Experiment– 10

Student Name: Abhay Kumar **UID: 22BCS13112**
Branch: BE-CSE **Section: 22BCS_IOT-614 (B)**
Semester: 6th **Date of Performance:17/4/2025**
Subject: AP LAB 2 **Subject Code: 22CSP-351**

1. **Aim: solve the various problems.**

   a. **Easy:** Hamming Distance, Pascal's Triangle
   b. **Medium:** Valid Parenthesis String , Divide Two Integers
   c. **Hard:** Max number of tasks you can assign ,

2. **Algorithm:**

   a. **EASY1:** Pascal's Triangle
   - Initialize a 2D vector triangle.
   - For each row i from 0 to numRows - 1:
   - Create a row of size i + 1.
   - Set first and last elements to 1.
   - Fill middle elements using values from previous row.
   - Return the triangle.

   b. **Easy2:** Hamming Distance
   - Initialize a counter cnt to 0. This will store the number of differing bits.
   - While both x and y are not zero:
     Compare the least significant bits of x and y using (x & 1) ^ (y & 1).
     If the bits differ, increment cnt.
     Right shift both x and y by 1 bit.
   - If x still has remaining bits:
     For each set bit in x, increment cnt.
     Right shift x until it's 0.
   - If y still has remaining bits:
     For each set bit in y, increment cnt.
     Right shift y until it's 0.
   - Return cnt.

c. **Medium1:**Divide Two Integers

- Handle overflow case: if dividend = INT_MIN and divisor = -1, return INT_MAX.
- Record the sign of result based on dividend and divisor.
- Convert both numbers to long long and take their absolute values.
- Repeatedly subtract (or use bit shifts to speed up) the divisor from dividend and count how many
- Apply the sign to the result and return.

d. **Medium2:** Valid Parenthesis String

- Initialize two counters:
  leftMin → the **minimum** number of unmatched '(' that might still be open.
  leftMax → the **maximum** number of unmatched '(' that might still be open.
- Loop through each character c in the string:
  If c is '(':
  Increase both leftMin and leftMax by 1.
  If c is ')':
  Decrease both leftMin and leftMax by 1.
  If c is '*':
  '*' can be '(', ')', or empty:
  Decrease leftMin by 1 (assume it's ')')
  Increase leftMax by 1 (assume it's '(')
- If at any point leftMax becomes negative:
  Return false → too many closing brackets.
- If leftMin drops below zero:
  Reset leftMin to 0 → we can ignore extra closing by treating '*' as empty.
- After processing the entire string:
  If leftMin is 0, return true (all parentheses balanced)
  Else, return false.

e. **Hard:** Max Number of tasks you can assign

- Let n be the size of target.
- Create a map map to store the index of each element in target □ Sort the tasks in ascending order.
- Sort the workers in ascending order.
- Set the search range for binary search: low = 0, high = min(number of workers, number of tasks).
- Initialize ans to store the final answer.
- **Binary Search Loop**:
  While low <= high:
  Set mid = (low + high) / 2 → trying to assign mid number of tasks.
  Copy all workers into a multiset st to efficiently remove used workers.
  Set a counter count = 0 to track pills used.
  Set a flag flag = true to track if assignment is successful.
  For each task from hardest to easiest among mid tasks:
  Get the strongest available worker.
  If the worker can do the task without a pill, assign and remove the worker.
  Else, find the **weakest** worker that can do the task **with pill boost**.
  If such worker exists, assign task, remove worker, increment count.
  If not, set flag = false, break.
  If count > p, break and set flag = false.
  If assignment was successful (flag == true), store ans = mid and try a higher value (low = mid + 1).
  Else, try a smaller value (high = mid - 1).
- Return ans → maximum number of tasks that can be assigned.

.

