# Assingment-05

**Student Name: Nikhil Kumar Tiwari** **UID: 22BCS10471**

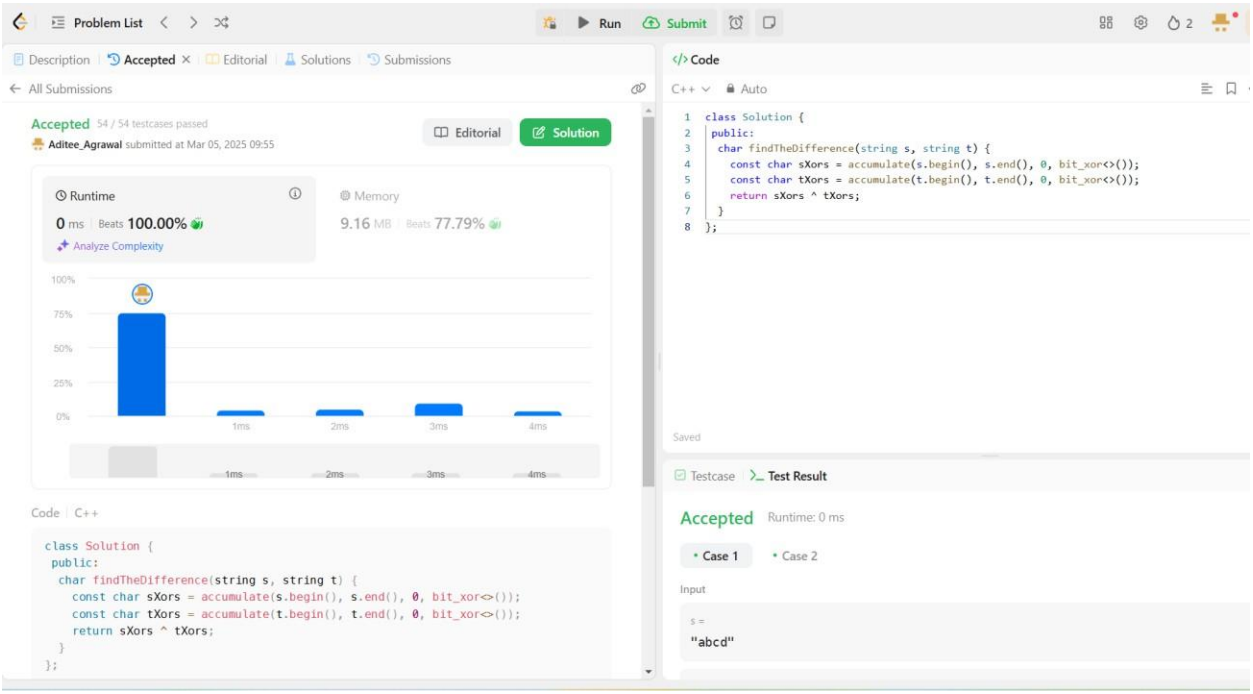
**Branch: BE-CSE** **Section/Group: 22BCS-IOT-FL-601 / A**

**Semester: 6th** **Subject Code: 22CSP-351**

**Subject Name: Advanced Programming lab - 2**

## 1. .Find the differnce

```
class Solution {
 public:
  char findTheDifference(string s, string t) {
    const char sXors = accumulate(s.begin(), s.end(), 0, bit_xor<>());
    const char tXors = accumulate(t.begin(), t.end(), 0, bit_xor<>());
    return sXors ^ tXors;
  }
};
```



