Advanced Programming LAB II

ASSIGNMENT - 7

Submitted by,

Ankush Thakur | 22BCS11815

22BCS_ IOT-637 (A)

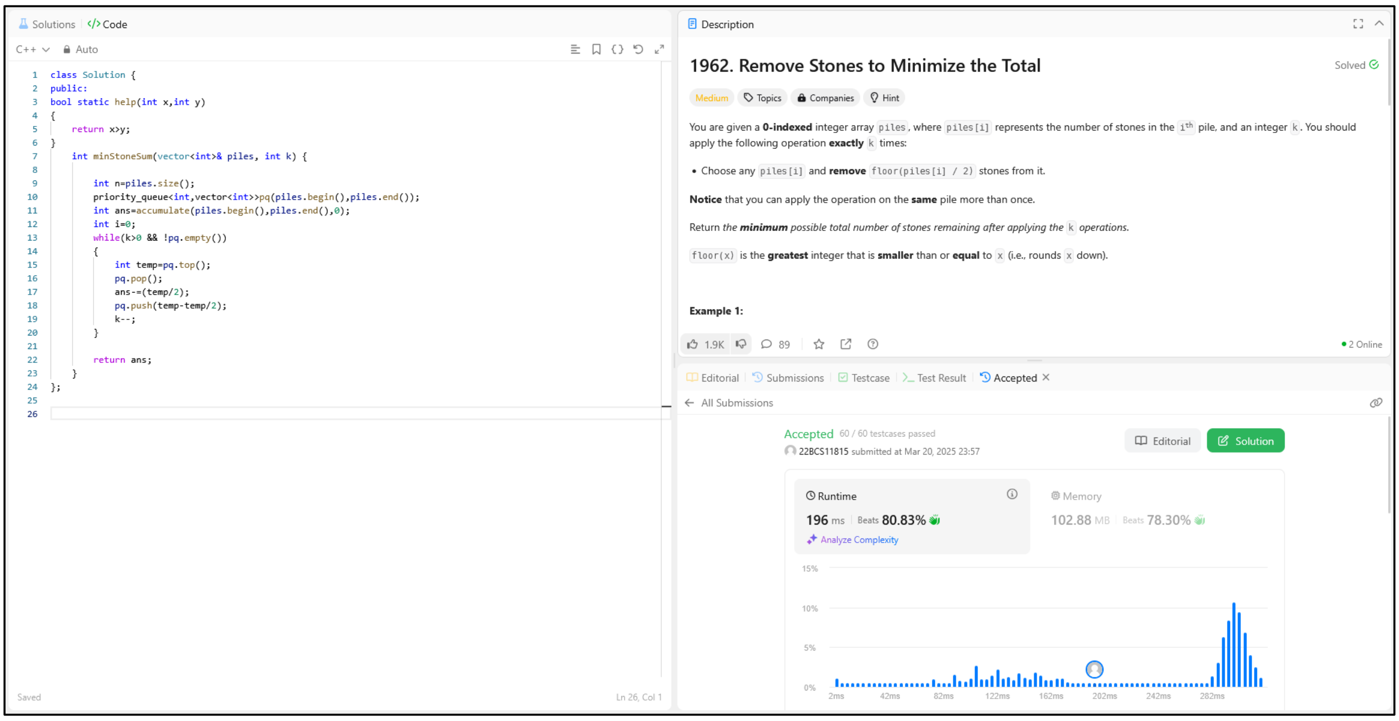
1962. Remove Stones to Minimize the Total

https://leetcode.com/problems/remove-stones-to-minimize-the-total/description

```cpp
class Solution {
public:
bool static help(int x,int y)
{
    return x>y;
}
    int minStoneSum(vector<int>& piles, int k) {

        int n=piles.size();
        priority_queue<int,vector<int>>pq(piles.begin(),piles.end());
        int ans=accumulate(piles.begin(),piles.end(),0);
        int i=0;
        while(k>0 && !pq.empty())
        {
            int temp=pq.top();
            pq.pop();
            ans-=(temp/2);
            pq.push(temp-temp/2);
            k--;
        }

        return ans;
    }
};
```



```cpp
class Solution {
public:
bool static help(int x,int y)
{
    return x>y;
}
    int minStoneSum(vector<int>& piles, int k) {

        int n=piles.size();
        priority_queue<int,vector<int>>pq(piles.begin(),piles.end());
        int ans=accumulate(piles.begin(),piles.end(),0);
        int i=0;
        while(k>0 && !pq.empty())
        {
            int temp=pq.top();
            pq.pop();
            ans-=(temp/2);
            pq.push(temp-temp/2);
            k--;
        }

        return ans;
    }
};
```

