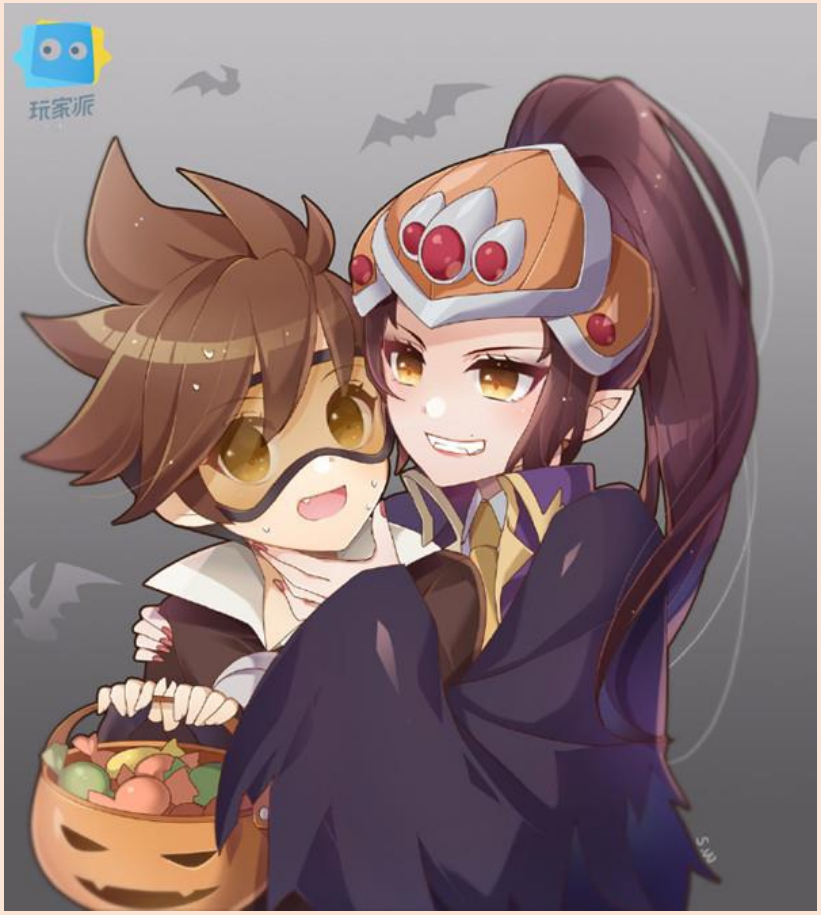
# #135.Candy

Yeung KinBong
2016/11/08

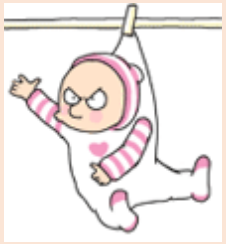


# Problem Description:

- There are *N* children standing in a line. Each child is assigned a rating value.
- You are giving candies to these children subjected to the following requirements:
  - Each child must have at least one candy.
  - Children with a higher rating get more candies than their neighbors.
- What is the minimum candies you must give?

# Solution:

Using greedy algorithm, this problem can be easily solved by scanning the array from left to right, right to left in O(n) time and O(n) space.



| 0 | 1 | 6 | 6 | 5 | 3 |
|---|---|---|---|---|---|

| 1 | 1 | 1 | 1 | 1 | 1 |
|---|---|---|---|---|---|

# Solution:

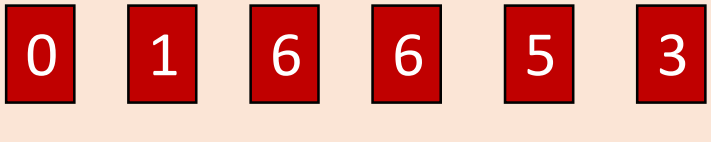
Using greedy algorithm, this problem can be easily solved by scanning the array from left to right, right to left in O(n) time and O(n) space.