3. Code:

a. Hamming Distance:

```
class Solution
{  public:
    int hamming Distance(int x, int
        y) { int cnt = 0;
        while(x && y)
            { if(x&1 ^ y&1)
                cnt++;       x >>= 1;       y >>= 1;
        }
        while(x) {
            if(x&1)   { cnt++; }
            x >>= 1;
        }
        while(y) {
            if(y&1)  { cnt++; }
            y >>= 1;
        }
        return cnt;
    }
};
```

b. **Pascal's Triangle:**

```
class Solution {
public:
vector<vector<int>> generate(int numRows) {
vector<vector<int>> res(numRows);
for (int i = 0; i < numRows; i++) {
 res[i].resize(i + 1, 1);
 long long ans = 1;  // Store intermediate values to avoid overflow
   for (int j = 1; j < i; j++) {
               ans = ans * (i - j + 1) / j;  // Using binomial coefficient formula
               res[i][j] = ans;
           }
       }
       return res;
   }
};
```

c. Divide Two Integers:

```
    int divide(int dividend, int divisor) {
    if (dividend == INT_MIN && divisor == -1)
        return INT_MAX;

    long long a = abs((long long)dividend);
    long long b = abs((long long)divisor);
    int result = 0;

    while (a >= b) {
        long long temp = b, multiple = 1;
        while (a >= (temp << 1)) {
            temp <<= 1;
            multiple <<= 1;
        }
        a -= temp;
        result += multiple;
    }

    return ((dividend > 0) ^ (divisor > 0)) ? -result : result;
}
```

d. Valid Parenthesis String:

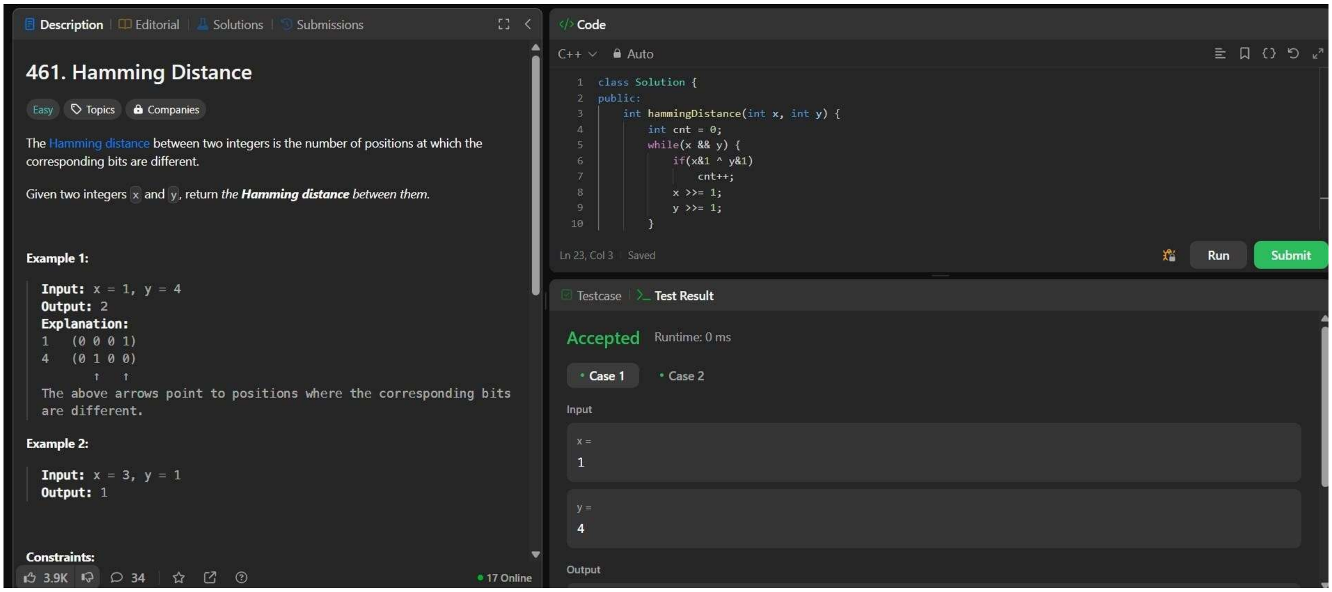
```
  class Solution
  { public:
            bool checkValidString(string
            s) { int leftMin = 0, leftMax =
            0; for (char c : s) {
              if (c == '(') {
                  leftMin++;
                  leftMax++;
              } else if (c == ')')
                  { leftMin--;
                  leftMax--;
              } else {
                  leftMin--;
                  leftMax++;
              }
              if (leftMax < 0) return false;
              if (leftMin < 0) leftMin = 0;
          }

          return leftMin == 0;
      }
  };
```