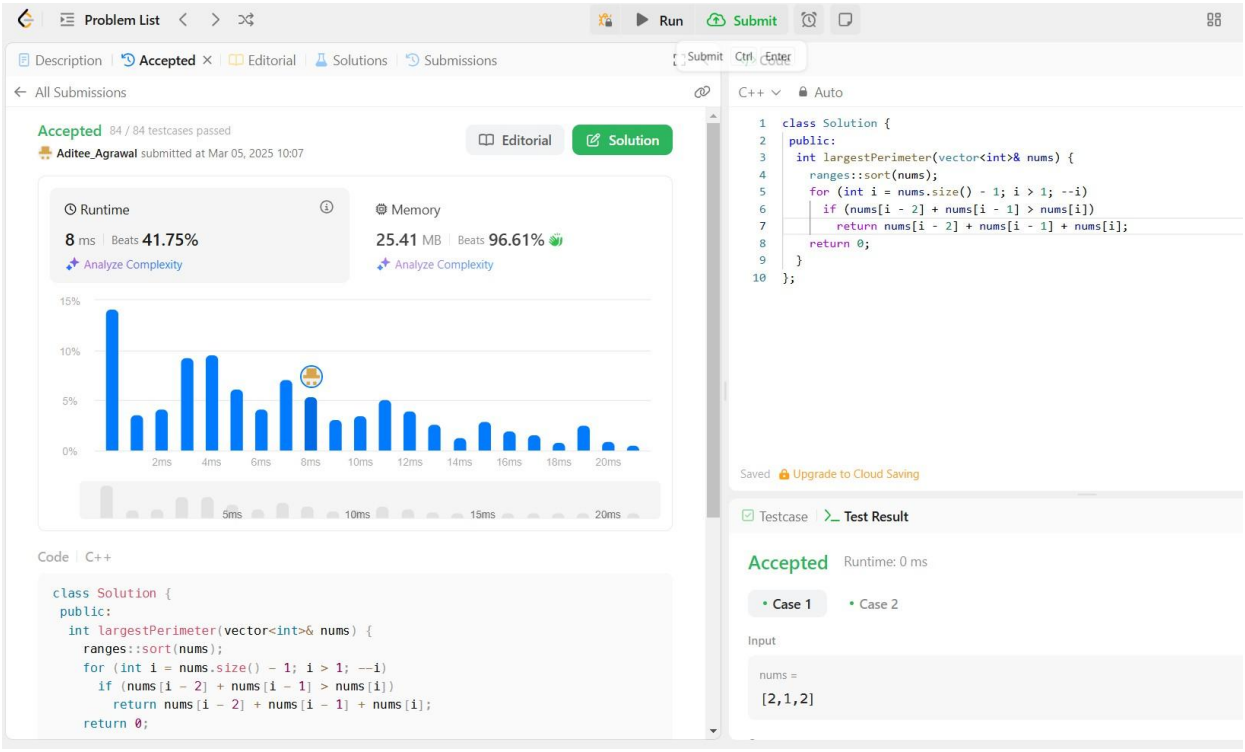

## 2. Largest Perimeter Triangle

```
class Solution {
 public:
  int largestPerimeter(vector<int>& nums) {
    ranges::sort(nums);

    for (int i = nums.size() - 1; i > 1; --i)
```

```
      if (nums[i - 2] + nums[i - 1] > nums[i])
        return nums[i - 2] + nums[i - 1] + nums[i];

    return 0;
  }
};
```