1962. Remove Stones to Minimize the Total

Solved

You are given a **0-indexed** integer array `piles`, where `piles[i]` represents the number of stones in the i^{th} pile, and an integer `k`. You should apply the following operation **exactly** `k` times:

- Choose any `piles[i]` and **remove** `floor(piles[i] / 2)` stones from it.

Notice that you can apply the operation on the **same** pile more than once.

Return *the **minimum** possible total number of stones remaining after applying the* `k` *operations.*

`floor(x)` is the **greatest** integer that is **smaller** than or **equal** to `x` (i.e., rounds `x` down).

Example 1:

Accepted 60 / 60 testcases passed

22BCS11815 submitted at Mar 20, 2025 23:57

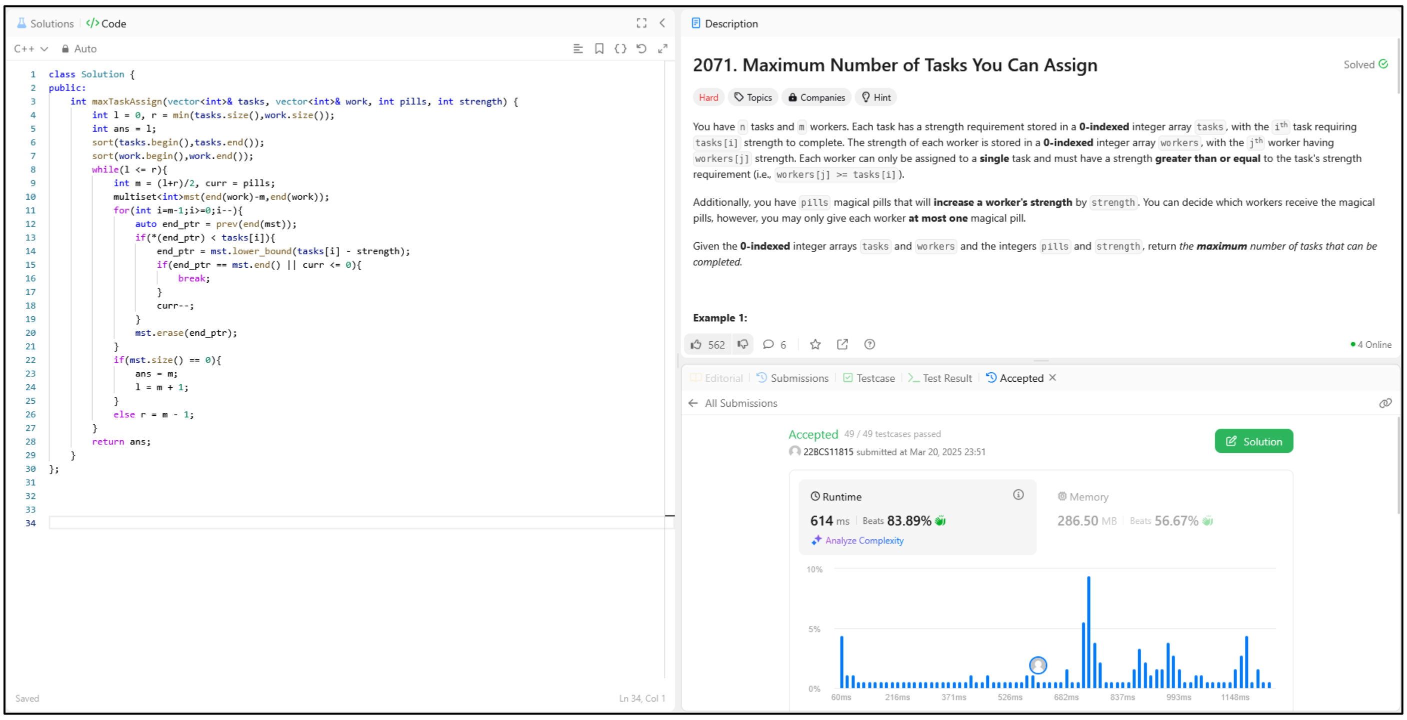
Runtime
196 ms | Beats 80.83%

Memory
102.88 MB | Beats 78.30%

2071. Maximum Number of Tasks You Can Assign

https://leetcode.com/problems/maximum-number-of-tasks-you-can-assign/description/

```cpp
class Solution {
public:
    int maxTaskAssign(vector<int>& tasks, vector<int>& work, int pills, int strength) {
        int l = 0, r = min(tasks.size(),work.size());
        int ans = l;
        sort(tasks.begin(),tasks.end());
        sort(work.begin(),work.end());
        while(l <= r){
            int m = (l+r)/2, curr = pills;
            multiset<int>mst(end(work)-m,end(work));
            for(int i=m-1;i>=0;i--){
                auto end_ptr = prev(end(mst));
                if(*(end_ptr) < tasks[i]){
                    end_ptr = mst.lower_bound(tasks[i] - strength);
                    if(end_ptr == mst.end() || curr <= 0){