e. Max number of task you can assign

```
class Solution
{ public:
    int maxTaskAssign(vector<int>& tasks, vector<int>& workers, int p, int strength)
        { int n = tasks.size(), m = workers.size();
        sort(tasks.begin(), tasks.end());
        sort(workers.begin(), workers.end());
        int lo = 0, hi = min(m, n);
        int ans;
        while(lo <= hi) {
            int mid = lo + (hi - lo) / 2;
            int count = 0;
            bool flag = true;
            multiset<int> st(workers.begin(), workers.end());
            for(int i = mid - 1; i >= 0; i--) {
                auto it = prev(st.end());
                if(tasks[i] <= *it) {
                    st.erase(it);
                } else {
                    auto it = st.lower_bound(tasks[i] - strength);
                    if(it != st.end()) {
                        count++;
                        st.erase(it);
                    } else {
                        flag = false;
                        break;
                    }
                }
                if(count  >  p)
                    {    flag    =
                    false; break;
                }
            }
            if(flag) {
                ans = mid;
                lo = mid + 1;
            } else {
                hi = mid - 1;
            }}  return ans;  }};
```

4. Output:

Easy: Hamming Distance



461. Hamming Distance

Easy Topics Companies

The Hamming distance between two integers is the number of positions at which the corresponding bits are different.

Given two integers x and y, return *the* ***Hamming distance*** *between them.*

Example 1:

```
Input: x = 1, y = 4
Output: 2
Explanation:
1   (0 0 0 1)
4   (0 1 0 0)
       ↑   ↑
The above arrows point to positions where the corresponding bits
are different.
```

Example 2:

```
Input: x = 3, y = 1
Output: 1
```

Constraints:

```cpp
class Solution {
public:
    int hammingDistance(int x, int y) {
        int cnt = 0;
        while(x && y) {
            if(x&1 ^ y&1)
                cnt++;
            x >>= 1;
            y >>= 1;
        }
```