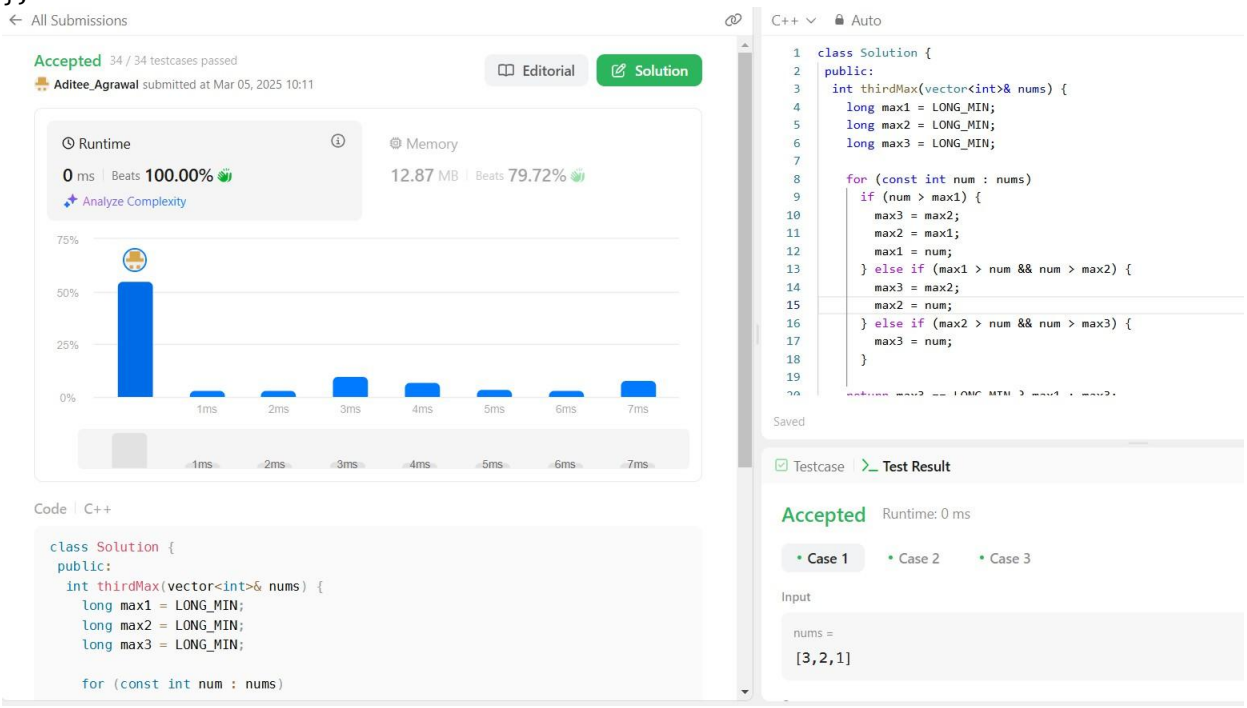

3. [Third Maximum Number]

```
class Solution {
 public:
  int thirdMax(vector<int>& nums) {
    long max1 = LONG_MIN;
    long max2 = LONG_MIN;
    long max3 = LONG_MIN;

    for (const int num : nums)
      if (num > max1) {
        max3 = max2;
        max2 = max1;
        max1 = num;
      } else if (max1 > num && num > max2) {
        max3 = max2;
        max2 = num;
      } else if (max2 > num && num > max3) {
        max3 = num;
```

```
    }

    return max3 == LONG_MIN ? max1 : max3;
  }
};
```

