 유형이 많으니, 심심할 때 하나씩 풀어 두시면 좋습니다.

2017 10회 EPPER 2번

1) Python 소스코드

```python
def solution(user_input):
  # 총 점수를 저장하기 위한 변수
  total_score = 0
  # 'O'이 지속되는 횟수를 저장하기 위한 변수
  score = 0

  # 매 문제마다 점수를 업데이트한다
  for a in list(user_input):
      if a == "O":
          # 'O'가 지속될 경우 누적해서 score을 더한다
          score += 1
      else:
          # 'X'를 만나면 점수를 다시 0으로 초기화시킨다
          score = 0
      # 매 문제마다 총점수를 업데이트 한다
      total_score += score

  return total_score
if __name__=='__main__':
  user_input = input()
  if len(user_input) > 1000:
```

2017 10회 EPPER 2번

2) JAVA 소스코드

```
import java.util.Scanner;

public class low {
  public static int solution(String str) {
      // 총 점수를 저장하기 위한 변수
      int totalScore = 0;
      // 'O'이 지속되는 횟수를 저장하기 위한 변수
      int score = 0;
      // 매 문제마다 점수를 업데이트한다
      for (int i=0; i<str.length(); i++){
          // 'O'가 지속될 경우 누적해서 score을 더한다
          if (str.charAt(i)=='O'){
              score += 1;
          }
          // 'X'를 만나면 점수를 다시 0으로 초기화시킨다
          else{
              score = 0;
          }
          // 매 문제마다 총점수를 업데이트 한다
          totalScore += score;
      }
  return totalScore;
  }
  public static void main(String [] args){
      Scanner scanner =new Scanner(System.in);
```

백준 13305번 주유소 최소비용

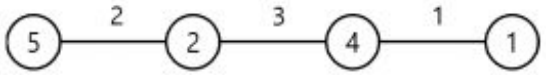
[문제링크]

문제

어떤 나라에 N개의 도시가 있다. 이 도시들은 일직선 도로 위에 있다. 편의상 일직선을 수평 방향으로 두자. 제일 왼쪽의 도시에서 제일 오른쪽의 도시로 자동차를 이용하여 이동하려고 한다. 인접한 두 도시 사이의 도로들은 서로 길이가 다를 수 있다. 도로 길이의 단위는 km를 사용한다.

처음 출발할 때 자동차에는 기름이 없어서 주유소에서 기름을 넣고 출발하여야 한다. 기름통의 크기는 무제한이어서 얼마든지 많은 기름을 넣을 수 있다. 도로를 이용하여 이동할 때 1km마다 1리터의 기름을 사용한다. 각 도시에는 단 하나의 주유소가 있으며, 도시 마다 주유소의 리터당 가격은 다를 수 있다. 가격의 단위는 원을 사용한다.

예를 들어, 이 나라에 다음 그림처럼 4개의 도시가 있다고 하자. 원 안에 있는 숫자는 그 도시에 있는 주유소의 리터당 가격이다. 도로 위에 있는 숫자는 도로의 길이를 표시한 것이다.



제일 왼쪽 도시에서 6리터의 기름을 넣고, 더 이상의 주유 없이 제일 오른쪽 도시까지 이동하면 총 비용은 30원이다. 만약 제일 왼쪽 도시에서 2리터의 기름을 넣고(2×5 = 10원) 다음 번 도시까지 이동한 후 3리터의 기름을 넣고(3×2 = 6원) 다음 도시에서 1리터의 기름을 넣어(1×4 = 4원) 제일 오른쪽 도시로 이동하면, 총 비용은 20원이다. 또 다른 방법으로 제일 왼쪽 도시에서 2리터의 기름을 넣고(2×5 = 10원) 다음 번 도시까지 이동한 후 4리터의 기름을 넣고(4×2 = 8원) 제일 오른쪽 도시까지 이동하면, 총 비용은 18원이다.