Ln 23, Col 3 Saved

Accepted Runtime: 0 ms

Case 1 Case 2

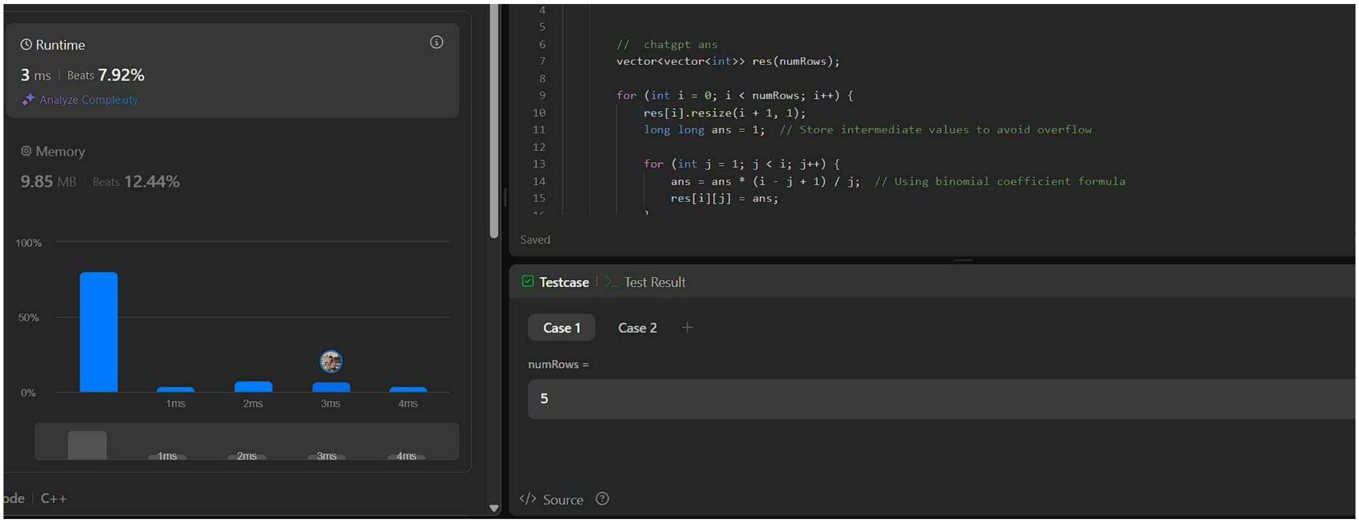
Input

x =
1

y =
4

Output

EASY : PASCAL TRIANGLE



3 ms | Beats 7.92%

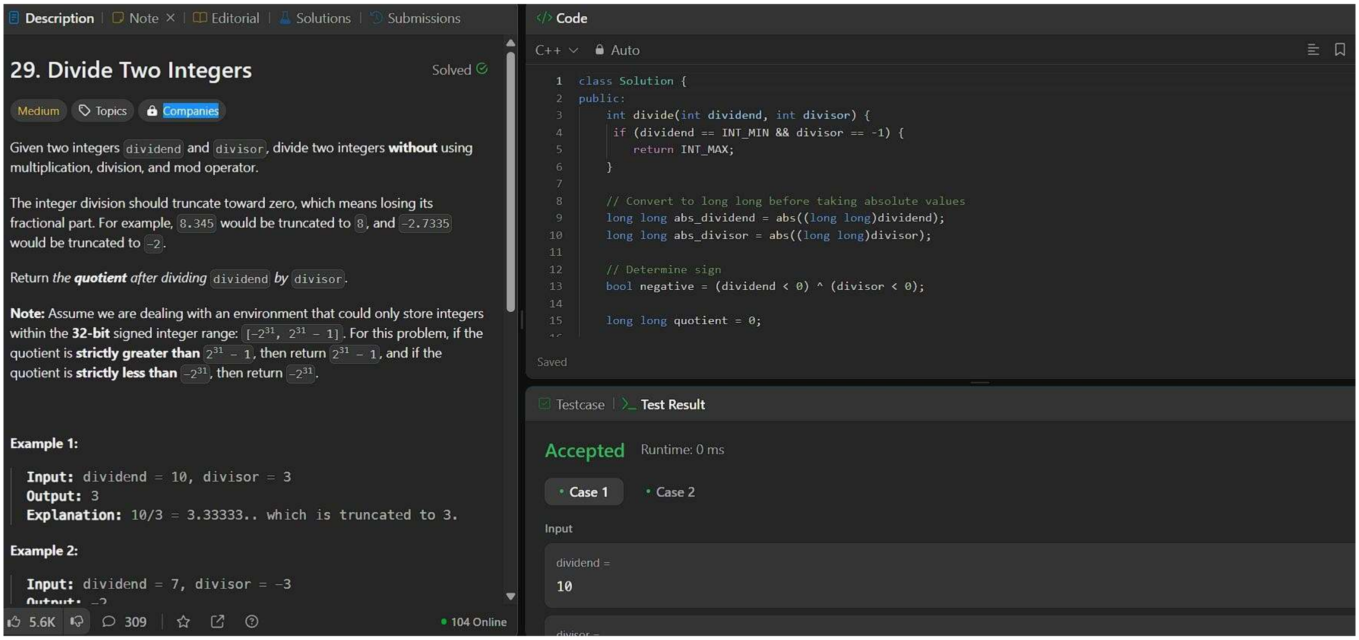
9.85 MB | Beats 12.44%

```cpp


        //  chatgpt ans
        vector<vector<int>> res(numRows);

        for (int i = 0; i < numRows; i++) {
            res[i].resize(i + 1, 1);
            long long ans = 1;  // Store intermediate values to avoid overflow

            for (int j = 1; j < i; j++) {
                ans = ans * (i - j + 1) / j;  // Using binomial coefficient formula
                res[i][j] = ans;
```

Saved

numRows =
5

Medium: Divide Two Integers



29. Divide Two Integers

Solved

Medium · Topics · Companies

Given two integers `dividend` and `divisor`, divide two integers **without** using multiplication, division, and mod operator.

The integer division should truncate toward zero, which means losing its fractional part. For example, `8.345` would be truncated to `8`, and `-2.7335` would be truncated to `-2`.

Return *the* ***quotient*** *after dividing* `dividend` *by* `divisor`.

Note: Assume we are dealing with an environment that could only store integers within the **32-bit** signed integer range: $[-2^{31}, 2^{31} - 1]$. For this problem, if the quotient is **strictly greater than** $2^{31} - 1$, then return $2^{31} - 1$, and if the quotient is **strictly less than** -2^{31}, then return -2^{31}.