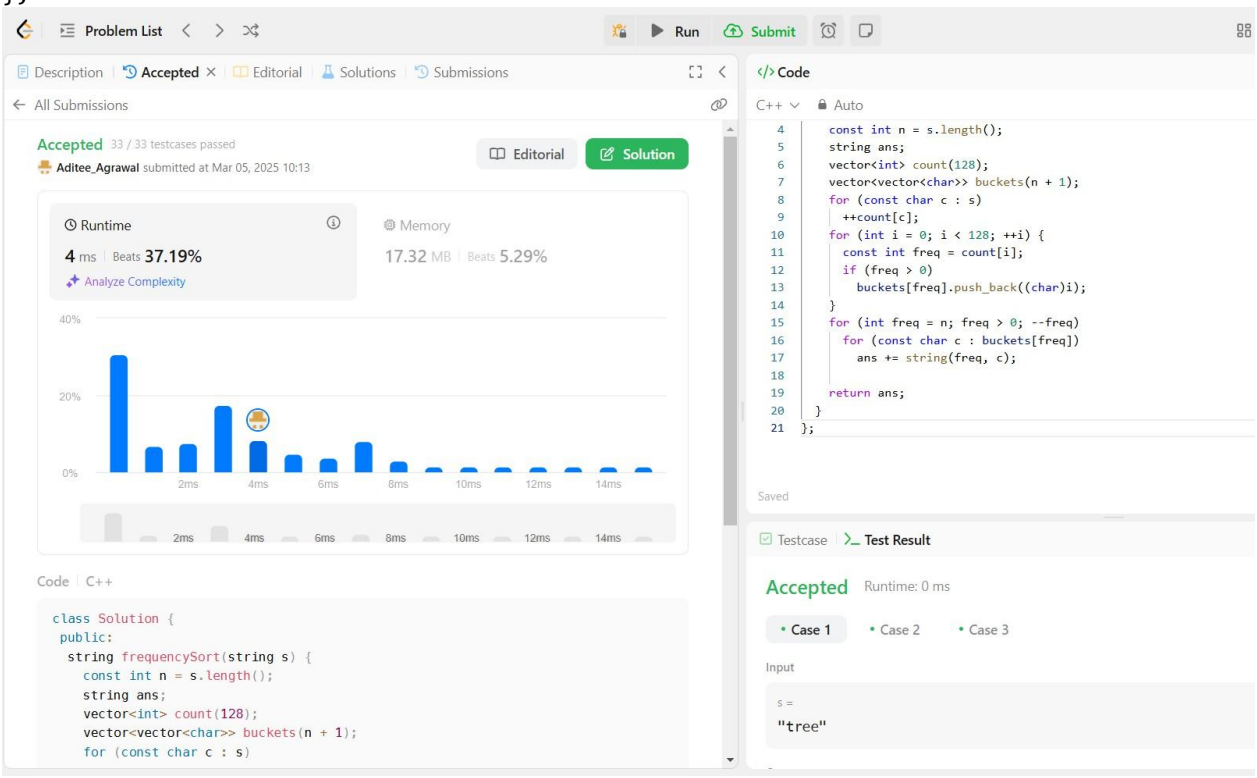



## 4. Sort Characters By Frequency

```
class Solution {
 public:
  string frequencySort(string s) {
    const int n = s.length();
    string ans;
    vector<int> count(128);
    vector<vector<char>> buckets(n + 1);
    for (const char c : s)
      ++count[c];
    for (int i = 0; i < 128; ++i) {
      const int freq = count[i];
      if (freq > 0)
        buckets[freq].push_back((char)i);
    }
    for (int freq = n; freq > 0; --freq)
      for (const char c : buckets[freq])
        ans += string(freq, c);

    return ans;
  }
```

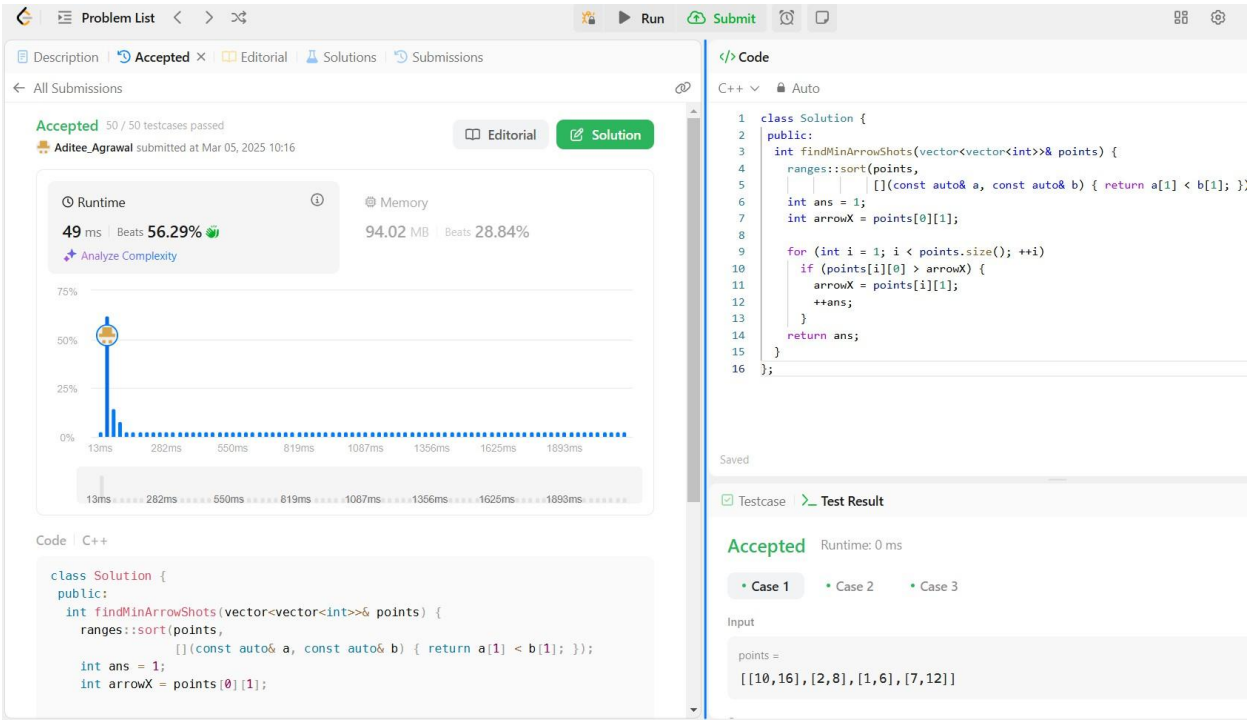
};




