Experiment 8A

Student Name: Tanmaya Kumar Pani	**UID: 22BCS12986**
Branch: BE-CSE	**Section/Group: IOT-613B**
Semester: 5	**Date of Performance: 24/09/2024**
Subject Name: AP Lab	**Subject Code: 22CSH-311**

1. TITLE:

Marc's Cakewalk

2. AIM:

Problem 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 ccc calories he must walk at least 2j×c miles to maintain his weight.

3. Objective

Print Complete the marcsCakewalk function in the editor below.

marcsCakewalk has the following parameters(s):

int calorie[n]: the calorie counts for each cupcake

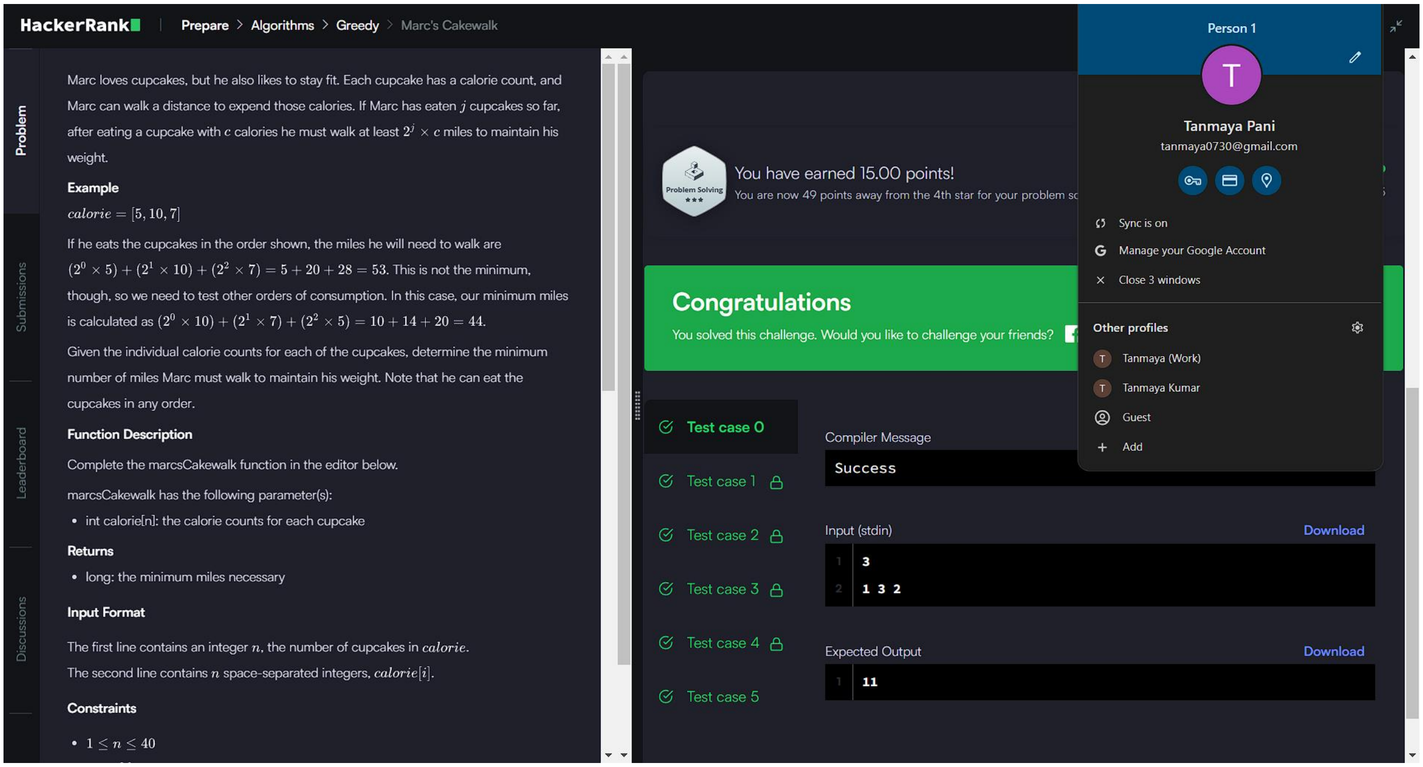
4. Algorithm

1. Accept the number of calorie values n and input each calorie into a fixed-size array.
2. Sort the array of calorie values in descending order to maximize the total miles.
3. Sort Iterate through the sorted array, computing the total miles by multiplying each calorie value by a power of two (specifically, 2 for each index i).
4. Sum up the miles for each calorie value to get the cumulative total.
5. Print the total miles calculated from the sorted calorie contributions.

