# Editorial

quangvn2508

February 11, 2020

In this editorial, I would like to provide my approach for each of the questions. These solutions are not the only or the best and optimised solution. If you have a better solution and want to share please feel free to email me and I will update the solution to other contestants.

Visit my **Github** to view full code solution and test case generator.

# 1 Question 1: String evaluation [problem link]

**Setter: quangvn2508**
**Solution by quangvn2508**

- Time complexity: O($n$)

Straight forward, go through each of character in the string (S), add the ASCII value of that character to the total sum.

**Algorithm**

1. $sum \leftarrow 0$
2. FOR each character c in S:
   (a) $sum \leftarrow sum$ + ASCII value of c
3. PRINT $sum$ to standard output

## 2 Question 2: Skyfall [problem link]

**Setter: quangvn2508**
**Solution by quangvn2508**

- Time complexity: O($nlogn$)

First step, the array are not guarantee to be sorted (the sample test cases are). Therefore, I will firstly sort the array in decreasing order. Using the sort function in standard library, time complexity O(nlogn).

The idea is, as we pass through the sorted array, at each position, calculate the total of floors that the sky need to destroy before reach the current height. If the total of floors is more then the value P of the sky then the sky is stopped. Let look at the algorithm before I explain further.

Let define some symbol used in the algorithm:

- $height$ is the array denoted the height of each building (remember that we have sorted the array
- $height[i]$ is the height of building at position $i$ $(0 \leq i \leq n-1)$ and $height[i] > height[i+1]$.

**Algorithm**

$current\_height \leftarrow height[0]$
$current\_floors \leftarrow 0$
FOR $i = 1, 2, ..., n-1$:
    $change\_of\_height \leftarrow current\_height - height[i]$
    $increase\_of\_floors \leftarrow change\_of\_height * i$
    IF $current\_floors + increase\_of\_floors \geq P$:
        PRINT $current\_height - \lceil (P - current\_floors)/i \rceil$
    ELSE:
        $current\_floors \leftarrow current\_floors + increase\_of\_floors$
        $current\_height \leftarrow height[i]$
PRINT $-1$

Lets look at sample test case.

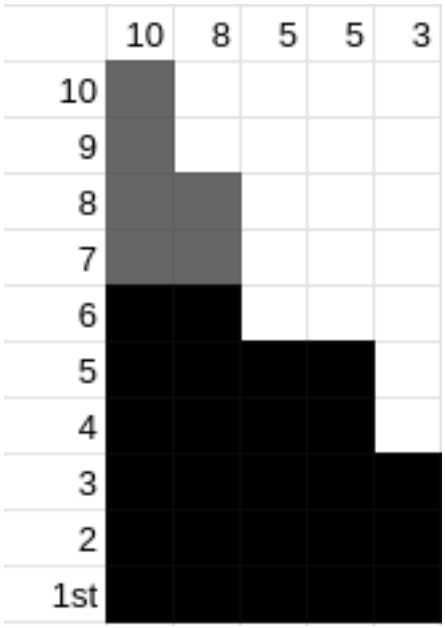


Figure 1: Sample test case

At position 0, the total floor the sky need to destroy before reach this height is 0 (since it is the highest building).

At position 1, the total floors is 2, in order to reach height 8, the sky need to destroy 2 floors of the first building. Therefore, $current\_floors = change\_of\_height * i = 2 * 1$. Note that $i$ also denote the number of building before building at $i$.

Moving on to position 2, the total floors to add is 6. Although the different of height is 3, there are two building before position 2. In this example, $P = 6$, therefore, $current\_floors + increase\_of\_floors = 2 + 6 > P = 6$.

Now we need to work out the floor where $current\_floors + increase\_of\_floors$ just over $P$. At height 8, the sky has destroyed 2 floors, therefore the remaining is 4. To work out how deep the sky will go, we use the width ($i$) divide the remaining value and the result will be 2, which mean the sky will stop after destroy 2 floors below 8.

We need to be careful when the result is not integer. For example, sample test 2 where $P$ is 7 instead of 6. At the last step, the result is 2.5 indicated that the sky stop between floor 6th and 5th. However, we are asked to report the number where the sky stop. There for the result need to be round up (ceiling at line 7).

**Comment:** since this algorithm go through the array once, the time complexity is O(n). However, the time complexity overall is O(nlogn) because of sorting.

# 3 Question 3: kth digit of nth Fibonacci number [problem link]

**Setter: quangvn2508**
**Solution by quangvn2508**

- Time complexity: O($n * k$)

Look at the constraints, we can see that the number can be massive (normal sequence start at 0 and 1 will have $F_{10000} > 10^{2000}$).

As I have mentioned in analysis, this question is unfair for those who not use Python. However, some participants (possibly all) exploited this language too much. For example, if the test case ask for the $2^{nd}$ digit of the $F_{10000}$ then after compute the $F_i$, the result should be modulo with 100 to save memory (@_@)

For those who did not know that python can handle significantly massive number then this section is for you. The solution I have in mind is to use array as an representation for the numbers and perform addition like we did in elementary school.

Let define some symbol used in the algorithm:

- $F_i, F_{i-1}, F_{i-2}$ are arrays of size k+1
- $F_i[j]$ is the $j^{th}$ digit from the least significant.