각 도시에 있는 주유소의 기름 가격과, 각 도시를 연결하는 도로의 길이를 입력으로 받아 제일 왼쪽 도시에서 제일 오른쪽 도시로 이동하는 최소의 비용을 계산하는 프로그램을 작성하시오.

백준 13305번 주유소 최소비용

입력

표준 입력으로 다음 정보가 주어진다. 첫 번째 줄에는 도시의 개수를 나타내는 정수 N(2 ≤ N ≤ 100,000)이 주어진다. 다음 줄에는 인접한 두 도시를 연결하는 도로의 길이가 제일 왼쪽 도로부터 N-1개의 자연수로 주어진다. 다음 줄에는 주유소의 리터당 가격이 제일 왼쪽 도시부터 순서대로 N개의 자연수로 주어진다. 제일 왼쪽 도시부터 제일 오른쪽 도시까지의 거리는 1이상 1,000,000,000 이하의 자연수이다. 리터당 가격은 1 이상 1,000,000,000 이하의 자연수이다.

출력

표준 출력으로 제일 왼쪽 도시에서 제일 오른쪽 도시로 가는 최소 비용을 출력한다.

예제 입력 1 복사

```
4
2 3 1
5 2 4 1
```

예제 출력 1 복사

```
18
```

예제 입력 2 복사

```
4
3 3 4
1 1 1 1
```

예제 출력 2 복사

```
10
```

백준 13305번 주유소 최소비용

► 접근 방법

최소비용의 주유값으로 최대한 멀리 가는 것이 목적입니다. 따라서 상황별 최선의 선택을 하는 그리디 알고리즘을 사용하여 문제를 푸는 것이 좋습니다.

► 그리디 알고리즘이란?

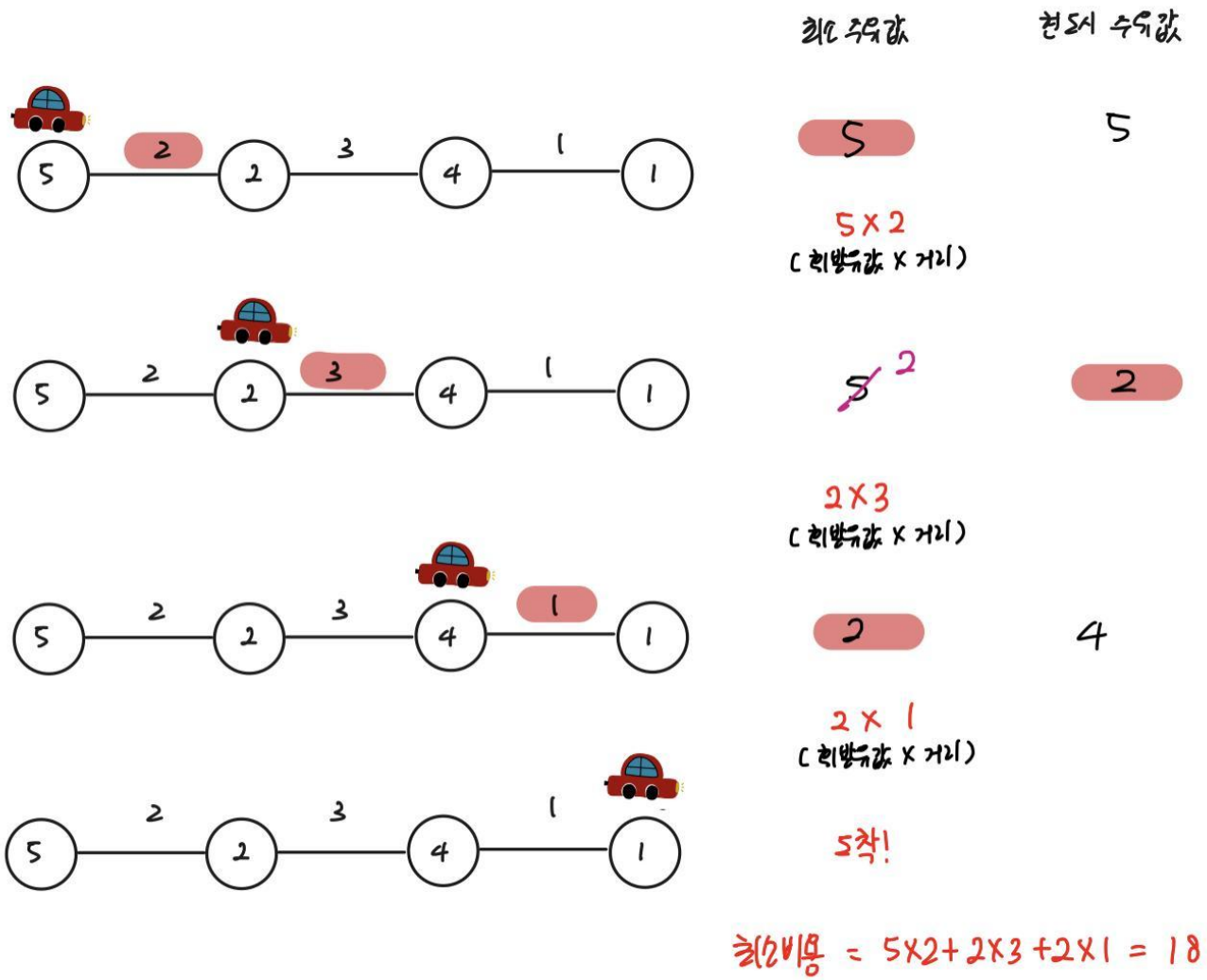
Greedy algorithm은 어떤 문제에 대한 최적의 결괏값 또는 해결방안을 찾기 위해 사용하는 메소드중 하나로, 순간마다 최적의 답이라고 판단되는 대상을 순차적으로 선택해나가는 방식이다.

