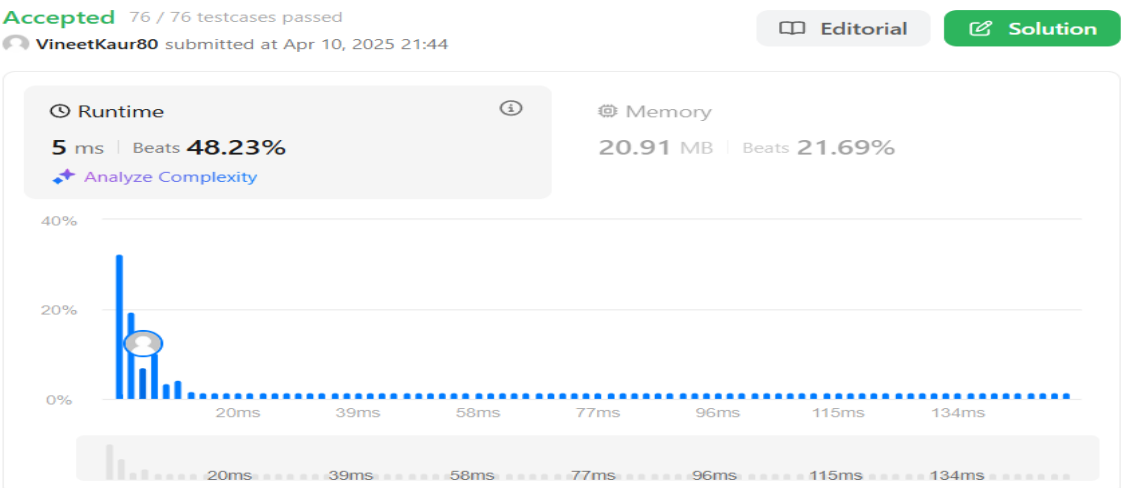
# AP 8th EXPERIMENT

## Max Units on a Truck

1. **CODE:**

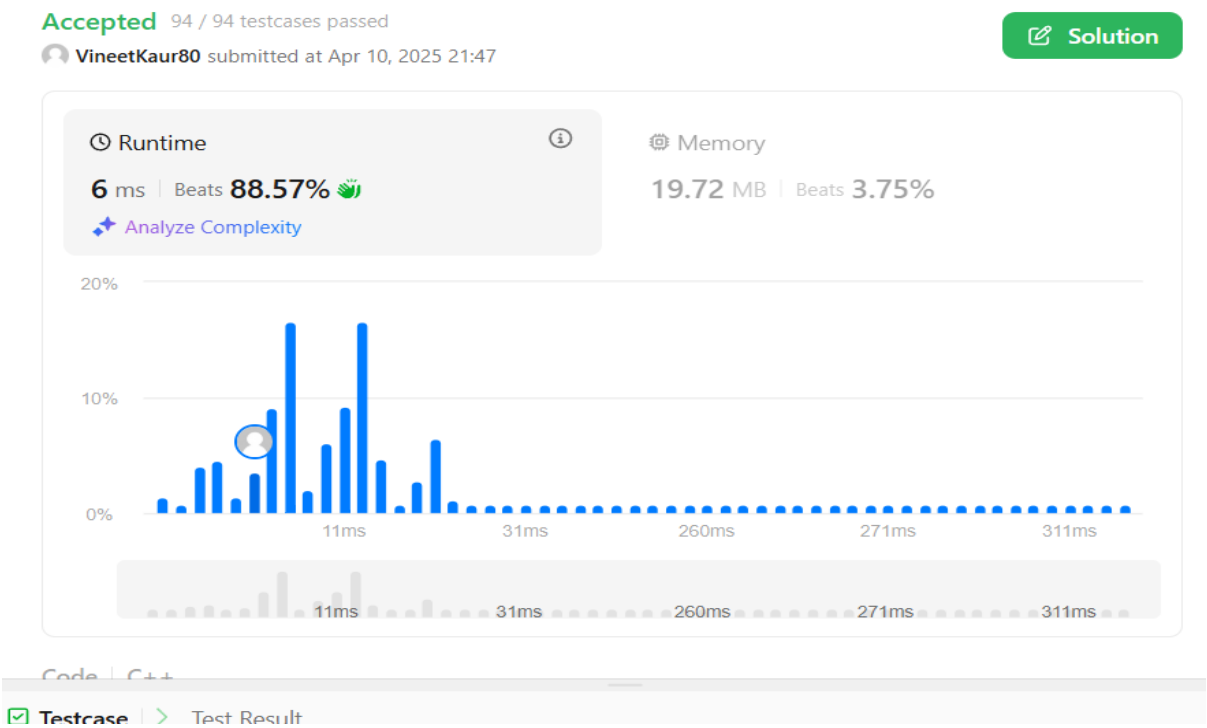
```
class Solution {
public:
    int maximumUnits(vector<vector<int>>& boxTypes, int truckSize) {
        sort(boxTypes.begin(),boxTypes.end(),[](const vector<int>& a,const vector<int>&b){
            return a[1]>b[1];
        });
        int maxi=0;

        for(auto num:boxTypes){
            if(truckSize==0) break;

            int box=min(num[0],truckSize);
            maxi +=box*num[1];
            truckSize -= box;
        }
    return maxi;
    }
};
```