## 5. [Minimum Number of Arrows to Burst Balloons]

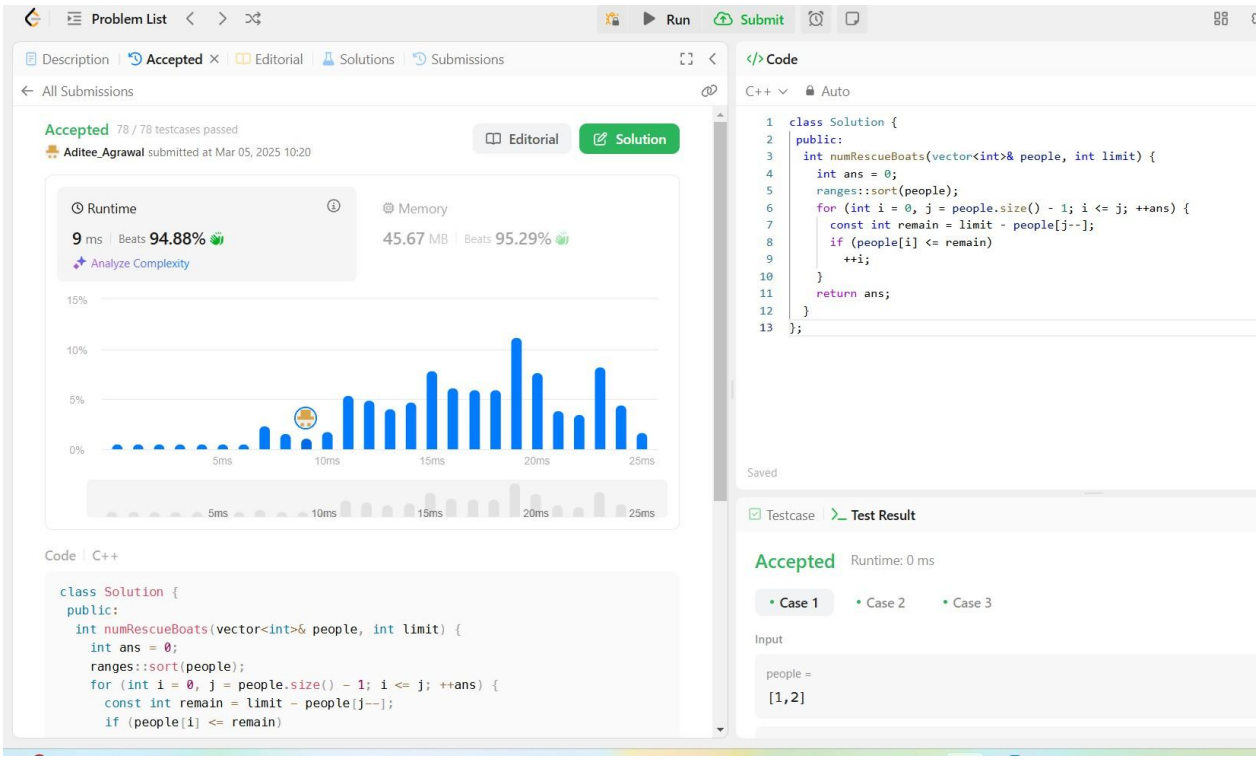
```
class Solution {
 public:
  int findMinArrowShots(vector<vector<int>>& points) {
    ranges::sort(points,
                 [](const auto& a, const auto& b) { return a[1] < b[1]; });
    int ans = 1;
    int arrowX = points[0][1];

    for (int i = 1; i < points.size(); ++i)
      if (points[i][0] > arrowX) {
        arrowX = points[i][1];
        ++ans;
      }
    return ans;
  }
};
```




## 6. Boats to Save People

```
class Solution {
 public:
  int numRescueBoats(vector<int>& people, int limit) {
    int ans = 0;
    ranges::sort(people);
    for (int i = 0, j = people.size() - 1; i <= j; ++ans) {
      const int remain = limit - people[j--];
      if (people[i] <= remain)
        ++i;
    }
    return ans;
  }
};
```




