Find Peak Element

```
class Solution {
public:
    int findPeakElement(vector<int>& nums) {
        int left=0, right= nums.size()-1;

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

            if(nums[mid]>nums[mid+1]){
                right=mid;
            }
            else{
                left=mid+1;
            }
        }
        return left;
    }
};
```

Search in Rotated Sorted Array

```
class Solution {
public:
    int search(vector<int>& nums, int target) {
        int left = 0, right = nums.size() - 1;

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

            if (nums[mid] == target) {
                return mid;
            }
            if (nums[left] <= nums[mid]) {
                if (nums[left] <= target && target < nums[mid]) {
                    right = mid - 1;
                } else {
                    left = mid + 1;
                }
            }
            else {
                if (nums[mid] < target && target <= nums[right]) {
                    left = mid + 1;
                } else {
                    right = mid - 1;
                }
            }
        }

        return -1;
    }
};
```

Count and Say

```
class Solution {
public:
    string countAndSay(int n) {
        string currentTerm = "1";
        while(--n){
            string nextTerm= "";

            for(int i=0; i<currentTerm.size();){
                int countIndex=i;
                while(countIndex<currentTerm.size()&& currentTerm[countIndex]==currentTerm[i]){
                    ++countIndex;
                }
                nextTerm+=to_string(countIndex - i);
                nextTerm+=currentTerm[i];
                i=countIndex;
            }
            currentTerm = nextTerm;
        }
        return currentTerm;
    }
};
```

Number of Substrings Containing All Three Characters

```
class Solution {
public:
    int numberOfSubstrings(string s) {
        int lastSeenPositions[3]={-1,-1,-1};
        int substringCount=0;
        for(int index=0;index<s.size();++index){
            lastSeenPositions[s[index] - 'a'] = index;
            int minLastSeenPosition = min(lastSeenPositions[0],min(lastSeenPositions[1], lastSeenPositions[2])) + 1;
            substringCount += minLastSeenPosition;
        }
        return substringCount;
    }
};
```

Koko Eating Bananas

```
class Solution {
public:
    int minEatingSpeed(vector<int>& piles, int h) {
        int l=1;
        int r=1e9;
        while(l<r){
            int mid=l+(r-l) /2;
            int hours=0;

            for (int pile:piles){
                hours+=(pile+mid-1)/mid;
            }
            if(hours<=h){
                r=mid;
            }
            else{
                l=mid+1;
            }
        }
        return l;
    }
};
```

Group Anagrams

```
#include <vector>
#include <string>
#include <unordered_map>
#include <algorithm>

class Solution {
public:
    vector<vector<string>> groupAnagrams(vector<string>& strs) {
        unordered_map<string, vector<string>> anagramGroups;
                for (auto& str : strs) {
                    string key = str;
                    sort(key.begin(), key.end());
                    anagramGroups[key].emplace_back(str);
                }
                vector<vector<string>> groupedAnagrams;
                for (auto& pair : anagramGroups) {
```

```
                groupedAnagrams.emplace_back(pair.second);
            }
        return groupedAnagrams;
    }
};
```

Destroying Asteroids

```
#include <vector>
#include <algorithm>
using namespace std;

class Solution {
public:
    bool asteroidsDestroyed(int mass, vector<int>& asteroids) {
        sort(asteroids.begin(), asteroids.end());
        long long currentMass = mass;
        for (int asteroidMass : asteroids) {
            if (currentMass < asteroidMass) {
                return false;
            }
            currentMass += asteroidMass;
        }
        return true;
    }
};
```

Majority Element II

```
class Solution {
public:
    vector<int> majorityElement(vector<int>& nums) {
        int count1 = 0, count2 = 0;
        int candidate1 = 0, candidate2 = 1;
        for (int num : nums) {
            if (num == candidate1) {
                ++count1;
            } else if (num == candidate2) {
                ++count2;
            } else if (count1 == 0) {
                candidate1 = num;
                count1 = 1;
            } else if (count2 == 0) {
                candidate2 = num;
                count2 = 1;
            } else {
                --count1;
                --count2;
            }
        }

        std::vector<int> result;
        if (std::count(nums.begin(), nums.end(), candidate1) > nums.size() / 3) {
            result.push_back(candidate1);
        }
        if (candidate1 != candidate2 && std::count(nums.begin(), nums.end(), candidate2) > nums.size() / 3) {
            result.push_back(candidate2);
        }

        return result;
    }
};
```

Trapping Rain Water

```
class Solution:
    def trap(self, height: List[int]) -> int:
        left, right = 0, len(height) - 1
        left_max, right_max = 0, 0
```

```
water = 0

while left < right:
    if height[left] < height[right]:
        if height[left] >= left_max:
            left_max = height[left]
        else:
            water += left_max - height[left]
        left += 1
    else:
        if height[right] >= right_max:
            right_max = height[right]
        else:
            water += right_max - height[right]
        right -= 1

return water
```