

1763.Longest Nice Substring

```
class Solution {
public:
    string longestNiceSubstring(string s) {
        string ans="";
        for(int i=0;i<s.length();i++){
            int count=0;
            string temp="";
            vector<bool> a(26,0),b(26,0);
            for(int j=i;j<s.length();j++){
                temp.push_back(s[j]);
                if(s[j]>='A' && s[j]<='Z'){
                    if(a[s[j]-'A']==0 && b[s[j]-'A']==0)
                        count++;
                    else if(a[s[j]-'A'] && !b[s[j]-'A'])
                        count--;
                    b[s[j]-'A']=1;
                }
                else{
                    if(b[s[j]-'a']==0 && a[s[j]-'a']==0)
                        count++;
                    else if(b[s[j]-'a'] && !a[s[j]-'a'])
                        count--;
                    a[s[j]-'a']=1;
                }
            }
            if(ans.size()<temp.size() && count==0)
                ans=temp;
        }
    }
}
```

```

    }

    return ans;

}

};

```

The screenshot displays the LeetCode submission page for the problem "Longest Nice Substring". The submission is accepted, with a runtime of 0 ms and memory usage of 9.03 MB. The code is written in C++ and implements a solution that iterates through the string, maintaining a count of characters and a temporary string to build the longest nice substring. The test result for Case 1 shows the input "YazaAay" and the output "aAa".

Code:

```

1 class Solution {
2 public:
3     string longestNiceSubstring(string s) {
4         string ans="";
5         for(int i=0;i<s.length();i++){
6             int count=0;
7             string temp="";
8             vector<bool> a(26,0),b(26,0);
9             for(int j=i;j<s.length();j++){
10                 temp.push_back(s[j]);
11                 if(s[j]>='A' && s[j]<='Z'){
12                     if(a[s[j]-'A']==0 && b[s[j]-'A']==0)
13                         continue;
14                 }
15             }
16             ans=temp;
17         }
18     }
19 }

```

Runtime: 0 ms | Beats 100.00%
Memory: 9.03 MB | Beats 90.60%

Test Result: Accepted | Runtime: 0 ms
Case 1: Input: "YazaAay", Output: "aAa"

190.Reverse Bits

```

class Solution {
public:
    uint32_t reverseBits(uint32_t n) {
        uint32_t ans=0;
        for(int i=0;i<32;i++){
            if((n>>i)&1)
                ans+=(1<<(31-i));
        }
    }
}

```

```

        return ans;
    }
};

```

The screenshot displays the LeetCode interface for the 'Reverse Bits' problem. The left sidebar shows the problem is 'Accepted' with 600/600 testcases passed. The main area shows the C++ code for the solution, which uses a loop to reverse the bits of a 32-bit integer. The runtime is 0 ms (100.00% beats) and memory is 7.58 MB (99.07% beats). A bar chart shows the runtime distribution. The right sidebar shows the test case result for 'Case 1', which is 'Accepted' with a runtime of 2 ms. The input is a 32-bit integer with a specific bit pattern, and the output is the reversed bit pattern.

Code:

```

1 class Solution {
2 public:
3     uint32_t reverseBits(uint32_t n) {
4         uint32_t ans=0;
5         for(int i=0;i<32;i++){
6             if((n>>i)&1)
7                 ans+=(1<<(31-i));
8         }
9         return ans;
10    }
11 };

```

Testcase Result:

Accepted Runtime: 2 ms

Case 1

Input: n = 00000010100101000001111010011100

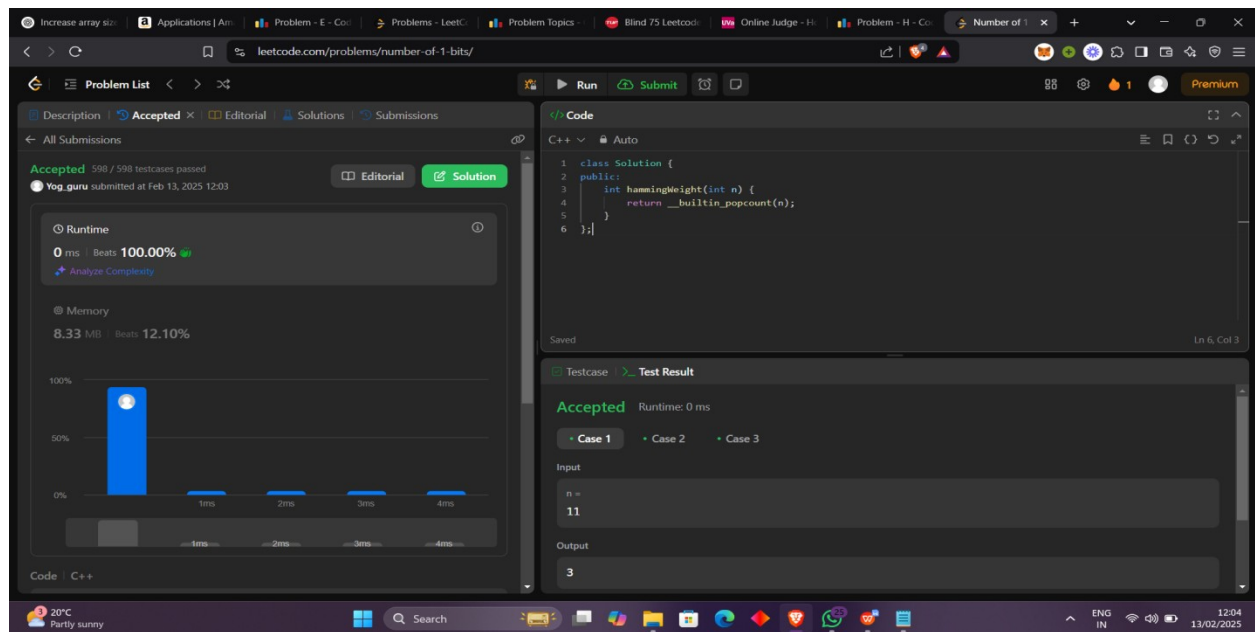
Output: 964176192 (00111001011110000010100101000000)

191. Number of 1 Bits

```

class Solution {
public:
    int hammingWeight(int n) {
        return __builtin_popcount(n);
    }
};

```



53. Maximum Subarray

```
class Solution {
```

```
public:
```

```
    int maxSubArray(vector<int>& nums) {
```

```
        int curr=0;
```

```
        int max=nums[0];
```

```
        for(int i=0;i<nums.size();i++){
```

```
            curr+=nums[i];
```

```
            if(curr>max)
```

```
                max=curr;
```

```
            if(curr<0)
```