## 2. Min Operations to Make Array Increasing

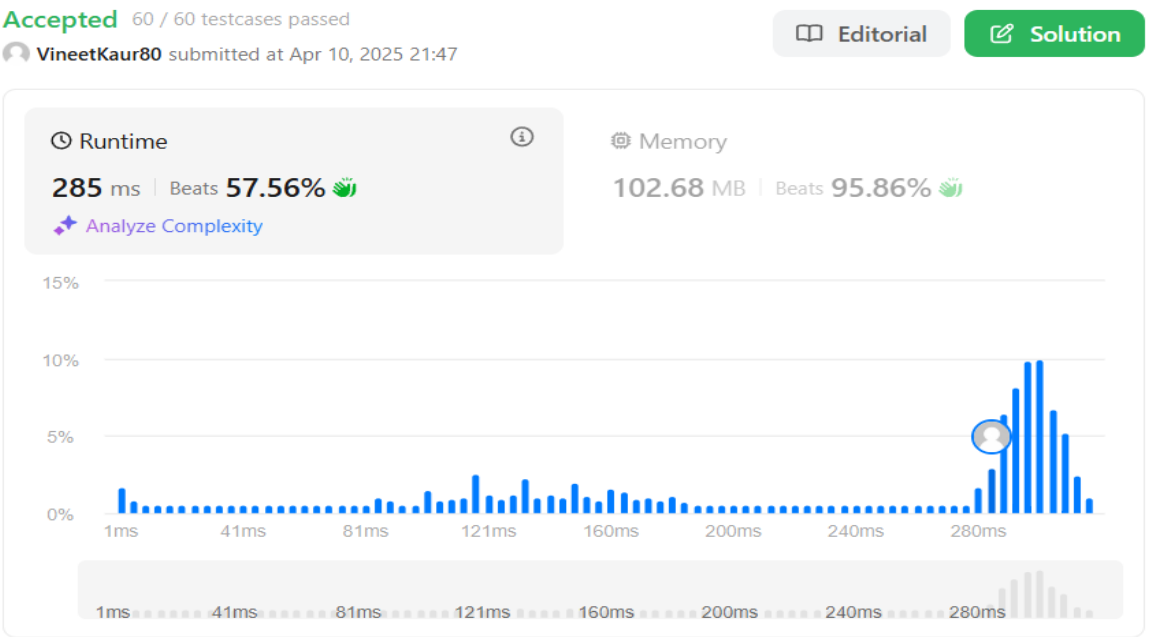
**CODE:**

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



### 3. Remove Stones to Maximize Total

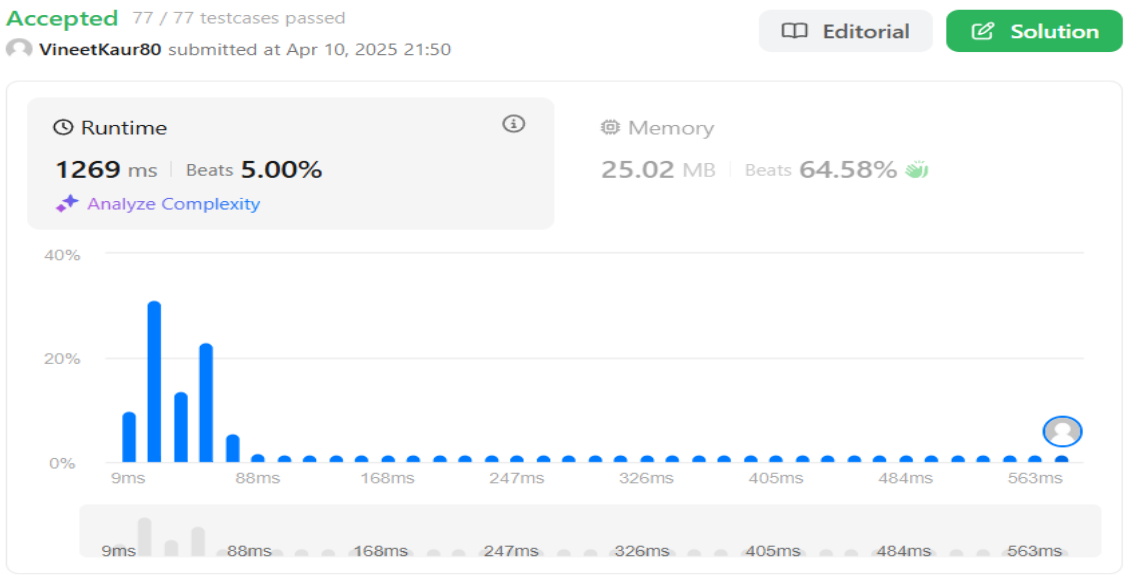
```
class Solution {
public:
    int minStoneSum(vector<int>& piles, int k) {
        priority_queue <int> pq ( piles.begin(), piles.end() );
        int stoneSum = 0;
        while( k-- ){
            int biggest = pq.top();
            pq.pop();
            pq.push( biggest - floor( biggest / 2 ) );
        }
        while( !pq.empty() ){
            stoneSum += pq.top();
            pq.pop();
        }
        return stoneSum;
    }
};
```



## 4. Max 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);
    }
};
```




### 5. Min Operations to Make a Subsequence

```
class Solution {
 public:
  //  seen
  int minOperations(vector<int>& t, vector<int>& a) {
    vector<int> lis = {};

    unordered_map<int, int> idxMap;
```

```
    for (int i = 0; i < t.size(); i++) idxMap[t[i]] = i;

    for (int i = 0; i < a.size(); i++) {
      if (idxMap.count(a[i]))
        a[i] = idxMap[a[i]];
      else
        a[i] = -1;
    }

    for (int i = 0; i < a.size(); i++) {
      if (a[i] == -1) continue;

      if (lis.size() == 0)
        lis.push_back(a[i]);
      else {
        if (lis.back() < a[i])
          lis.push_back(a[i]);
        else {
          int lb = lower_bound(lis.begin(), lis.end(), a[i]) - lis.begin();
          lis[lb] = a[i];
        }
      }
    }

    return (int)(t.size() - lis.size());
  }
};
```


## 6. Max 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;
  }
};
```