**[Maximum Number of Tasks You Can Assign](https://leetcode.com/problems/maximum-number-of-tasks-you-can-assign/)**

class Solution {

public:

bool canAssign(vector<int>& tasks, vector<int>& workers, int pills, int strength, int mid) {

multiset<int> availableWorkers; // Multiset keeps workers sorted

for (int i = workers.size() - mid; i < workers.size(); i++) {

availableWorkers.insert(workers[i]);

}

int pillsLeft = pills;

// Assign mid hardest tasks

for (int i = mid - 1; i >= 0; i--) {

int task = tasks[i];

auto it = availableWorkers.lower\_bound(task);

if (it != availableWorkers.end()) {

availableWorkers.erase(it); // Assign worker without pill

}

else if (pillsLeft > 0) {

// No direct match, find weakest worker that can be boosted

it = availableWorkers.lower\_bound(task - strength);

if (it == availableWorkers.end()) return false; // No valid worker

availableWorkers.erase(it);

pillsLeft--;

}

else return false; // Task cannot be assigned

}

return true;

}

int maxTaskAssign(vector<int>& tasks, vector<int>& workers, int pills, int strength) {

sort(tasks.begin(), tasks.end()); // Sort tasks in increasing order

sort(workers.begin(), workers.end()); // Sort workers in increasing order

int left = 0, right = min(tasks.size(), workers.size()), ans = 0;

while (left <= right) {

int mid = left + (right - left) / 2;

if (canAssign(tasks, workers, pills, strength, mid)) {

ans = mid; // If mid tasks can be assigned, try more

left = mid + 1;

} else {

right = mid - 1; // Reduce number of tasks

}

}

return ans;

}

};

**Minimum Operations to Make the Array Increasing**

class Solution {

public:

    int minOperations(vector<int>& nums) {

        int operations = 0;

        for (int i = 1; i < nums.size(); i++) {

            if (nums[i] <= nums[i - 1]) {

                int increment = nums[i - 1] - nums[i] + 1;

                nums[i] += increment;

                operations += increment;

            }

        }

        return operations;

    }

};

[**Remove Stones to Minimize the Total**](https://leetcode.com/problems/remove-stones-to-minimize-the-total/)

class Solution {

public:

int minStoneSum(vector<int>& piles, int k) {

priority\_queue<int> maxHeap(piles.begin(), piles.end());

while (k-- > 0) {

int top = maxHeap.top();

maxHeap.pop();

top -= top / 2;

maxHeap.push(top);

}

int total = 0;

while (!maxHeap.empty()) {

total += maxHeap.top();

maxHeap.pop();

}

return total;

}

};