



## Experiment 8

**Student Name:** Zatch

**Branch:** CSE

**Semester:**5

**Subject Name:** AP

**UID:**

**Section/Group:**

**Date of Performance:**

**Subject Code:** 22CSP-314

### 1. Aim:

**Problem Statement:** - Alice is a kindergarten teacher. She wants to give some candies to the children in her class. All the children sit in a line and each of them has a rating score according to his or her performance in the class. Alice wants to give at least 1 candy to each child. If two children sit next to each other, then the one with the higher rating must get more candies. Alice wants to minimize the total number of candies she must buy.

### 2. Objective:

The objective is to determine the minimum number of candies Alice needs to distribute to children seated in a line based on their rating scores. Each child must receive at least one candy, and children with higher ratings than their adjacent neighbors must receive more candies than those neighbors.

### 3. Implementation/Code:

```
public class Solution {  
    static BigInteger candies(int n, int[] arr) {  
        int[] cache = new int[arr.length];  
        cache[0] = 1;  
        for (int i = 1; i < arr.length; i++) {  
            if (arr[i-1] < arr[i]) {
```

```
        cache[i] = cache[i-1] + 1;
    }

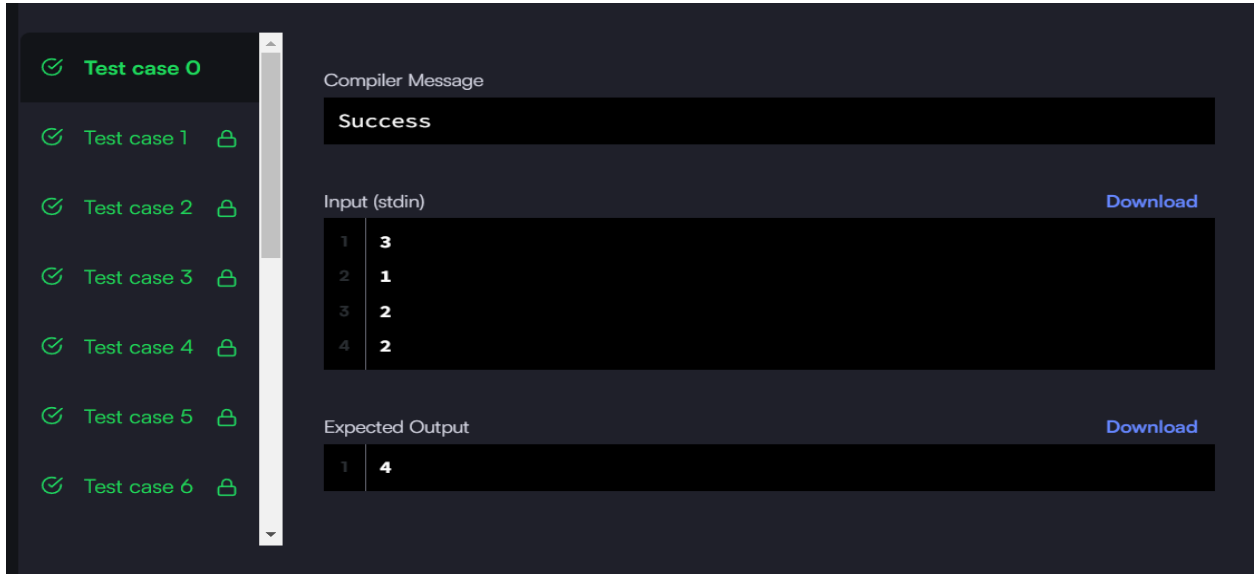
    if (arr[i-1] >= arr[i]) {
        cache[i] = 1;
    }
}

for (int i = arr.length - 2; i >= 0; i--) {
    if (arr[i] > arr[i+1]) {
        if (cache[i] <= cache[i+1]) {
            cache[i] = cache[i+1] + 1;
        }
    }
}

BigInteger sum = BigInteger.valueOf(0);
for (int i = 0; i < cache.length; i++) {
    sum = sum.add(BigInteger.valueOf(cache[i]));
}

return sum;
}
```

## 4. Output:



The screenshot shows a testing interface with a sidebar on the left containing a list of test cases from 0 to 6, each with a green checkmark and a lock icon. The main area on the right is divided into three sections: 'Compiler Message' showing 'Success', 'Input (stdin)' with a table of 4 rows, and 'Expected Output' with a table of 1 row. Each of these sections has a 'Download' link to its right.