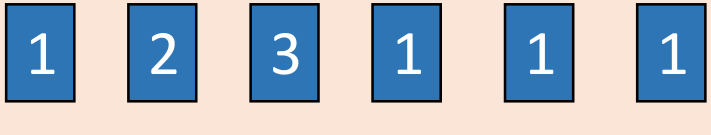
| 0 | 1 | 6 | 6 | 5 | 3 |
|---|---|---|---|---|---|

| 1 | 2 | 1 | 1 | 1 | 1 |
|---|---|---|---|---|---|

# Solution:

Using greedy algorithm, this problem can be easily solved by scanning the array from left to right, right to left in O(n) time and O(n) space.




# Solution:

Using greedy algorithm, this problem can be easily solved by scanning the array from left to right, right to left in O(n) time and O(n) space.

| 0 | 1 | 6 | 6 | 5 | 3 |
|---|---|---|---|---|---|

| 1 | 2 | 3 | 1 | 1 | 1 |
|---|---|---|---|---|---|

# Solution:

Using greedy algorithm, this problem can be easily solved by scanning the array from left to right, right to left in O(n) time and O(n) space.

| 0 | 1 | 6 | 6 | 5 | 3 |
|---|---|---|---|---|---|

| 1 | 2 | 3 | 1 | 2 | 1 |
|---|---|---|---|---|---|

# Solution:

Using greedy algorithm, this problem can be easily solved by scanning the array from left to right, right to left in O(n) time and O(n) space.

| 0 | 1 | 6 | 6 | 5 | 3 |
|---|---|---|---|---|---|

| 1 | 2 | 3 | 1 | 2 | 1 |
|---|---|---|---|---|---|

# Solution:

Using greedy algorithm, this problem can be easily solved by scanning the array from left to right, right to left in O(n) time and O(n) space.

| 0 | 1 | 6 | 6 | 5 | 3 |
|---|---|---|---|---|---|

| 1 | 2 | 3 | 1 | 2 | 1 |
|---|---|---|---|---|---|

# Solution:

Using greedy algorithm, this problem can be easily solved by scanning the array from left to right, right to left in O(n) time and O(n) space.

| 0 | 1 | 6 | 6 | 5 | 3 |
|---|---|---|---|---|---|

| 1 | 2 | 3 | 3 | 2 | 1 |
|---|---|---|---|---|---|

# Solution:

Using greedy algorithm, this problem can be easily solved by scanning the array from left to right, right to left in O(n) time and O(n) space.

| 0 | 1 | 6 | 6 | 5 | 3 |
|---|---|---|---|---|---|

| 1 | 2 | 3 | 3 | 2 | 1 |
|---|---|---|---|---|---|

# Solution:

Using greedy algorithm, this problem can be easily solved by scanning the array from left to right, right to left in O(n) time and O(n) space.

| 0 | 1 | 6 | 6 | 5 | 3 |
|---|---|---|---|---|---|

| 1 | 2 | 3 | 3 | 2 | 1 |
|---|---|---|---|---|---|

# Solution:

Using greedy algorithm, this problem can be easily solved by scanning the array from left to right, right to left in O(n) time and O(n) space.

| 0 | 1 | 6 | 6 | 5 | 3 |
|---|---|---|---|---|---|

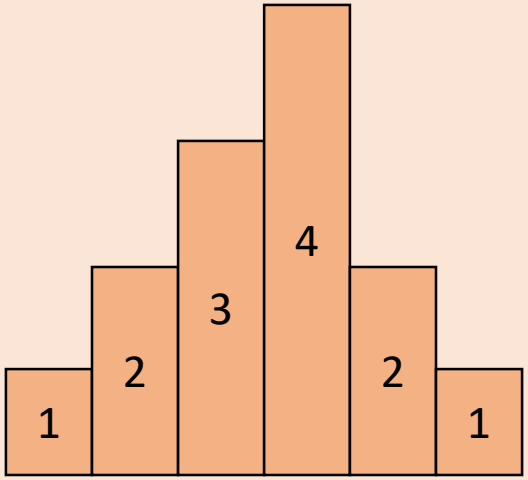
| 1 | 2 | 3 | 3 | 2 | 1 |
|---|---|---|---|---|---|