Example 1:

```
Input: dividend = 10, divisor = 3
Output: 3
Explanation: 10/3 = 3.33333.. which is truncated to 3.
```

Example 2:

```
Input: dividend = 7, divisor = -3
Output: -2
```

```cpp
class Solution {
public:
    int divide(int dividend, int divisor) {
     if (dividend == INT_MIN && divisor == -1) {
        return INT_MAX;
    }

    // Convert to long long before taking absolute values
    long long abs_dividend = abs((long long)dividend);
    long long abs_divisor = abs((long long)divisor);

    // Determine sign
    bool negative = (dividend < 0) ^ (divisor < 0);

    long long quotient = 0;
```

Saved

Accepted Runtime: 0 ms

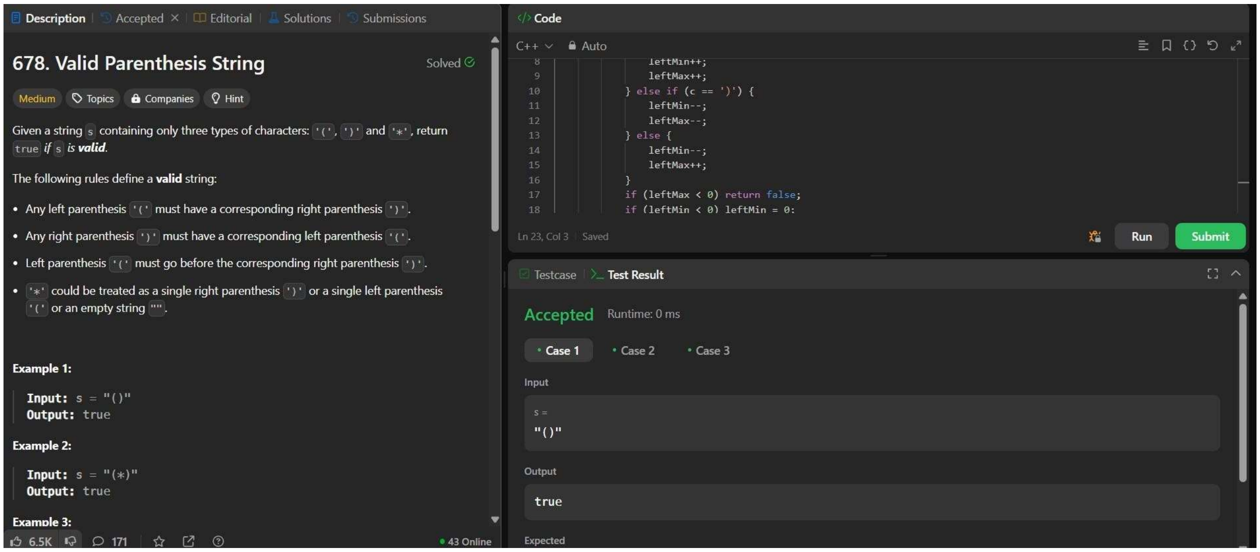
Case 1 · Case 2

Input

```
dividend =
10
```

```
divisor =
```

Medium: Valid parenthesis String


Description | Accepted × | Editorial | Solutions | Submissions
Code
C++
Auto
Run
Submit
Testcase | Test Result
6.5K
171
43 Online

678. Valid Parenthesis String

Solved

Medium · Topics · Companies · Hint

Given a string `s` containing only three types of characters: `'('`, `')'` and `'*'`, return `true` *if* `s` *is* ***valid***.

The following rules define a **valid** string:

- Any left parenthesis `'('` must have a corresponding right parenthesis `')'`.
- Any right parenthesis `')'` must have a corresponding left parenthesis `'('`.
- Left parenthesis `'('` must go before the corresponding right parenthesis `')'`.
- `'*'` could be treated as a single right parenthesis `')'` or a single left parenthesis `'('` or an empty string `""`.

Example 1:

```
Input: s = "()"
Output: true
```

Example 2:

```
Input: s = "(*)"
Output: true
```

Example 3:

```cpp
                leftMin++;
                leftMax++;
            } else if (c == ')') {
                leftMin--;
                leftMax--;
            } else {
                leftMin--;
                leftMax++;
            }
            if (leftMax < 0) return false;
            if (leftMin < 0) leftMin = 0:
```

