Maximum Units on a Truck

```
class Solution {
public:
static bool myfunction(vector<int>& a, vector<int>& b){
return a[1] > b[1];
}
int maximumUnits(vector<vector<int>>& boxTypes, int truckSize) {

sort(boxTypes.begin(),boxTypes.end(),myfunction);

int ans=0;
for(auto box: boxTypes){
int x=min(box[0],truckSize);
ans+=(x*box[1]);
truckSize-=x;
if(!truckSize) break;
}
return ans;
}
};
```

Minimum Operations to Make the Array Increasing

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

Remove Stones to Minimize the Total

```
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;
    }
};
```

Maximum Score From Removing Substrings

```
class Solution {
    void getCount(string str, string sub, int& cnt1, int& cnt2) {

        char first = sub[0], second = sub[1];
        int i = 1;
        while(i < str.length()) {
            if(i > 0 && str[i-1] == first && str[i] == second) {
```

```
            cnt1++;
            str.erase(i-1, 2);
            i--;
            continue;
        }
        i++;
    }

    i = 1;
    while(i < str.length()) {
        if(i > 0 && str[i-1] == second && str[i] == first) {
            cnt2++;
            str.erase(i-1, 2);
            i--;
            continue;
        }
        i++;
    }
    return;
  }
public:
  int maximumGain(string s, int x, int y) {

    int mxABcnt = 0;
    int mxBAcnt = 0;
    int minBAcnt = 0;
    int minABcnt= 0;
```

```
        getCount(s, "ab", mxABcnt, minBAcnt);
        getCount(s, "ba", mxBAcnt, minABcnt);

        int operation1 = mxABcnt * x + minBAcnt * y;
        int operation2 = mxBAcnt * y + minABcnt * x;
        return max(operation1, operation2);
    }
};
```

Minimum Operations to Make a Subsequence

```
class Solution {
public:
    int minOperations(vector<int>& target, vector<int>& arr) {
        unordered_map<int, int> mp;
        for(int i = 0; i < target.size(); i++) mp[target[i]] = i;
        vector<int> v;
        for(int a: arr) if (mp.count(a)) v.push_back(mp[a]);
        int n = v.size(), ans = 0;
        vector<int> tail(n + 1, INT_MAX);
        tail[0] = INT_MIN;
        for(int a: v) {
            int b = upper_bound(tail.begin(), tail.begin() + min(ans + 1, n), a) - tail.begin();
            if (b == 0 || (tail[b - 1] < a && tail[b] > a)) {
                tail[b] = a;
```

```
            ans = max(ans, b);
        }
    }
    return target.size() - ans;
  }
};
```

Maximum Number of Tasks You Can Assign

```
class Solution {
public:
  bool check(vector<int>& tasks, vector<int>& workers, int pills, int strength,int index)
  {
    multiset<int> st;
    for(auto it:workers)
    {
      st.insert(it);
    }
    for(int i=index-1;i>=0;i--)
    {
      auto it=st.lower_bound(tasks[i]);
      if(it!=st.end())
      {
        st.erase(it);
      }
      else
      {
```

```
            if(pills<=0)
            {
                return false;
            }
            else
            {
                it=st.lower_bound(tasks[i]-strength);
                if(it!=st.end())
                {
                    st.erase(it);
                    pills--;
                }
                else
                {
                    return false;
                }
            }
        }
    }
    return true;
}
int maxTaskAssign(vector<int>& tasks, vector<int>& workers, int pills, int strength) {
    sort(tasks.begin(),tasks.end());
    sort(workers.begin(),workers.end());
    int low=0;
    int high=min(workers.size(),tasks.size());
    while(low<high)
```

```
        {
            int mid=(low+high+1)/2;
            if(check(tasks,workers,pills,strength,mid)==true)
            {
                low=mid;
            }
            else
            {
                high=mid-1;
            }
        }
        return high;
    }
};
```