5. Implemetation/Code

```
#include <bits/stdc++.h>
using namespace std;

int main() {
    int n;
    cin >> n;

    int calorie[40];
    for (int i = 0; i < n; ++i)
        cin >> calorie[i];

    // Sort the calories in descending order
    sort(calorie, calorie + n, greater<int>());

    long long total_miles = 0;

    // Calculate the total miles using the formula (1LL << i) * calorie[i]
    for (int i = 0; i < n; ++i)
        total_miles += (1LL << i) * calorie[i];

    cout << total_miles;
    return 0;
}
```

6. Output:



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 $2^j \times c$ miles to maintain his weight.

Example

$calorie = [5, 10, 7]$

If he eats the cupcakes in the order shown, the miles he will need to walk are $(2^0 \times 5) + (2^1 \times 10) + (2^2 \times 7) = 5 + 20 + 28 = 53$. This is not the minimum, though, so we need to test other orders of consumption. In this case, our minimum miles is calculated as $(2^0 \times 10) + (2^1 \times 7) + (2^2 \times 5) = 10 + 14 + 20 = 44$.

Given the individual calorie counts for each of the cupcakes, determine the minimum number of miles Marc must walk to maintain his weight. Note that he can eat the cupcakes in any order.

Function Description

Complete the marcsCakewalk function in the editor below.

marcsCakewalk has the following parameter(s):

- int calorie[n]: the calorie counts for each cupcake

Returns

- long: the minimum miles necessary

Input Format

The first line contains an integer n, the number of cupcakes in $calorie$.
The second line contains n space-separated integers, $calorie[i]$.

Constraints

- $1 \leq n \leq 40$

You have earned 15.00 points!
You are now 49 points away from the 4th star for your problem so

Tanmaya Pani
tanmaya0730@gmail.com

Congratulations
You solved this challenge. Would you like to challenge your friends?

✓ Test case 0
✓ Test case 1
✓ Test case 2
✓ Test case 3
✓ Test case 4
✓ Test case 5

Compiler Message

```
Success
```

Input (stdin)

```
3
1 3 2
```

Expected Output

```
11
```

7. Time Complexity : O(N*logn)

8. Space Complexity : O(n)

9. Learning Outcomes:-

1. Learn how sorting can impact problem-solving strategies by arranging elements to maximize or minimize certain conditions
2. Understand the usage of bitwise operations (shifting) in practical scenarios like exponential multiplication.
3. Develop an intuition for greedy algorithms where arranging input in a certain order can lead to optimal solutions.
4. Manage basic I/O operations in C++.

Experiment 8B

Student Name: Tanmaya Kumar Pani **UID: 22BCS12986**

Branch: BE-CSE **Section/Group: IOT-613B**

Semester: 5 **Date of Performance: 24/09/2024**

Subject Name: AP Lab **Subject Code: 22CSH-311**

1. TITLE:

Candies

2. AIM:

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.

3. Objective

Return Complete the candies function in the editor below.

candies has the following parameter(s):