Ln 23, Col 3 | Saved

Accepted Runtime: 0 ms

Case 1 · Case 2 · Case 3

Input

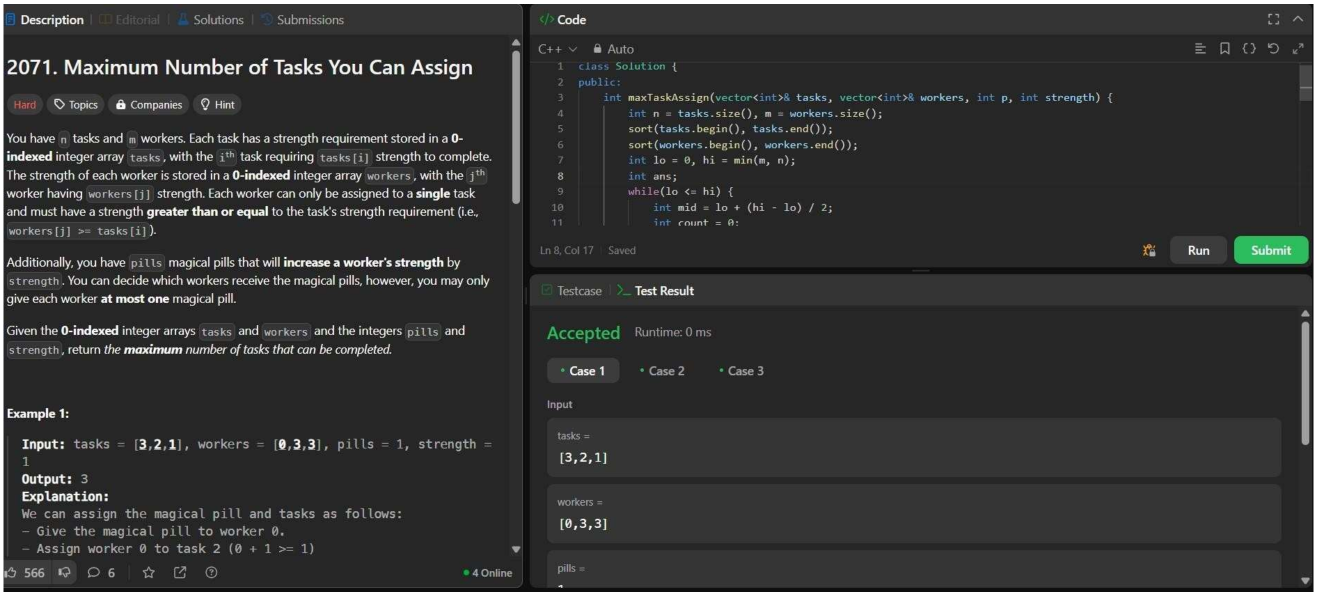
```
s =
"()"
```

Output

```
true
```

Expected

Hard: Max number of tasks you can assign



2071. Maximum Number of Tasks You Can Assign

You have `n` tasks and `m` workers. Each task has a strength requirement stored in a **0-indexed** integer array `tasks`, with the `ith` task requiring `tasks[i]` strength to complete. The strength of each worker is stored in a **0-indexed** integer array `workers`, with the `jth` worker having `workers[j]` strength. Each worker can only be assigned to a **single** task and must have a strength **greater than or equal** to the task's strength requirement (i.e., `workers[j] >= tasks[i]`).

Additionally, you have `pills` magical pills that will **increase a worker's strength** by `strength`. You can decide which workers receive the magical pills, however, you may only give each worker **at most one** magical pill.

Given the **0-indexed** integer arrays `tasks` and `workers` and the integers `pills` and `strength`, return *the **maximum** number of tasks that can be completed.*

Example 1:

```
Input: tasks = [3,2,1], workers = [0,3,3], pills = 1, strength =
1
Output: 3
Explanation:
We can assign the magical pill and tasks as follows:
- Give the magical pill to worker 0.
- Assign worker 0 to task 2 (0 + 1 >= 1)
```

```cpp
class Solution {
public:
    int maxTaskAssign(vector<int>& tasks, vector<int>& workers, int p, int strength) {
        int n = tasks.size(), m = workers.size();
        sort(tasks.begin(), tasks.end());
        sort(workers.begin(), workers.end());
        int lo = 0, hi = min(m, n);
        int ans;
        while(lo <= hi) {
            int mid = lo + (hi - lo) / 2;
            int count = 0;
```

Ln 8, Col 17 | Saved

Accepted Runtime: 0 ms

Case 1 Case 2 Case 3

Input

tasks =

```
[3,2,1]
```

workers =

```
[0,3,3]
```

pills =

```
1
```