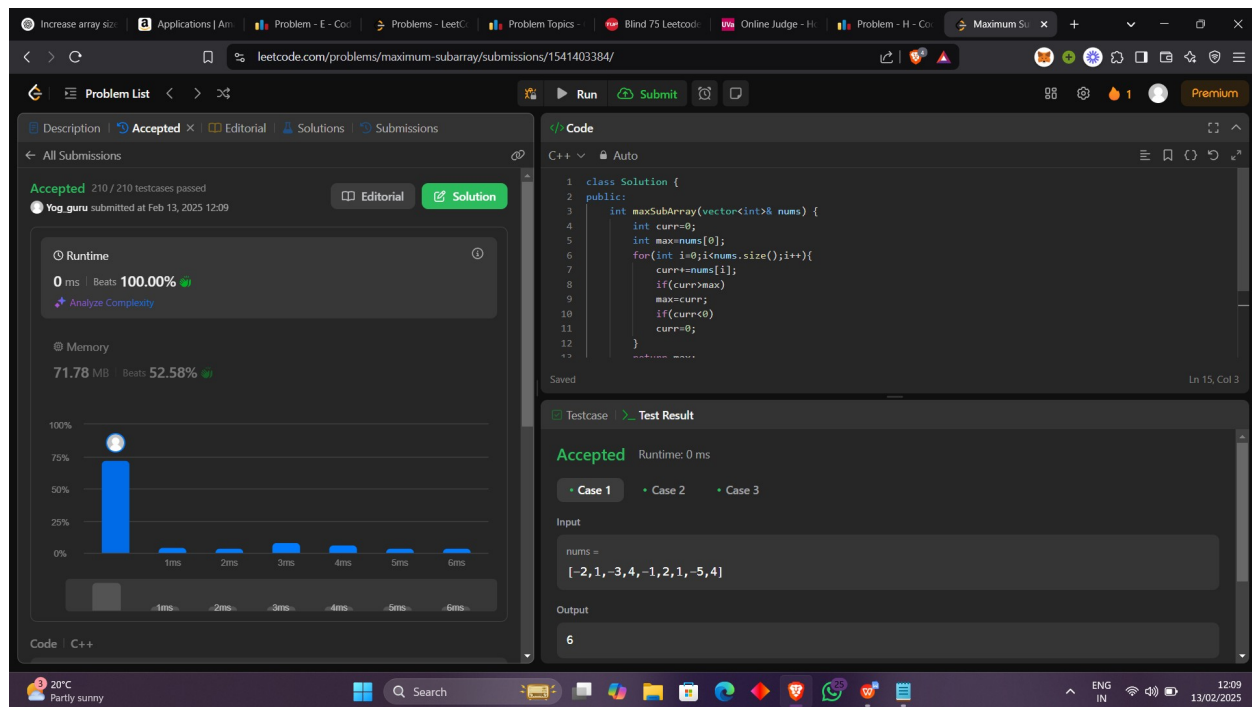
```
                curr=0;
```

```
        }
```

```

        return max;
    }
};

```



240. Search a 2D Matrix II

```

class Solution {
public:
    bool searchMatrix(vector<vector<int>>& matrix, int target) {
        for(int i=0;i<matrix.size();i++){
            if(matrix[i][0]<=target && matrix[i][matrix[i].size()-1]>=target){
                int start=0;
                int end=matrix[i].size()-1;
                while(start<=end){

```

```

        int mid=start+(end-start)/2;

        if(matrix[i][mid]==target)

            return true;

        if(matrix[i][mid]>target)

            end=mid-1;

        else

            start=mid+1;

    }

}

}

return false;

}

};

```

The screenshot displays a LeetCode submission for the problem "Search a 2D Matrix II". The submission is by user "Yog.guru" and was submitted on Feb 13, 2025, at 12:11. The solution is in C++ and is marked as "Accepted".

Runtime: 60 ms | Beat: 29.64%

Memory: 18.87 MB | Beat: 36.97%

The code implements a search function that iterates through each row of the matrix. For each row, it performs a binary search to find the target. If the target is found in any row, it returns true; otherwise, it returns false.

Test Results: The submission passed all test cases. The "Test Result" section shows two cases, both of which were "Accepted".

Case 1:

```

matrix = [[1,4,7,11,15],[2,5,8,12,19],[3,6,9,16,22],[10,13,14,17,24],[18,21,23,26,30]]
target = 5

```

Case 2:

```

matrix = [[1,4,7,11,15],[2,5,8,12,19],[3,6,9,16,22],[10,13,14,17,24],[18,21,23,26,30]]
target = 5

```

75. Sort Colours

```
class Solution {
```

```
public:
```

```
    void sortColors(vector<int>& nums) {
```

```
        int a=0,b=0,c=0;
```

```
        for(int i=0;i<nums.size();i++){
```

```
            if(nums[i]==0)
```

```
                a++;
```

```
            else if(nums[i]==1)
```

```
                b++;
```

```
            else
```

```
                c++;
```

```
        }
```

```
        for(int i=0;i<nums.size();i++){
```

```
            if(a>0){
```

```
                nums[i]=0;
```

```
                a--;
```

```
            }
```

```
            else if(b>0){
```

```
                nums[i]=1;
```

```
                b--;
```

```
            }
```

```
            else
```

```
                nums[i]=2;
```

```
        }
```

```
    }
```

```
};
```

