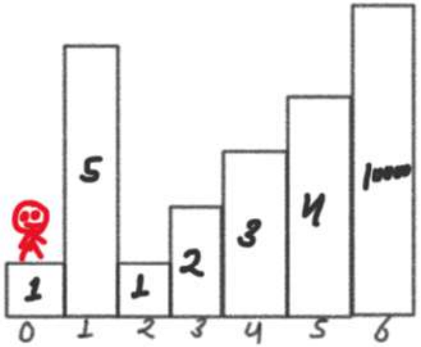
Furthest Building You Can Reach (LEETCODE-1642)

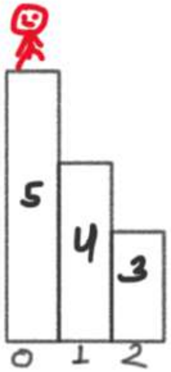
GitHub: **github.com/BCAPATHSHALA**
Date: 17-02-2024

READ Problem statement First

Example 1:
Input: heights = [1,5,1,2,3,4,10000], bricks = 5, ladders = 1
Output: 5



Example 2:
Input: heights = [5,4,3], bricks = 0, ladders = 0
Output: 2

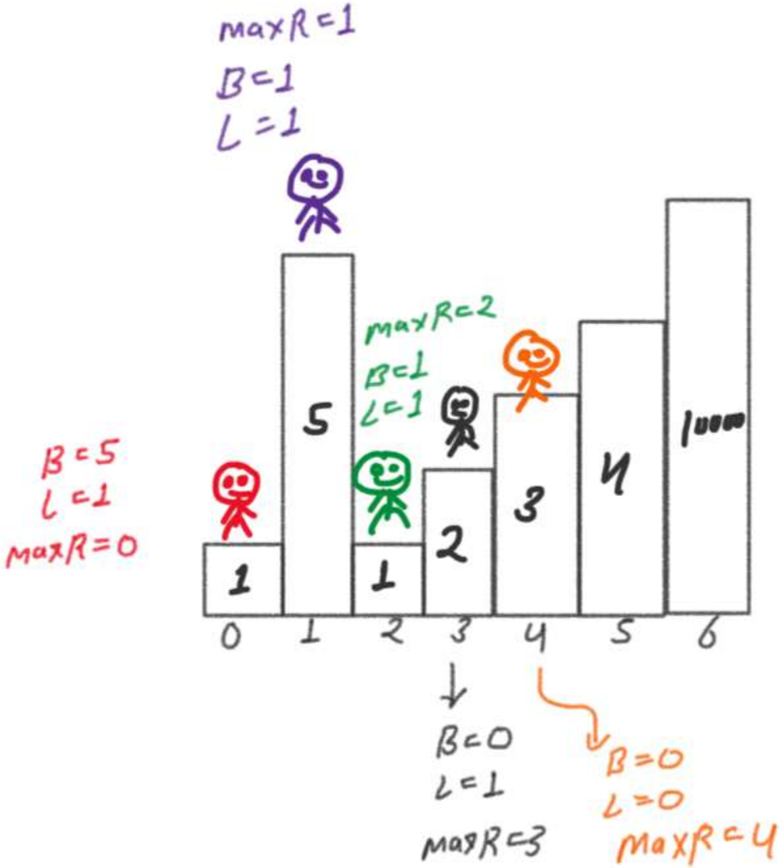


Explanation

Example 1:
Input: heights = [1,5,1,2,3,4,10000], bricks = 5, ladders = 1
Output: 5

Start with bricks uses:

MaxReach = 4

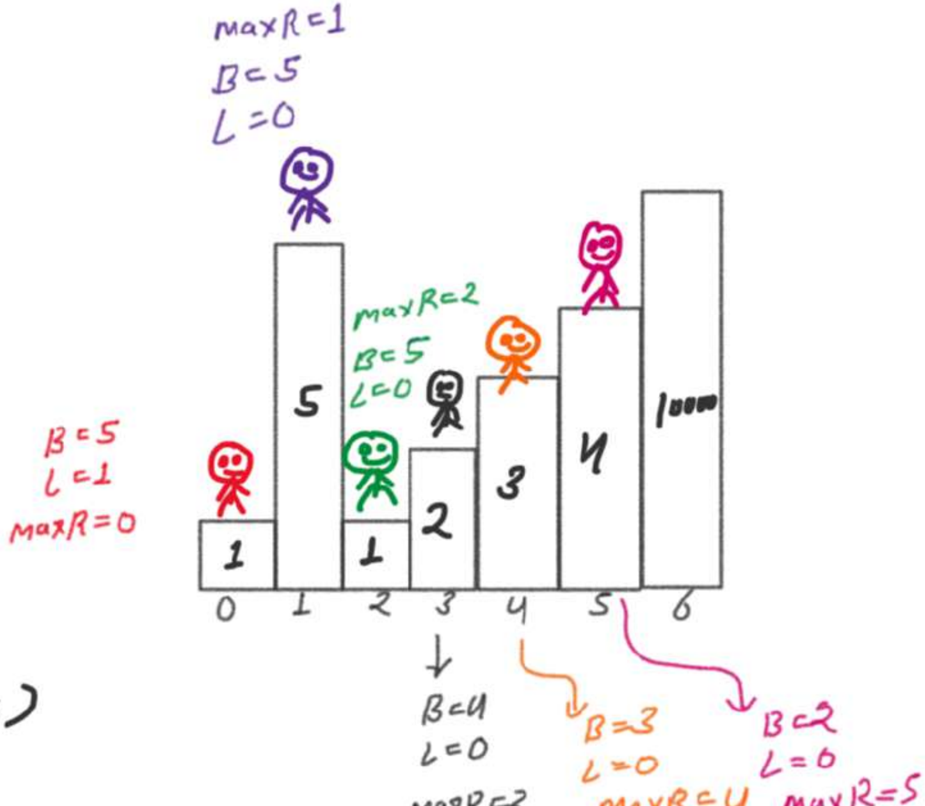


Example 1:

```
Input: heights = [1,5,1,2,3,4,10000], bricks = 5, ladders = 1
Output: 5
```

Start with ladders uses:

MaxReach = 5



=> max(start with Bricks, start with Ladders)

=> max(4, 5) => $\boxed{5}$ output

Intuition

First, I want to use the *all bricks* and if *bricks* are not sufficient then I will use the *ladders*.

```
Example 3:
Input: heights = [4,2,7,6,9,14,12], bricks = 5, ladders = 1
Output: 4
```