                        break;
                    }
                    curr--;
                }
                mst.erase(end_ptr);
            }
            if(mst.size() == 0){
                ans = m;
                l = m + 1;
            }
            else r = m - 1;
        }
        return ans;
    }
};
```



```cpp
class Solution {
public:
    int maxTaskAssign(vector<int>& tasks, vector<int>& work, int pills, int strength) {
        int l = 0, r = min(tasks.size(),work.size());
        int ans = l;
        sort(tasks.begin(),tasks.end());
        sort(work.begin(),work.end());
        while(l <= r){
            int m = (l+r)/2, curr = pills;
            multiset<int>mst(end(work)-m,end(work));
            for(int i=m-1;i>=0;i--){
                auto end_ptr = prev(end(mst));
                if(*(end_ptr) < tasks[i]){
                    end_ptr = mst.lower_bound(tasks[i] - strength);
                    if(end_ptr == mst.end() || curr <= 0){
                        break;
                    }
                    curr--;
                }
                mst.erase(end_ptr);
            }
            if(mst.size() == 0){
                ans = m;
                l = m + 1;
            }
            else r = m - 1;
        }
        return ans;
    }
};
```

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 `i`th task requiring `tasks[i]` strength to complete. The strength of each worker is stored in a **0-indexed** integer array `workers`, with the `j`th 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:

Accepted 49 / 49 testcases passed
22BCS11815 submitted at Mar 20, 2025 23:51

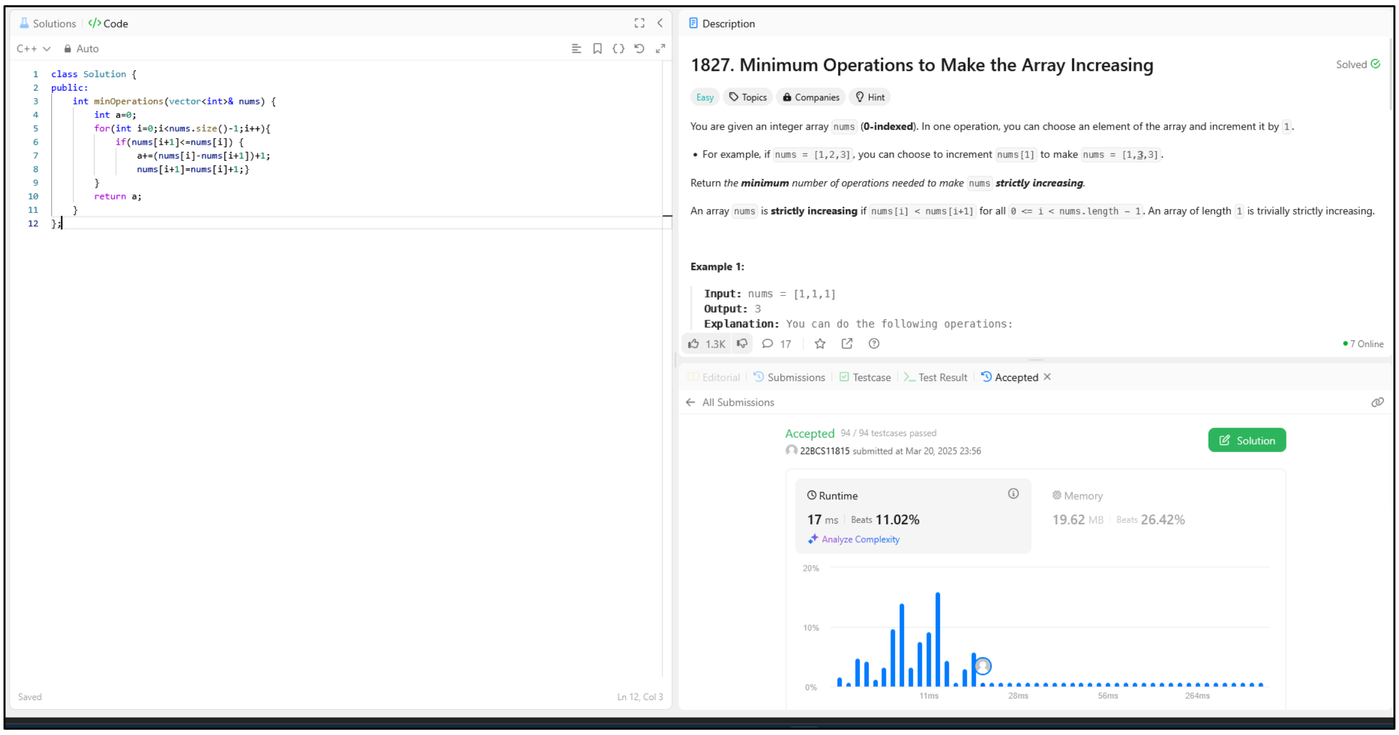
Runtime
614 ms | Beats 83.89%

Memory
286.50 MB | Beats 56.67%

1827. Minimum Operations to Make the Array Increasing

https://leetcode.com/problems/minimum-operations-to-make-the-array-increasing/description

```cpp
class Solution {
public:
    int maxTaskAssign(vector<int>& tasks, vector<int>& work, int pills, int strength) {
        int l = 0, r = min(tasks.size(),work.size());
        int ans = l;
        sort(tasks.begin(),tasks.end());
        sort(work.begin(),work.end());
        while(l <= r){
            int m = (l+r)/2, curr = pills;
            multiset<int>mst(end(work)-m,end(work));
            for(int i=m-1;i>=0;i--){
                auto end_ptr = prev(end(mst));
                if(*(end_ptr) < tasks[i]){
                    end_ptr = mst.lower_bound(tasks[i] - strength);
                    if(end_ptr == mst.end() || curr <= 0){
                        break;
                    }
                    curr--;
                }
                mst.erase(end_ptr);
            }
            if(mst.size() == 0){
                ans = m;
                l = m + 1;
            }
            else r = m - 1;
        }
        return ans;
    }
};
```



```cpp
class Solution {
public:
    int minOperations(vector<int>& nums) {
        int a=0;
        for(int i=0;i<nums.size()-1;i++){
            if(nums[i+1]<=nums[i]) {
                a+=(nums[i]-nums[i+1])+1;
                nums[i+1]=nums[i]+1;}
        }
        return a;
    }
};
```

1827. Minimum Operations to Make the Array Increasing

You are given an integer array `nums` (**0-indexed**). In one operation, you can choose an element of the array and increment it by `1`.

- For example, if `nums = [1,2,3]`, you can choose to increment `nums[1]` to make `nums = [1,3,3]`.

Return *the **minimum** number of operations needed to make* `nums` ***strictly increasing**.*

An array `nums` is **strictly increasing** if `nums[i] < nums[i+1]` for all `0 <= i < nums.length - 1`. An array of length `1` is trivially strictly increasing.

Example 1:

```
Input: nums = [1,1,1]
Output: 3
Explanation: You can do the following operations:
```

Accepted 94 / 94 testcases passed
22BCS11815 submitted at Mar 20, 2025 23:56

Runtime 17 ms | Beats 11.02%

Memory 19.62 MB | Beats 26.42%