그리디 알고리즘은 순간의 최선의 선택을 하기 때문에, 항상 최적의 결과를 가져온다고 할 수는 없다. 단, 대부분의 경우 정답에 근접한 해답을 준다.

본 문제의 경우, 그리디 알고리즘을 사용하면 최적해를 구할 수 있다. 그리디 알고리즘은 자주 출제되는 문제유형 중 하나로, 그리디 알고리즘을 모아서 풀어두어 유형을 익히는 것을 추천한다.

[그리디 알고리즘 백준 문제 모음]

백준 13305번 주유소 최소비용



▶ 필요한 코드

첫번째 주유소에서는 무조건 다음 도시로 갈정도의 휘발유를 주유해야합니다.

그 이후의 주유소에서는

1. 이전의 도시중 가장 싼 주유소와
2. 현 도시의 주유소 가격을

비교하여 더 싼 주유소에서 주유를 하고 다음 도시로 진행ㅎ야 합니다.

백준 13305번 주유소 최소비용

1) Python 소스코드

```python
def solution(cities, distance, price):
  # 총 주유 가격
  cost = 0
  # 처음 시작할때에는 다음 도시로 가기 위해 무조건 주유를 해야한다
  min_cost = price[0]
  # 모든 도시를 다 방문하는동안,
  for i in range(cities - 1):
      # 최소비용의 도시를 찾고 ( 지나온 도시 + 현 도시)
      if price[i] < min_cost:
          # 만일 현 도시의 주유값이 더 쌀 경우, 최소 주유비용을 업데이트한다
          min_cost = price[i]
      # 다음도시로 가기  위한 거리와 가장 싼 주유값을 곱하여 다음 도시로 진행한다
      cost += min_cost * distance[i]
  return cost

if __name__=='__main__':
  cities = int(input())
  distance = list(map(int, input().split()))
  price = list(map(int, input().split()))
  print(solution(cities, distance, price))
```