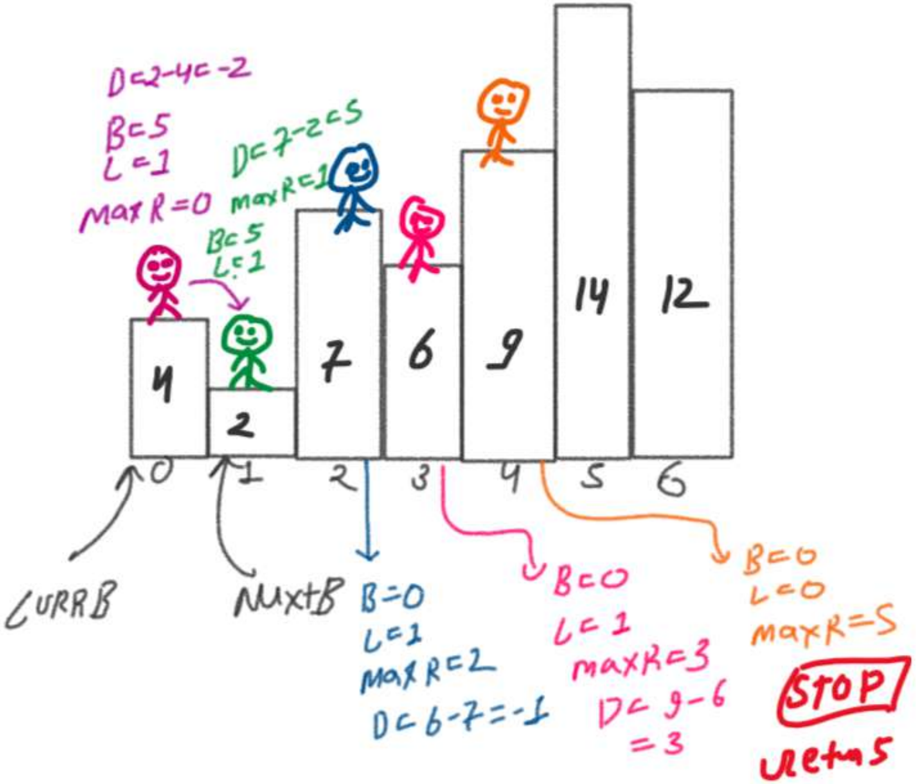
Approach 1

STEP1 Uses the All Brick first

STEP2 Uses the All ladders second

Note nextBH - currBH < 0

↳ Direct jump to nextB

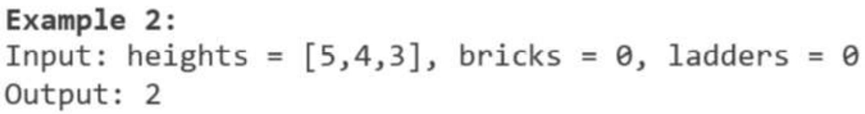


```
if( Bicks <=0 && Ladders <=0)
    ↳ return maxReach
if( Diff > Bricks && Ladder <=0)
    ↳ return maxReach
```

EDGE CASE

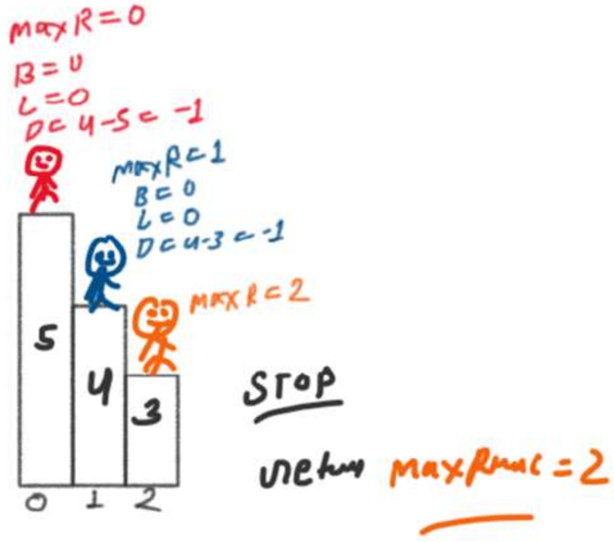
Traverse Loop from 0 to size-2

because of ArrayIndex out of Bound Error.



Example 2:

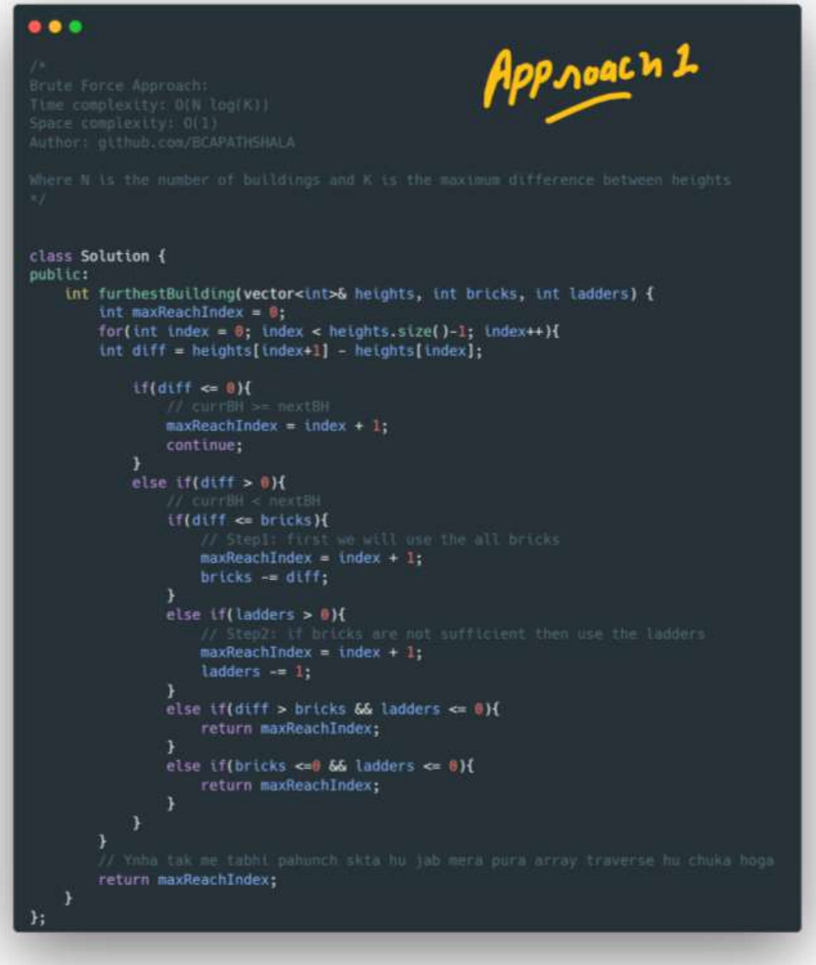
```
Input: heights = [5,4,3], bricks = 0, ladders = 0
Output: 2
```



Ex 3,2 are solved using Approach1

but not Ex 1 ☺?