**Algorithm**

```
1  WHILE n > 2:
2      FOR j = 0, 1, 2, ..., k − 1:
3          F_i[j] ← F_{i−1}[j] + F_{i−2}[j]
4          IF F_i[j] > 9:
5              F_i[j + 1] ← ⌊F_i[j]/10⌋
6              F_i[j + 1] ← F_i[j]  (mod 10)
7      F_{i−2} ← F_{i−1}
8      F_{i−1} ← F_i
9      RESET F_i
10 PRINT F_{i−1}[k − 1]
```

If the array size is k then we might have problem at line 6, 7. Since if $k^{th}$ digit is overflow ($> 9$) then we will have run-time error. By setting the size of array to k, we can avoid more complicate code. The logic is trivial so I will not mention any example.

# 4 Question 4: Influenced sum [problem link]

**Setter: quangvn2508**
**Solution by quangvn2508**

- Time complexity: average O($n$), worst O($n^2$)

Since the influence of a node is depend on the longest distance from start node (S). Therefore, the obvious approach is to traverse the graph from (S) to find the longest distance (D), then traverse a second time to calculate the influenced value.

However, it is possible to calculate the influenced sum by perform breadth-first-search once. The idea is to find the sum of labels of all nodes with the same distance from (S). We can do this since all node have the same shortest distance (d) from (S) will have influence (F) and their $sum_d$ is

$$sum_d = n_1 * F + n_2 * F + ... + n_i * F = (n_1 + n_2 + ... + n_i) * F$$

After calculate all $sum_0, sum_1, ..., sum_D$. We can calculate the influenced sum by this equation

$$sum = sum_0 * (D + 1) + sum_1 * (D) + sum_2 * (D - 1) + ... + sum_{D-1} * 2 + sum_D * 1$$

Let define some symbol used in the algorithm:

- $Sum\_array$ will holds sum of node with the same distance.
- $Sum\_array[i]$ will contains $sum_i$ as mentioned above.
- $QNode$ is a queue of nodes' label.

**Algorithm**
BFS using queue:

```
1  PUSH start node S to QNode
2  WHILE QNode not empty:
3      s ← size of QNode
4      sum_i ← 0
5      FOR j = 1, 2, ..., s:
6          current_node ← pop first element in QNode
7          sum_i ← sum_i + current_node
8          FOR unvisited nodes which is adjacent to current_node:
9              mark node as visited
10             PUSH node to QNode
11     ADD sum_i to back of Sum_array
```

Calculate the influenced sum

```
12 max_influence ← size of Sum_array
13 influence_sum ← 0
14 FOR i = 0, 1, ..., max_influence − 1:
15     influence_sum ← influence_sum + (max_influence − i) * Sum_array[i]
16 PRINT influence_sum
```

Note that every time the while loop on line 2 repeat, the distance from start node (S) increase by 1 since the number of pop we perform in one while loop is equal to the size of queue at the beginning. Each node is add and remove from the $QNode$ once, therefore, BFS will have average time complexity of O($n$). However, the for loop on line 8 is difficult to analysis the number of repetitions. Worst case happen when the graph is a completed graph, since for each node, we need to go through their adjacency list which have (n-1) elements, hence, complexity O($n^2$).

Let take an example (test case #5)

```
10 3
2 1 6
4 0 4 8 9
5 3 5 6 7 8
1 2
5 1 5 7 8 9
2 2 4
3 0 2 8
3 2 4 9
4 1 2 4 6
3 1 4 7
```

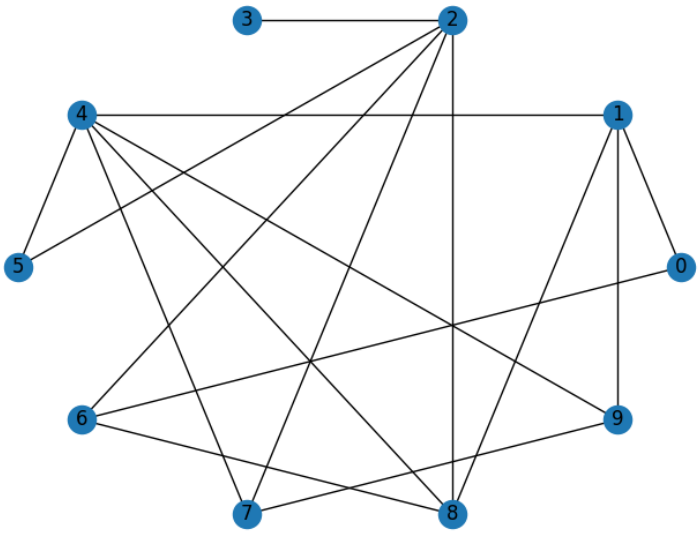


Figure 2: Test case 5

$n = 10$
$S = 3$
Follow the algorithm for BFS:

- $QNode = 3$ (line 1). Currently, the distance is 0 (only start node). Go through all node with this distance, $sum_0 = 3$ and $QNode = 2$ (node 2 is the only neighbor of 3).
- Similarly, visit node with distance 1 from (S). $sum_1 = 2$ and $QNode = 5, 6, 7, 8$ (node 2 have 5 neighbors 3,5,6,7,8, however, 3 is already visited).
- Visit nodes with distance 2 from (S). $sum_2 = 5 + 6 + 7 + 8 = 26$ and $QNode = 4, 0, 9, 1$.
- Lastly, nodes with distance 3 from (S). $sum_3 = 4 + 0 + 9 + 1 = 14$ and $QNode =$ (stop BFS)

Finally, we know that the maximum shortest distance is 3, hence, max influence is 4 (at S). Calculate the influenced sum

$$\begin{aligned} influence\_sum &= 4 * sum_0 + 3 * sum_1 + 2 * sum_2 + 1 * sum_3 \\ &= 4 * 3 + 3 * 2 + 2 * 26 + 1 * 14 \\ &= 84 \end{aligned}$$

which is the answer.