백준 13305번 주유소 최소비용

2) JAVA 소스코드

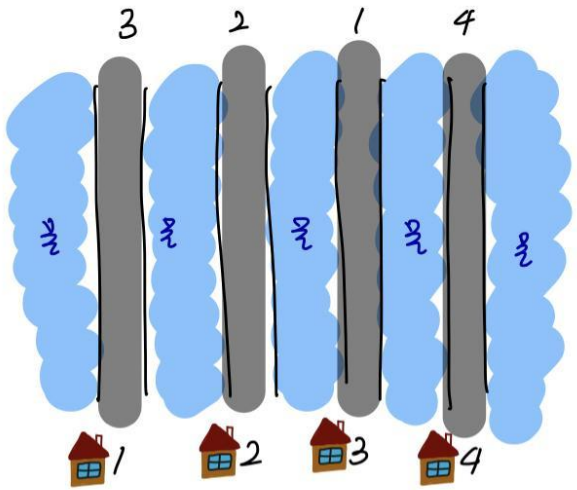
```
import java.util.Scanner;

public class mid {
  public static int solution(int cities, int[] distance, int [] price) {
      // 총 주유 가격
      int totalPrice = 0;
      // 처음 시작할때에는 다음 도시로 가기 위해 무조건 주유를 해야한다
      int minCost = price[0];
      // 모든 도시를 다 방문하는동안,
      for (int i=0; i<cities-1; i++){
          // 최소비용의 도시를 찾고 ( 지나온 도시 + 현 도시)
          if (price[i] < minCost)
              // 만일 현 도시의 주유값이 더 쌀 경우, 최소 주유비용을 업데이트한다
              minCost = price[i];
          // 다음도시로 가기  위한 거리와 가장 싼 주유값을 곱하여 다음 도시로 진행한다
          totalPrice += minCost * distance[i];
      }
      return totalPrice;
  }
  public static void main(String [] args){
      Scanner scanner =new Scanner(System.in);
      int cities = scanner.nextInt();
      int[] distance = new int[cities-1];
      for(int i=0; i<cities-1; i++)
```

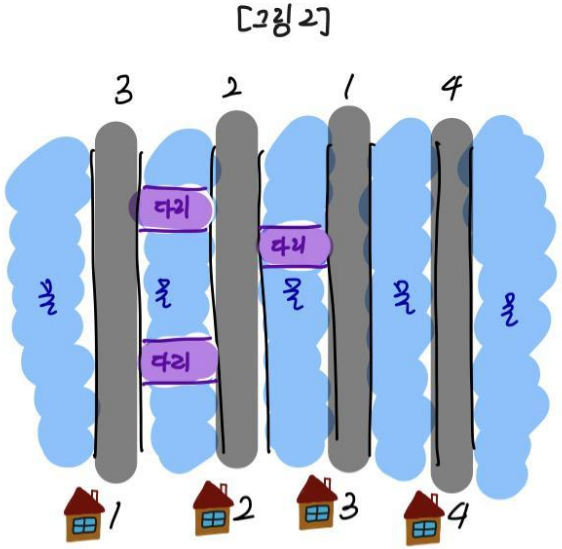
[문제 링크]

2021 16회 EPPER 프로그래밍 8번

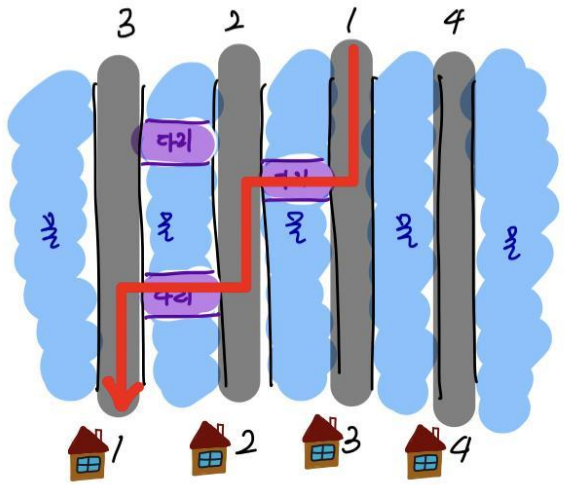
[그림1]