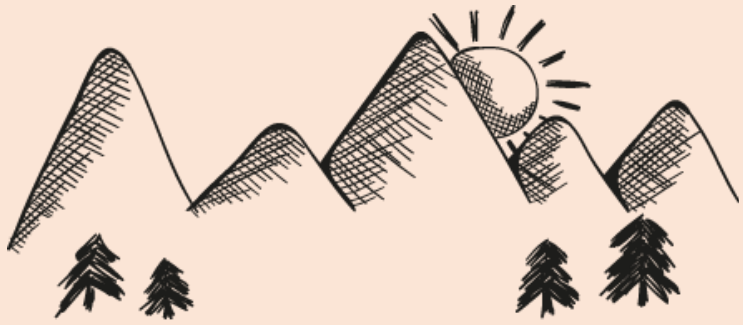
# Code:

```
class Solution{
public:
    int candy(vector<int>& ratings){
        vector<int> portion(ratings.size(), 1);

        for (unsigned i = 1; i<ratings.size(); i++){
            if ((ratings[i]>ratings[i - 1]) && portion[i] <= portion[i - 1]){
                portion[i] = portion[i - 1] + 1;
            }
        }

        int total = 0;
        for (unsigned i = ratings.size() - 1; i > 0; i--){
            if ((ratings[i - 1]>ratings[i]) && portion[i - 1] <= portion[i]){
                portion[i - 1] = portion[i] + 1;
            }
            total += portion[i];
        }
        total += portion[0];

         return total;
    }
};
```

# Proof:

- Every peak is independent. So we can divide the candy distribution into independent peaks.
- If we can prove that the construction of a single peak is optimal, we can prove the whole candy distribution is optimal.
- It's easy to prove every single peak is optimal.



# Another Solution:

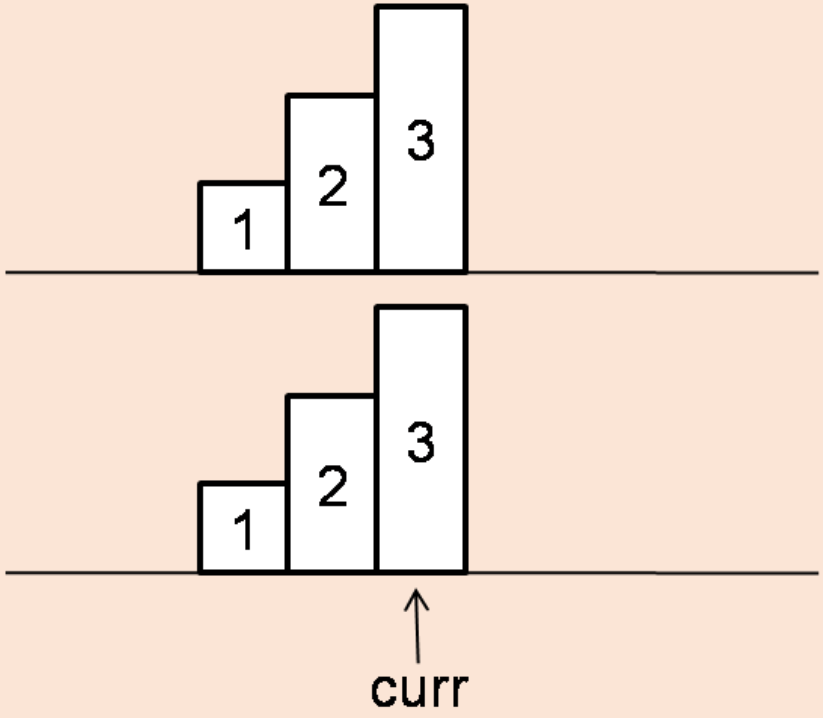
- **It takes O(n) time and O(1) space.**
- **This solution picks each element from the input array only once. First, we give a candy to the first child. Then for each child we have three cases:**
    1. His/her rating is equal to the previous one -> give 1 candy.
    2. His/her rating is greater than the previous one -> give him (previous + 1) candies.
    3. His/her rating is less than the previous one -> don't know what to do yet, let's just count the number of such consequent cases.