↳ पहले Approach1 का code करते हैं।



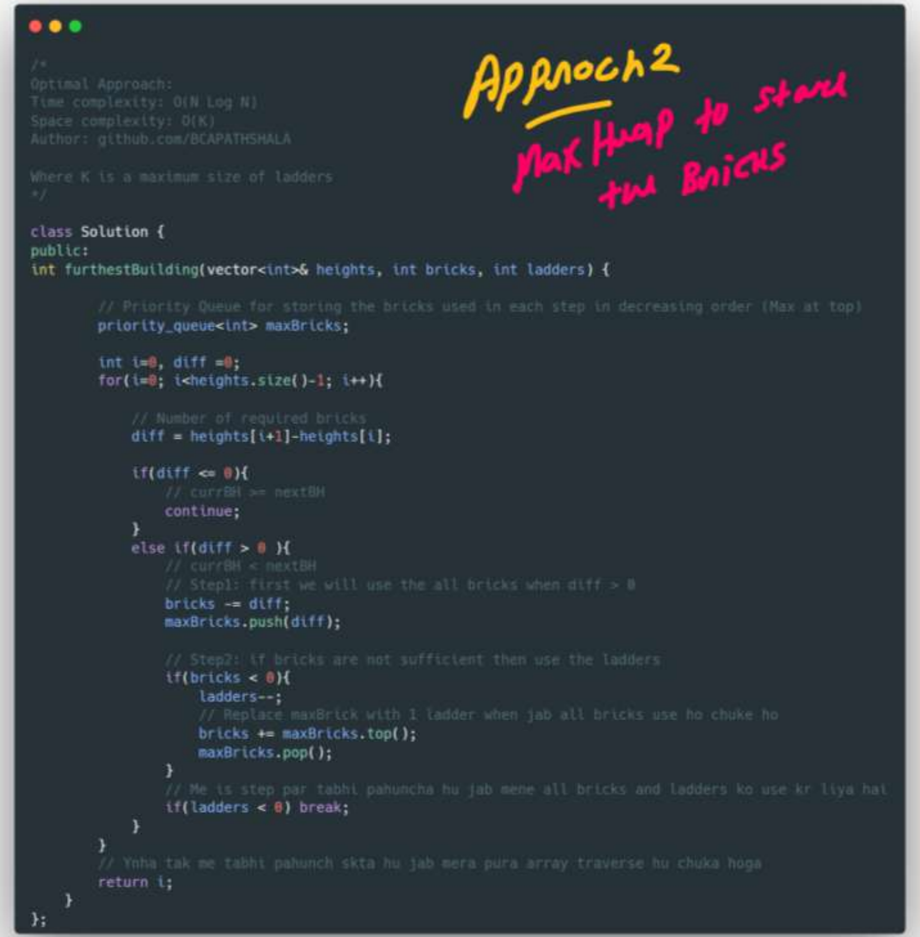
Approach 1

```
/*
Brute Force Approach:
Time complexity: O(N log(K))
Space complexity: O(1)
Author: github.com/BCAPATHSHALA

Where N is the number of buildings and K is the maximum difference between heights
*/


class Solution {
public:
    int furthestBuilding(vector<int>& heights, int bricks, int ladders) {
        int maxReachIndex = 0;
        for(int index = 0; index < heights.size()-1; index++){
        int diff = heights[index+1] - heights[index];

            if(diff <= 0){
                // currBH >= nextBH
                maxReachIndex = index + 1;
                continue;
            }
            else if(diff > 0){
                // currBH < nextBH
                if(diff <= bricks){
                    // Step1: first we will use the all bricks
                    maxReachIndex = index + 1;
                    bricks -= diff;
                }
                else if(ladders > 0){
                    // Step2: if bricks are not sufficient then use the ladders
                    maxReachIndex = index + 1;
                    ladders -= 1;
                }
                else if(diff > bricks && ladders <= 0){
                    return maxReachIndex;
                }
                else if(bricks <=0 && ladders <= 0){
                    return maxReachIndex;
                }
            }
        }
        // Ynha tak me tabhi pahunch skta hu jab mera pura array traverse hu chuka hoga
        return maxReachIndex;
    }
};
```



Approch2

Max Heap to store the Bricks

```
/*
Optimal Approach:
Time complexity: O(N Log N)
Space complexity: O(K)
Author: github.com/BCAPATHSHALA

Where K is a maximum size of ladders
*/

class Solution {
public:
int furthestBuilding(vector<int>& heights, int bricks, int ladders) {

        // Priority Queue for storing the bricks used in each step in decreasing order (Max at top)
        priority_queue<int> maxBricks;

        int i=0, diff =0;
        for(i=0; i<heights.size()-1; i++){

            // Number of required bricks
            diff = heights[i+1]-heights[i];

            if(diff <= 0){
                // currBH >= nextBH
                continue;
            }
            else if(diff > 0 ){
                // currBH < nextBH
                // Step1: first we will use the all bricks when diff > 0
                bricks -= diff;
                maxBricks.push(diff);

                // Step2: if bricks are not sufficient then use the ladders
                if(bricks < 0){
                    ladders--;
                    // Replace maxBrick with 1 ladder when jab all bricks use ho chuke ho
                    bricks += maxBricks.top();
                    maxBricks.pop();
                }
                // Me is step par tabhi pahuncha hu jab mene all bricks and ladders ko use kr liya hai
                if(ladders < 0) break;
            }
        }
        // Ynha tak me tabhi pahunch skta hu jab mera pura array traverse hu chuka hoga
        return i;
    }
};
```

why we max heaps?

Why use maxHeap to store the Bricks?