[그림2]



[그림3]



▶ 문제

겨울이와 친구들은 자신의 집으로 돌아가야 합니다. 겨울이 포함 4명의 사람이 있다고 했을 때, 현재 각 사람이 갈 수 있는 길은 그림1과 같습니다. 편의를 위해 사람에게 번호를 부여하였으며, 각 집에는 살고있는 사람의 번호로 표시됩니다.

이런 도로상황에서, 우리는 강을 가로지르는 다리를 추가할 수도 하지 않을 수도 있습니다. 다리 추가 작업을 마친 후, 각 사람은 도로를 따라 목표지점인 본인 번호의 집까지 가야합니다.

1번사람을 예시로 들면, 아무런 다리가 설치되어있지 않을 경우, 3번집으로 가게 됩니다. 하지만 그림2 와 같은 다리가 놓여있을 경우 그림3처럼 빨간 경로를 따라 1번집에 도착하게 됩니다.

다른 사람들의 경우에도, 2번사람은 2번집이라는 결과를 얻을 수 있습니다.
3번 사람은 3번집이라는 결과를 얻을 수 있습니다.
4번사람은 4번집이라는 결과를 얻을 수 있습니다.

문제에서는 초기의 사람 배치 순서를 입력으로 줍니다.
그림1의 경우 입력은 [3, 2, 1, 4] 입니다. 각 사람이 자신의 집으로 돌아갈 수 있게 하는 최소개의 다리의 수를 return 하도록 solution함수를 작성해주세요.

2021 16회 EPPER 프로그래밍 8번

▶ 입력

리스트의 길이

초기 사람 순서 리스트

▶ 출력

필요한 다리의 개수

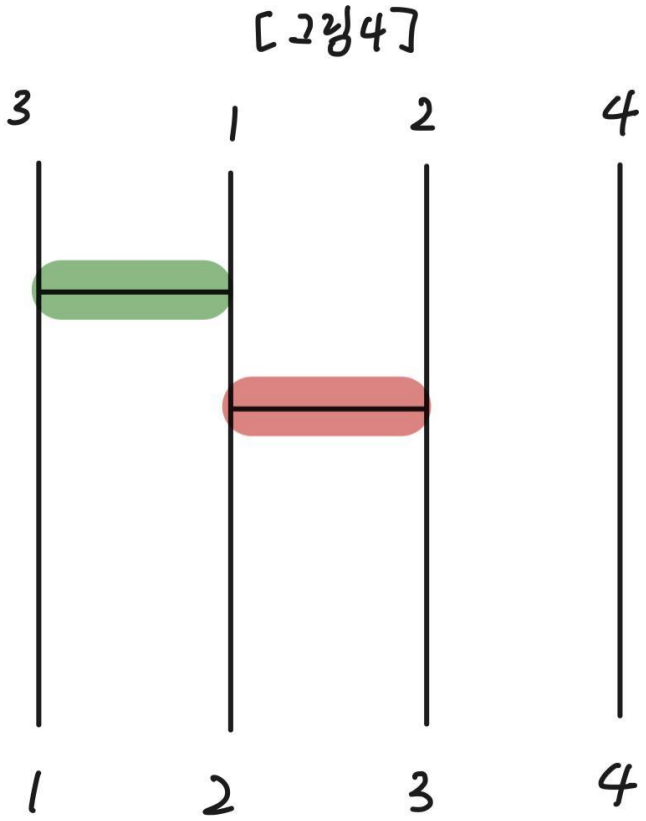
▶ 제한사항

- 1<=n<=100,000
- Goal의 길이 = n
- Goal의 모든 원소는 1-n이 각각 1번씩 포함되어 있습니다.