# Another Solution:

- When we enter 1 or 2 condition we can check our count from 3. If it's not zero then we know that we were descending before and we have everything to update our total candies amount: number of children in descending sequence of raitings - coundDown, number of candies given at peak - prev (we don't update prev when descending). Total number of candies for "descending" children can be found through arithmetic progression formula (1+2+...+countDown). Plus we need to update our peak child if his number of candies is less then or equal to countDown.

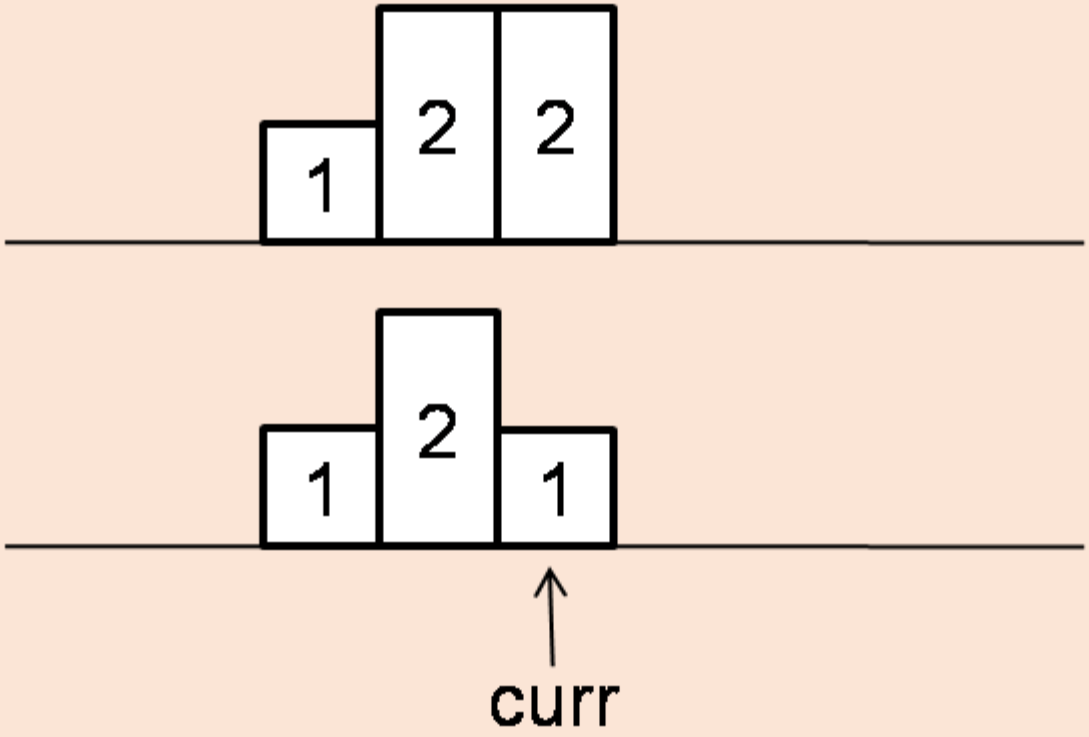
# Another Solution:

- If current rating is greater than previous one, then current value should be **prev + 1**