int n: the number of children in the class

int arr[n]: the ratings of each student

4. Algorithm

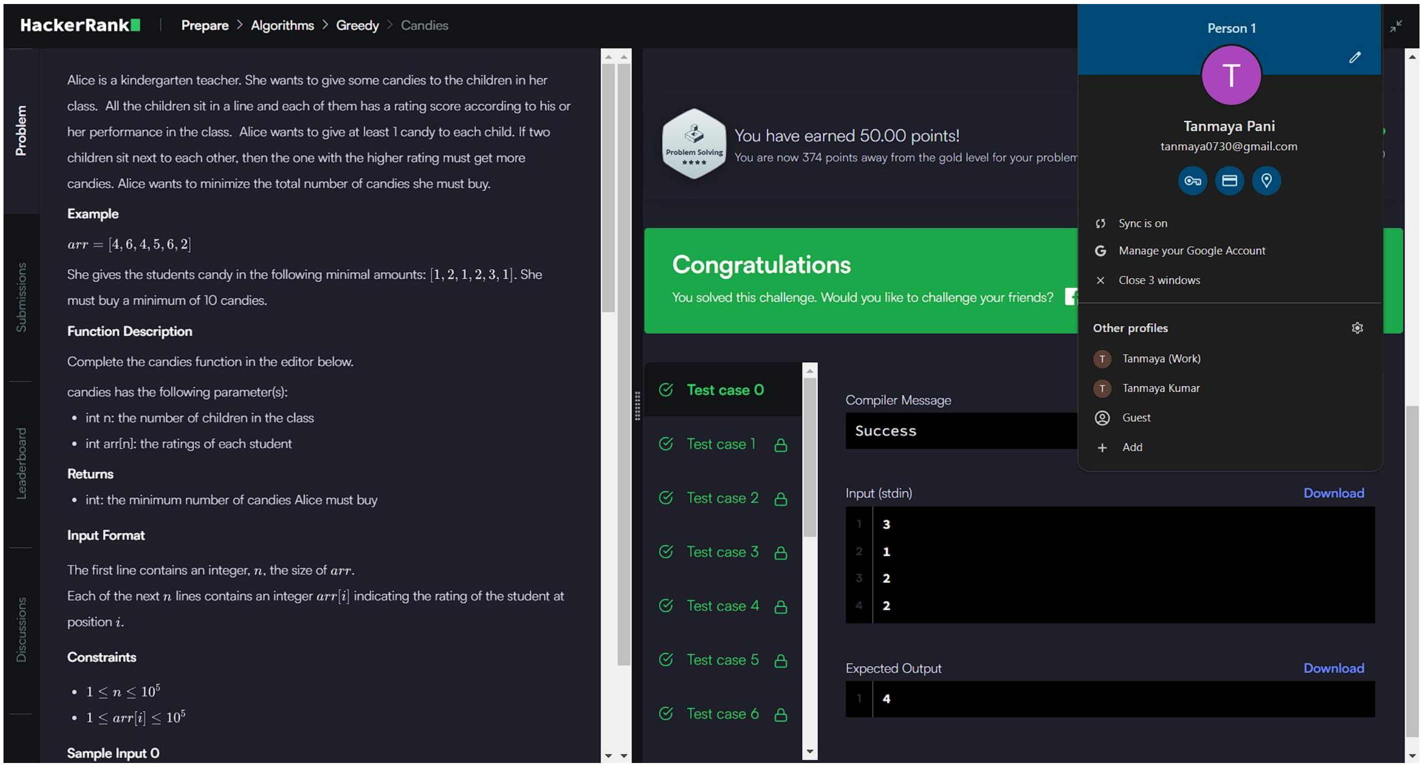
- Read the number of elements n, allocate two arrays arr and candy, and input n values into arr while initializing candy to 1 for each element.
- Iterate from the start to the end of the array, incrementing candy count for each element if it's greater than the preceding element.
- Iterate from the end to the start of the array, adjusting the candy count to ensure every element that's greater than the succeeding one has more candy.

- Sum all elements in the candy array using accumulate to determine the total number of candies required.
- Print the total number of candies.

5. Implemetation/Code:

```
#include <bits/stdc++.h>
using namespace std;

int main() {
   int n;
   cin >> n;

   int* arr = new int[n];   // Array to store ratings
   int* candy = new int[n]; // Array to store candy distribution

   // Input ratings and initialize candy array
   for (int i = 0; i < n; i++) {
      cin >> arr[i];
      candy[i] = 1; // Everyone gets at least one candy
   }

   // First pass (left to right): if arr[i] > arr[i-1], increment candy count
   for (int i = 1; i < n; i++) {
      if (arr[i] > arr[i - 1]) {
         candy[i] = candy[i - 1] + 1;
      }
   }

   // Second pass (right to left): if arr[i] > arr[i+1], ensure max candy
count
   for (int i = n - 2; i >= 0; i--) {
      if (arr[i] > arr[i + 1]) {
         candy[i] = max(candy[i], candy[i + 1] + 1);
      }
   }

   // Calculate total candies using accumulate
   cout << accumulate(candy, candy + n, 0LL);

   // Clean up dynamic memory
   delete[] arr;
   delete[] candy;

   return 0;
}
```

6. Output:



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.

Example

$arr = [4, 6, 4, 5, 6, 2]$

She gives the students candy in the following minimal amounts: $[1, 2, 1, 2, 3, 1]$. She must buy a minimum of 10 candies.

Function Description

Complete the candies function in the editor below.

candies has the following parameter(s):

- int n: the number of children in the class
- int arr[n]: the ratings of each student

Returns

- int: the minimum number of candies Alice must buy

Input Format

The first line contains an integer, n, the size of arr.

Each of the next n lines contains an integer $arr[i]$ indicating the rating of the student at position i.

Constraints

- $1 \leq n \leq 10^5$
- $1 \leq arr[i] \leq 10^5$

Sample Input 0

You have earned 50.00 points!

You are now 374 points away from the gold level for your problem

Congratulations

You solved this challenge. Would you like to challenge your friends?

Compiler Message

```
Success
```

Input (stdin)

```
3
1
2
2
```

Expected Output

```
4
```

6. Time Complexity : O(n)

7. Space Complexity : O(n)

8. Learning Outcomes:-

1. Learn how a greedy approach can be used effectively to solve optimization problems by making local optimal choices.
2. Gain knowledge on how a two-pass algorithm can solve problems that depend on conditions from both preceding and succeeding elements.
3. Basic array operations including dynamic memory management in C++.
4. Implement functions from the C++ standard library like accumulate for mathematical operations on arrays.