▶ 예시

n	final	result
4	[3, 2, 1, 4]	3
5	[1, 2, 3, 4, 5]	0
5	[5, 4, 3, 2, 1]	10
4	[2, 1, 4, 3]	2

2021 16회 EPPER 프로그래밍 8번



► 접근방법

본 문제는 전형적인 사다리 타기 문제로, 문제의 규칙만 파악하면 쉽게 풀리는 문제입니다.

사다리 타기에 있어서는, 사다리가 한 개 추가될 때마다 사다리의 오른쪽과 왼쪽에 있는 번호가 서로 바뀌게 됩니다. 그림4의 경우 초록색으로 표시된 사다리를 만났을 때 [3, 1, 2, 4] -> [1, 3, 2, 4]로 바뀝니다. 빨간색의 사다리를 만났을 때에는 [1, 3, 2, 4] -> [1, 2, 3, 4] 로 바뀝니다. 이렇게 사다리를 만났을 경우 두 자리가 스왑됐음을 알 수 있습니다. 다리를 사이에 두고 두 자리가 스왑된다는 규칙을 발견하면 문제의 90%는 푼 것입니다. 최종 결과가 [1, 2, 3, 4]로 정렬된 리스트입니다. 스왑을 이용한 정렬? -> 버블 소트를 이용하면 됩니다.
버블 소트를 이용하여 스왑이 필요한 곳마다 다리를 놓아주면 최종 정렬이 되기 위해 필요한 다리의 개수를 구할 수 있습니다.

► TIP

이러한 문제를 봤을 때, 무작정 문제를 풀기 시작하면, 메모리 제한이나 시간제한에 걸릴 수 있습니다. 그래서 문제를 풀기 전, 요구하는게 무엇인지 정확하게 파악하고 코드를 작성하기 시작해야 시간을 절약할 수 있습니다. 보통 이러한 제한이 걸려 있는 경우, 예시 입력은 작기 때문에 코드가 아무런 문제가 없어 보일 수 있으나, 입력의 개수가 늘어날 경우 영원히 끝나지 않는, 즉 런타임 에러를 내는 좋지 않은 코드를 쓰게 될 수도 있습니다. 대충 빅오를 계산했을 때, log n 의 시간이 되는 것이 가장 좋고, n까지도 괜찮지만, 2^n의 코드는 절대절대 좋지 않으므로, 이러한 실수를 하지 않기 위해 미리 풀이를 구상하는 것을 연습해보세요!

2021 16회 EPPER 프로그래밍 8번

1) Python 소스코드

```python
def solution(n, final):
  bridge = 0
  # bubble sort를 이용하여 순차적으로 정렬을 맞춰내려간다
  # swap이 필요할때마다 bridge 값을 증가시킨다
  for i in range(1, n):
      for j in range(i-1, -1, -1):
          if final[j] > final[i]:
              bridge += 1

  return bridge

if __name__=='__main__':
  n = int(input())
  final = list(map(int, input().split()))
  print(final)
  print(solution(n, final))
```

2021 16회 EPPER 프로그래밍 8번

2) JAVA 소스코드

```
import java.util.Scanner;

public class high {
  public static int solution(int n, int[] finalArray) {
      int bridge = 0;
      // bubble sort를 이용하여 순차적으로 정렬을 맞춰내려간다
      // swap이 필요할때마다 bridge 값을 증가시킨다
      for(int i = 0; i<n; i++)
          for( int j=i-1; j>=0; j--)
              if (finalArray[j]>finalArray[i])
                  bridge+=1;

      return bridge;
  }
  public static void main(String [] args){
      Scanner scanner =new Scanner(System.in);
      int n = scanner.nextInt();
      int[] finalArray = new int[n];
      for(int i=0; i<n; i++)
      {
          finalArray[i]=scanner.nextInt();
      }
      System.out.println(solution(n, finalArray));
```