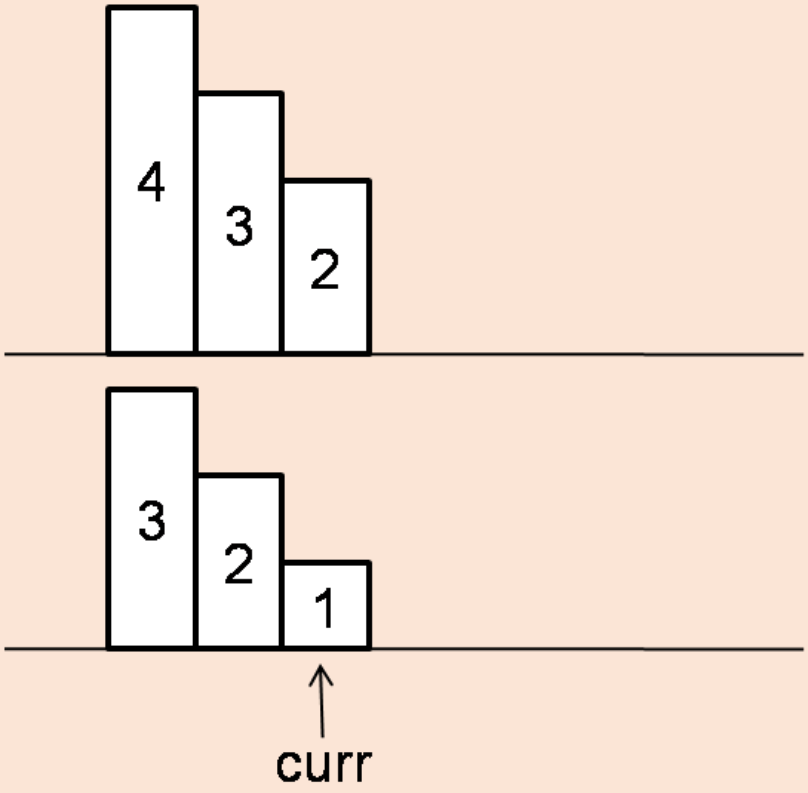
# Another Solution:

- If current rating is equal to previous one, then current value should be 1



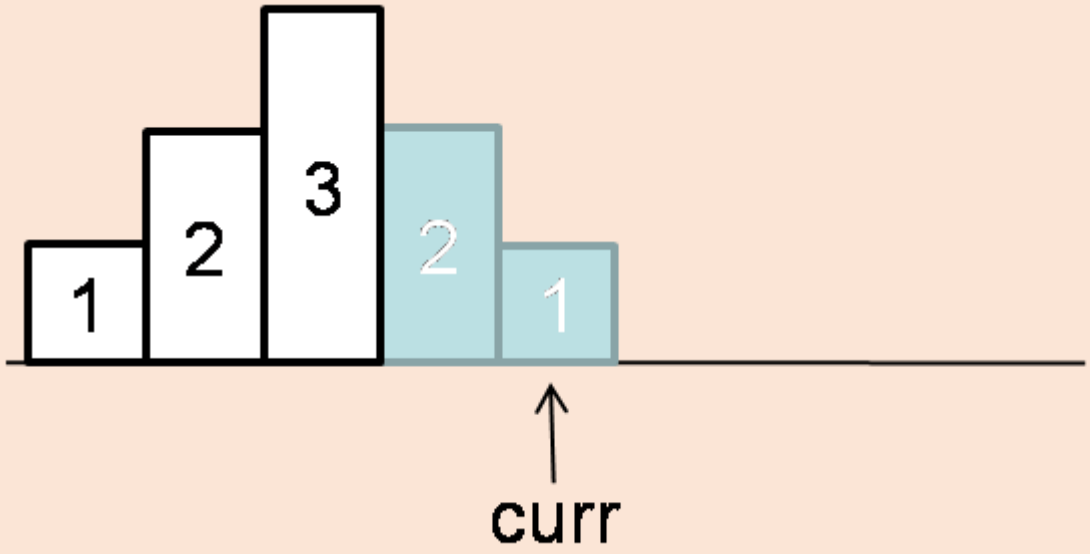
# Another Solution:

- If current rating is less than previous one, we don't know the value yet.