The screenshot displays a LeetCode submission for the 'Sort Colors' problem. The code editor shows a C++ solution using a counting sort algorithm. The test result panel indicates that the solution is 'Accepted' with a runtime of 0 ms. The input array is [2, 0, 2, 1, 1, 0] and the output array is [0, 0, 1, 1, 2, 2].

```

1 class Solution {
2 public:
3     void sortColors(vector<int>& nums) {
4         int a=0,b=0,c=0;
5         for(int i=0;i<nums.size();i++){
6             if(nums[i]==0)
7                 a++;
8             else if(nums[i]==1)
9                 b++;
10            else
11                c++;
12        }
13    }
14 }

```

Testcase 1: Runtime: 0 ms
Input: nums = [2, 0, 2, 1, 1, 0]
Output: [0, 0, 1, 1, 2, 2]

162. Find Peak Element

```
class Solution {
```

```
public:
```

```
    int findPeakElement(vector<int>& nums) {
```

```
        if(nums.size()==1)
```

```
            return 0;
```

```
        if(nums[0]>nums[1])
```

```
            return 0;
```

```
        if(nums[nums.size()-1]>nums[nums.size()-2])
```

```
            return nums.size()-1;
```

```
        int start=0;
```

```
        int end=nums.size()-1;
```

```
        while(start<=end){
```

```
            int mid=start+(end-start)/2;
```

```
            if(nums[mid]>nums[mid-1] && nums[mid]>nums[mid+1])
```

```
                return mid;
```



```

        if(start==0)

        start++;

        if(nums[mid]<nums[mid+1])

        start=mid+1;

        else

        end=mid-1;

    }

    return 0;

}

};

```

The screenshot shows a LeetCode submission for the "Find Peak Element" problem. The submission is accepted, with a runtime of 0 ms and memory usage of 12.71 MB. The code is in C++ and implements a binary search algorithm. The test result shows the input [1, 2, 3, 1] and the output 2.

Code:

```

1 class Solution {
2 public:
3     int findPeakElement(vector<int>& nums) {
4         if(nums.size()==1)
5             return 0;
6         if(nums[0]>nums[1])
7             return 0;
8         if(nums[nums.size()-1]>nums[nums.size()-2])
9             return nums.size()-1;
10        int start=0;
11        int end=nums.size()-1;
12        while(start<end){
13            int mid=(start+end)/2;
14        }
15    }
16 };

```

Test Result:

Accepted Runtime: 0 ms

Case 1 Case 2

Input

nums =
[1, 2, 3, 1]

Output

2

33. Search in Rotated Sorted Array

```

class Solution {
public:
    int search(vector<int>& nums, int target) {

```

```
int start=0;
int end=nums.size()-1;
while(start<=end){
    int mid=start+(end-start)/2;
    if(nums[mid]==target)
        return mid;
    if(nums[start]<=nums[mid]){
        if(nums[mid]>target && nums[start]<=target)
            end=mid-1;
        else
            start=mid+1;
    }
    else{
        cout<<mid<<endl;
        if(nums[mid]<target && nums[end]>=target)
            start=mid+1;
        else
            end=mid-1;
    }
    cout<<start<<" "<<end<<endl;
}
return -1;
}
};
```

The screenshot shows a LeetCode submission for the problem "Search in Rotated Sorted Array". The submission is accepted with a runtime of 0 ms and memory of 15.22 MB. The code is in C++ and implements a binary search algorithm. The test result shows the input [4, 5, 6, 7, 0, 1, 2] and target 0, resulting in an accepted status.