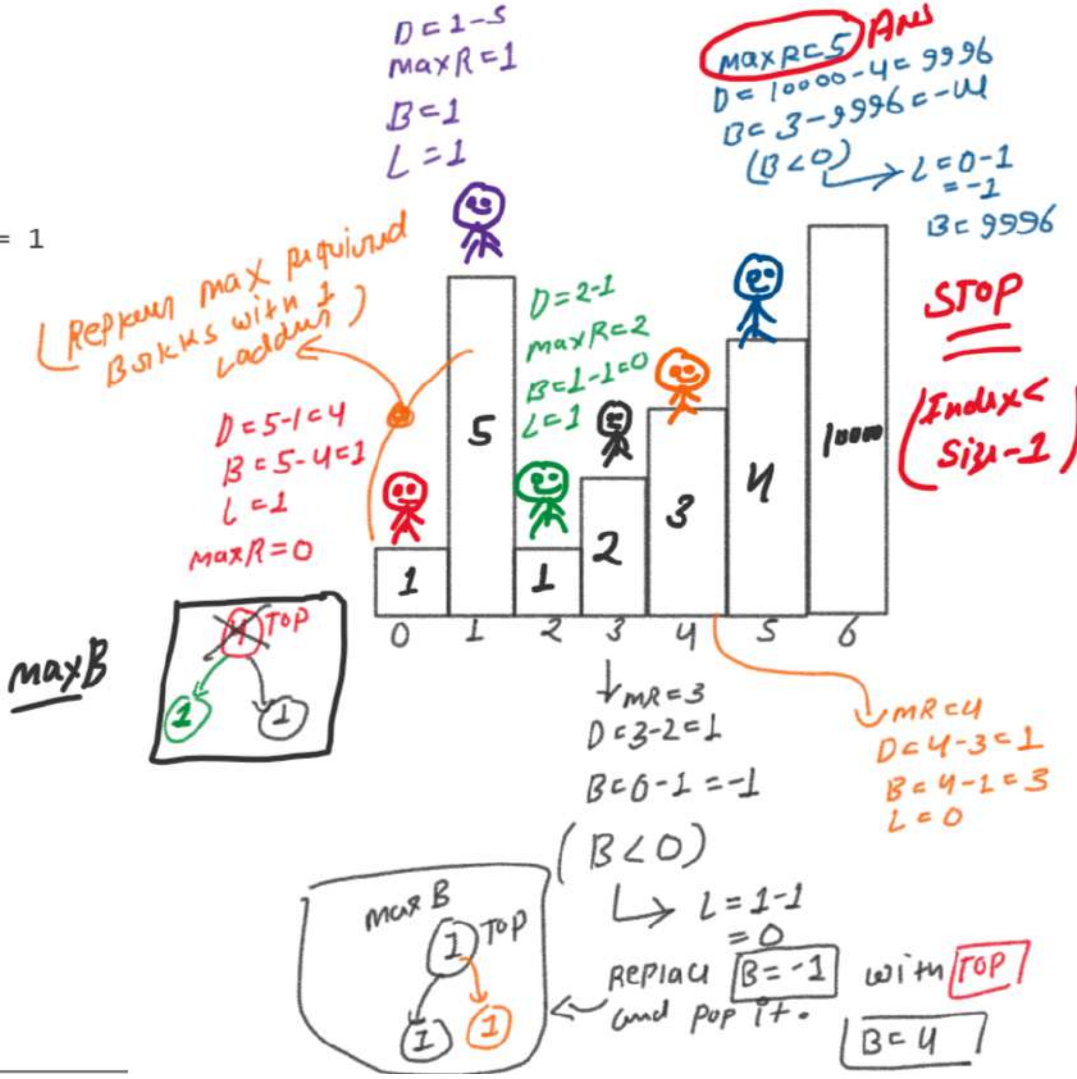
DRYRUN

Example 1:
Input: heights = [1,5,1,2,3,4,10000], bricks = 5, ladders = 1
Output: 5

Same Intuition

First, I want to use the *all bricks*
and if *bricks* are not sufficient then
I will use the *ladders*.

Start with bricks uses till bricks > 0 then
Uses the ladders one by one
but replace ladders with maxBricks on each time.



[DRY RUN itself on Different Test cases
↳ To Better understanding]