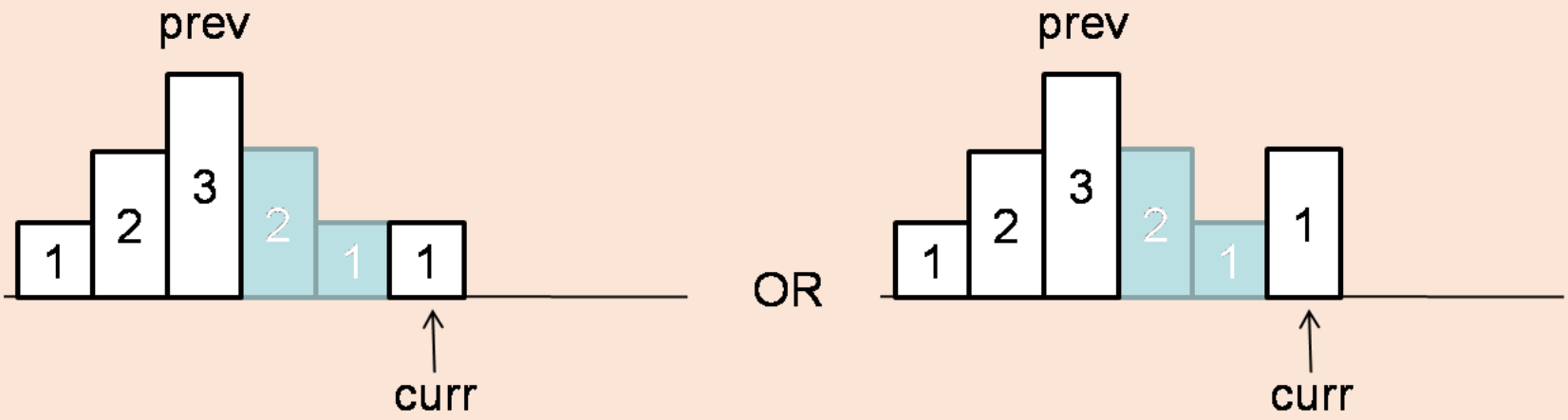
# Another Solution:

- Let's consider the continuous descending ratings. If there is continuous descending ratings, we will use countDown to memorize the descending length. Below case, countDown = 2 at curr position.



# Another Solution:

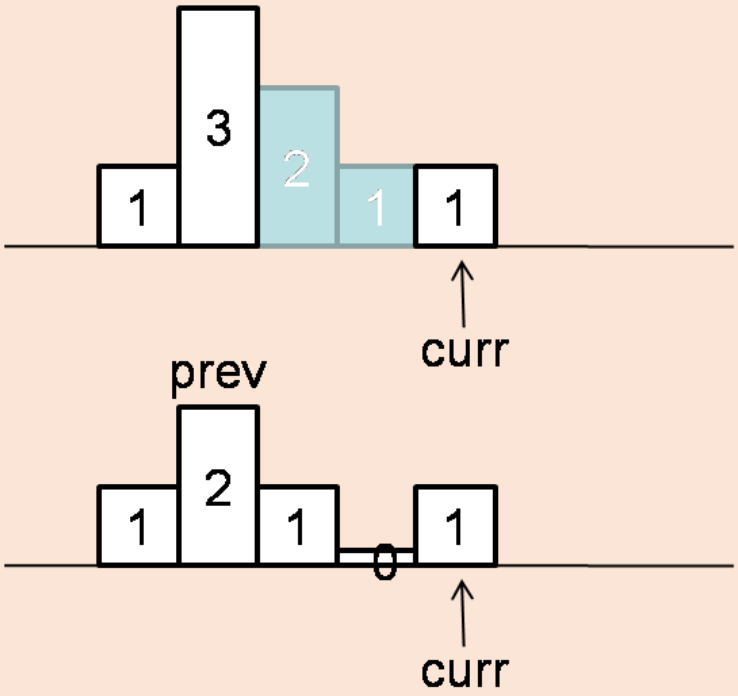
- During a descending order, when curr is equal to previous or greater than previous, we can use progression formula to calculate the descending area size.