## 7. [K Closest Points to Origin]

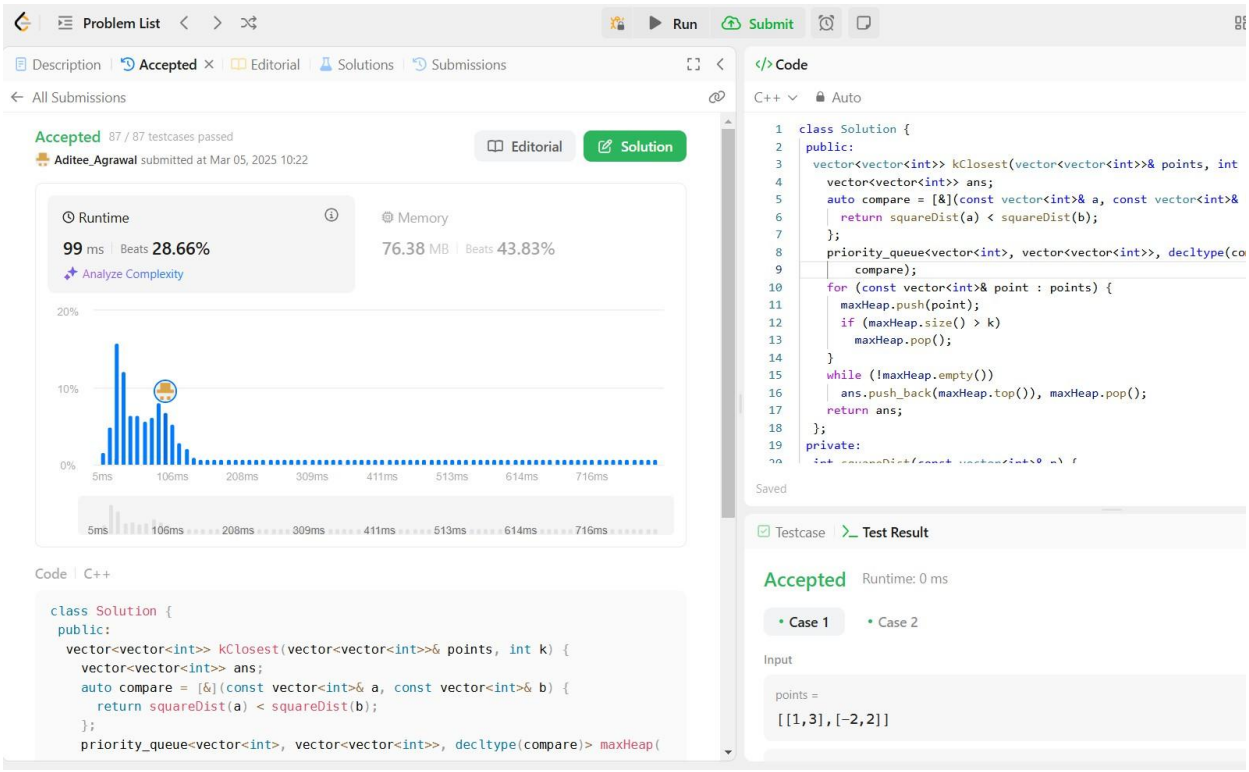
```
class Solution {
 public:
  vector<vector<int>> kClosest(vector<vector<int>>& points, int k) {
    vector<vector<int>> ans;
    auto compare = [&](const vector<int>& a, const vector<int>& b) {
      return squareDist(a) < squareDist(b);
    };
    priority_queue<vector<int>, vector<vector<int>>, decltype(compare)> maxHeap(
        compare);

    for (const vector<int>& point : points) {
      maxHeap.push(point);
      if (maxHeap.size() > k)
        maxHeap.pop();
    }

    while (!maxHeap.empty())
      ans.push_back(maxHeap.top()), maxHeap.pop();

    return ans;
  };
```

```
 private:
  int squareDist(const vector<int>& p) {
    return p[0] * p[0] + p[1] * p[1];
  }
};
```




## 8. [Reduce Array Size to The Half]

```
class Solution {
 public:
  int minSetSize(vector<int>& arr) {
    const int n = arr.size();
    int sum = 0;
    unordered_map<int, int> count;
    vector<pair<int, int>> numAndFreqs;
    for (const int a : arr)
      ++count[a];
    for (const auto& [a, freq] : count)
      numAndFreqs.emplace_back(a, freq);
    ranges::sort(
        numAndFreqs, ranges::greater{},
```

```
            [](const pair<int, int>& numAndFreq) { return
numAndFreq.second; });
        for (int i = 0; i < numAndFreqs.size(); ++i) {
          sum += numAndFreqs[i].second;
          if (sum >= n / 2)
            return i + 1;
        }
        throw;
      }
};
```