4. Median of Two Sorted Arrays

```
class Solution {
```

```
public:
```

```
    double findMedianSortedArrays(vector<int>& nums1, vector<int>& nums2) {
```

```
        int midder=(nums1.size()+nums2.size())/2;
```

```
        int n=nums1.size();
```

```
        int m=nums2.size();
```

```
        nums1.push_back(INT_MAX);
```

```
        nums2.push_back(INT_MAX);
```

```
        if((n+m)%2==0){
```

```
            if(n==0){
```

```
                return (nums2[m/2]+nums2[m/2-1])/2.0;
```

```
            }
```

```

if(m==0){
    return (nums1[n/2]+nums1[n/2-1])/2.0;
}
int start=0;
int end=n;
int mid1,mid2;
while(start<=end){
    mid1=start+(end-start)/2;
    mid2=midder-mid1;
    if(mid2<0){
        end=mid1-1;
    }
    else if(m<mid2){
        start=mid1+1;
    }
    else{
        if(mid1==0){
            if(nums2[mid2-1]<=nums1[mid1]){
                return (nums2[mid2-1]+min(nums1[mid1],nums2[mid2]))/2.0;
            }
            else{
                start=mid1+1;
            }
        }
        else if(mid2==0){
            if(nums1[mid1-1]<=nums2[mid2]){
                return (nums1[mid1-1]+min(nums1[mid1],nums2[mid2]))/2.0;
            }
        }
    }
}

```

```

        else{
            end=mid1-1;
        }
    }
    else{
        if(nums1[mid1-1]>nums2[mid2]){
            end=mid1-1;
        }
        else if(nums2[mid2-1]>nums1[mid1]){
            start=mid1+1;
        }
        else{
            return (max(nums1[mid1-1],nums2[mid2-1])+min(nums1[mid1],nums2[mid2]))/2.0;
        }
    }
}

}

}
else{
    if(n==0){
        return nums2[m/2];
    }
    if(m==0){
        return nums1[n/2];
    }
    midder++;
    int start=0;
    int end=n;

```

```

int mid1,mid2;
while(start<=end){
    mid1=start+(end-start)/2;
    mid2=midder-mid1;
    if(mid2<0){
        end=mid1-1;
    }
    else if(m<mid2){
        start=mid1+1;
    }
    else{
        if(mid1==0){
            if(nums2[mid2-1]<=nums1[mid1]){
                return nums2[mid2-1];
            }
            else{
                start=mid1+1;
            }
        }
        else if(mid2==0){
            if(nums1[mid1-1]<=nums2[mid2]){
                return nums1[mid1-1];
            }
            else{
                end=mid1-1;
            }
        }
    }
}

```

```
else{
    if(nums1[mid1-1]>nums2[mid2]){
        end=mid1-1;
    }
    else if(nums2[mid2-1]>nums1[mid1]){
        start=mid1+1;
    }
    else{
        return max(nums2[mid2-1],nums1[mid1-1]);
    }
}
}
}
}
return 1;
}
};
```

The screenshot shows a LeetCode submission for the problem "Median of Two Sorted Arrays". The submission is accepted, with a runtime of 0ms and memory usage of 95.20 MB. The code is in C++ and implements a binary search solution. The test case shows two arrays: `nums1 = [1, 3]` and `nums2 = [2]`.

278. First Bad Version

```
class Solution {
```

```
public:
```

```
    int firstBadVersion(int n) {
```

```
        int first = 1;
```

```
        int last = n;
```

```
        while (first < last) {
```

```
            int mid = first + (last - first) / 2;
```

```
            if (isBadVersion(mid)) {
```

```
                last = mid;
```

```
            } else {
```

```
                first = mid + 1;
```

```
            }
```

```
        }
```

```
        return first;
```


Accepted

24 / 24 testcases passed

Vog guru submitted at Feb 13, 2025 12:39

Runtime

2 msBeats 54.63%

Analyze Complexity

Memory

8.00 MBBeats 38.88%

Time Interval	Percentage
1ms	~45%
2ms	~35%
3ms	~15%
4ms	~5%

Code

```
1 class Solution {  
2 public:  
3     int firstBadVersion(int n) {  
4         int first = 1;  
5         int last = n;  
6         while (first < last) {  
7             int mid = first + (last - first) / 2;  
8             if (isBadVersion(mid)) {  
9                 last = mid;  
10            } else {  
11                first = mid + 1;  
12            }  
13        }  
14    }  
15 }
```

Testcase

Test Result

Accepted Runtime: 2 ms

Case 1 Case 2

Input

n = 5

bad = 4