size of descending = 1 + 2 = 2 * (2 + 1) / 2

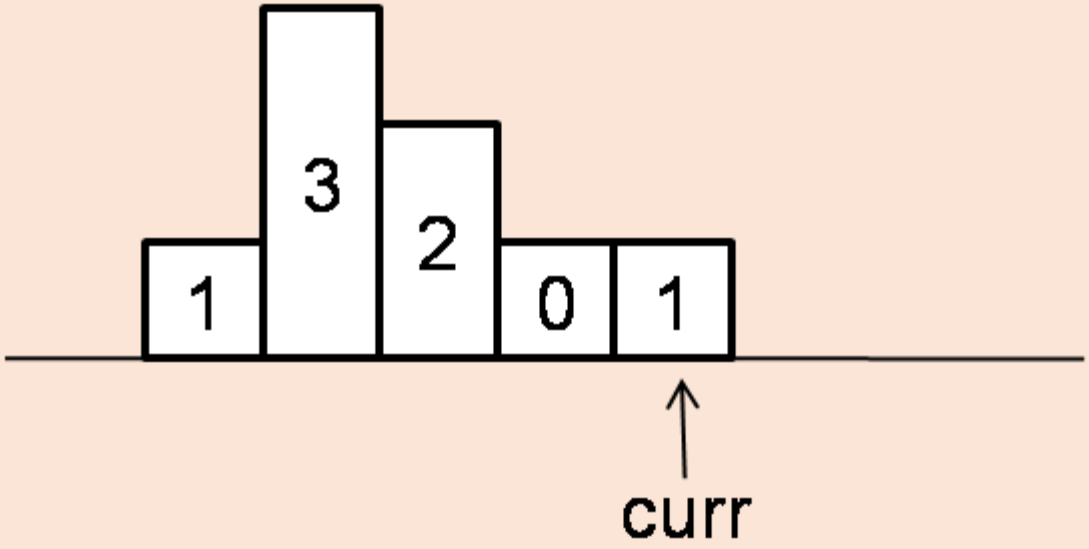
# Another Solution:

- Let's reconsider the prev when calculating size of descending area. Think about below case. Prev is 2, which is not tall enough and makes next 2 elements to 1 and 0.



# Another Solution:

- In this case when **countDown >= prev**, we should increase prev by **countDown – prev + 1.**



# Code:

```
public class Solution {
    public int candy(int[] ratings) {
        if (ratings == null || ratings.length == 0) return 0;
        int total = 1, prev = 1, countDown = 0;
        for (int i = 1; i < ratings.length; i++) {
            if (ratings[i] >= ratings[i-1]) {
                if (countDown > 0) {
                    total += countDown*(countDown+1)/2; // arithmetic progression
                    if (countDown >= prev) total += countDown - prev + 1;
                    countDown = 0;
                    prev = 1;
                }
                prev = ratings[i] == ratings[i-1] ? 1 : prev+1;
                total += prev;
            } else countDown++;
        }
        if (countDown > 0) { // if we were descending at the end
            total += countDown*(countDown+1)/2;
            if (countDown >= prev) total += countDown - prev + 1;
        }
        return total;
    }
}
```

# Other solutions:

- Dynamic Programming
  - https://discuss.leetcode.com/topic/48434/40-ms-cpp-dp-solution
  - https://www.quora.com/Dynamic-Programming-DP-How-do-I-solve-Candy-on-LeetCode
- Eight-Queens

Thank you!

X_X