Input (stdin)	
1	3
2	1
3	2
4	2

Expected Output	
1	4

## Problem -2

### 1. Aim:

**Problem Statement:** - Marc loves cupcakes, but he also likes to stay fit. Each cupcake has a calorie count, and Marc can walk a distance to expend those calories. If Marc has eaten  $j$  cupcakes so far, after eating a cupcake with  $c$  calories he must walk at least  $2j * cmiles$  to maintain his weight.

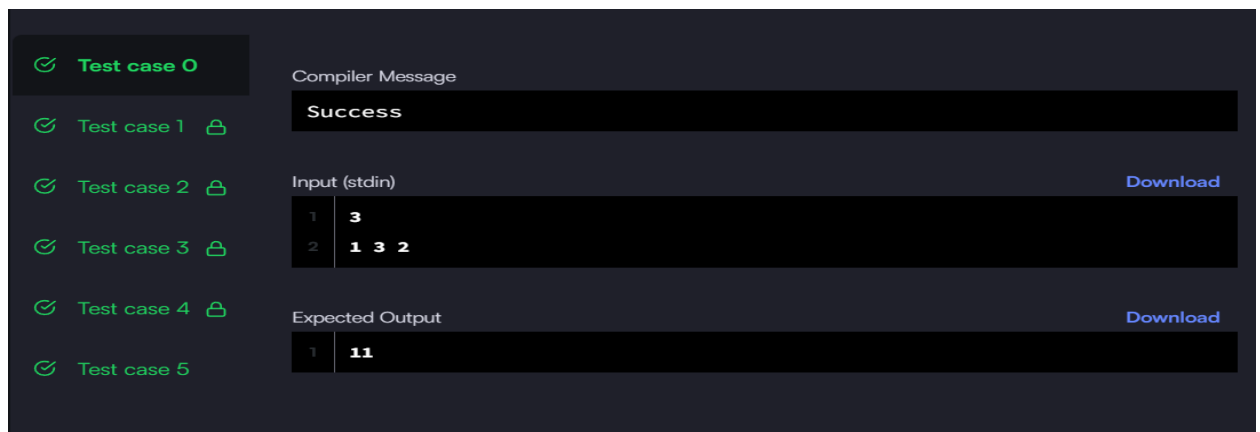
### 2. Objective:

The objective is to calculate the minimum total distance Marc needs to walk to offset the calories from the cupcakes he has eaten. Given that Marc must walk an increasing distance proportional to the number of cupcakes consumed, the goal is to determine the total walking distance required to balance out the calorie intake from all cupcakes.

### 3. Code:

```
class Result {  
  
    public static long marcsCakewalk(List<Integer> calorie) {  
        // Write your code here  
        Collections.sort(calorie, Collections.reverseOrder());  
  
        long totalCandies = 0;  
        for (int i = 0; i < calorie.size(); i++) {  
            totalCandies += (1L << i) * calorie.get(i);  
        }  
  
        return totalCandies;  
    }  
}
```

### 4. Output:



The screenshot displays a coding platform interface with a list of test cases on the left and a detailed view of the first test case on the right.

**Test Cases List:**

- Test case 0 (Selected)
- Test case 1
- Test case 2
- Test case 3
- Test case 4
- Test case 5

**Compiler Message:** Success

**Input (stdin):**

Case	Input
1	3
2	1 3 2

[Download](#)

**Expected Output:**

Case	Output
1	11

[Download](#)

## 5. Learning Outcome

- i. Learn to solve optimization problems where constraints involve comparative values between adjacent elements.
- ii. Understand how to implement a two-pass algorithm to achieve the optimal solution in such scenarios.
- iii. learn how to apply cumulative cost calculations in scenarios involving variable constraints.
- iv. They will understand how to optimize and calculate required values based on a progressive increase in factors, such as the distance Marc must walk per cupcake.
- v. This exercise will enhance problem-solving skills in handling scenarios with incremental constraints and applying mathematical calculations effectively.