

191. [Number of 1 Bits](#)

```
class Solution {  
  
public:  
  
    int hammingWeight(int n) {  
  
        int count = 0;  
  
        for(int i = 31; i >= 0; i--){  
  
            if(((n >> i) & 1) == 1)  
  
                count++;  
  
        }  
  
        return count;  
  
    }  
  
};
```

The screenshot displays a C++ IDE interface. The top toolbar includes icons for 'Problem List', 'Run', 'Submit', and other standard IDE functions. The main editor area shows the C++ code for the 'Number of 1 Bits' problem, which is a class 'Solution' with a public method 'hammingWeight' that iterates from bit 31 down to 0, counting the number of set bits. The code is syntactically correct and matches the provided text. Below the code editor, the 'Test Result' section shows that the solution was 'Accepted' with a runtime of 0 ms. It lists three test cases, with 'Case 1' selected. The input for Case 1 is 'n = 11'.

Status	Language	Runtime	Memory	Notes
1 Accepted Mar 04, 2025	C++	0 ms	8.3 MB	

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

Saved

Testcase Test Result

Accepted Runtime: 0 ms

• Case 1 • Case 2 • Case 3

Input

n =
11

108. [Convert Sorted Array to Binary Search Tree](#)

```
#include <vector>
```

```
using namespace std;
```

```
class Solution {
```

```
public:
```

```
    TreeNode* sortedArrayToBST(vector<int>& nums) {
```

```
        return helper(nums, 0, nums.size() - 1);
```

```
    }
```

```
private:
```

```
    TreeNode* helper(vector<int>& nums, int left, int right) {
```

```
        if (left > right) return nullptr;
```

```
        int mid = left + (right - left) / 2;
```

```
        TreeNode* root = new TreeNode(nums[mid]);
```

```
        root->left = helper(nums, left, mid - 1);
```

```
        root->right = helper(nums, mid + 1, right);
```

```
        return root;
```

```
    }
```

```
};
```

The screenshot displays the LeetCode submission interface. On the left, the 'Submissions' tab is active, showing a submission with status 'Accepted' at 'a few seconds ago' using 'C++' with a runtime of '0 ms' and memory usage of '22.9 MB'. On the right, the 'Code' editor shows the C++ implementation of the solution. The code defines a class 'Solution' with a public method 'sortedArrayToBST' and a private recursive helper method 'helper'.

```
1 class Solution {
2 public:
3     TreeNode* sortedArrayToBST(vector<int>& nums) {
4         return helper(nums, 0, nums.size() - 1);
5     }
6 private:
7     TreeNode* helper(vector<int>& nums, int left, int right) {
8         if (left > right) return nullptr;
9         int mid = left + (right - left) / 2;
10        TreeNode* root = new TreeNode(nums[mid]);
11        root->left = helper(nums, left, mid - 1);
12        root->right = helper(nums, mid + 1, right);
13        return root;
14    }
15 };
16
```

912. [Sort an Array](#)

```
#include <vector>

#include <functional>

class Solution {
public:
    vector<int> sortArray(vector<int>& nums) {
        std::function<void(int, int)> quickSort = [&](int left, int right) {
            if (left >= right) {
                return;
            }
            int pivotIndex = left + (right - left) / 2;
            int pivotValue = nums[pivotIndex];
            int i = left - 1;
            int j = right + 1;
            while (i < j) {
                do {
                    i++;
                } while (nums[i] < pivotValue);
                do {
                    j--;
                } while (nums[j] > pivotValue);
                if (i < j) {
                    std::swap(nums[i], nums[j]);
                }
            }
        };
    }
};
```

```

        quickSort(left, j);

        quickSort(j + 1, right);

    };

    quickSort(0, nums.size() - 1);

    return nums;

}

};

```

The screenshot shows a C++ IDE with two main panels. The left panel displays the submission status: 'Accepted' a few seconds ago, C++ language, 558 ms runtime, and 78.3 MB memory. The right panel shows the source code for a quicksort function. The code defines a class 'Solution' with a public method 'sortArray' that takes a vector of integers and returns it sorted. The sorting is done using a recursive quicksort function. The quicksort function has a base case where if 'left' is greater than or equal to 'right', it returns. It then selects a pivot at the middle index, partitions the array by moving elements less than the pivot to the left and greater to the right, and recursively sorts the left and right sub-arrays. Finally, it returns the sorted array.

```

1  class Solution {
2  public:
3      vector<int> sortArray(vector<int>& nums) {
4          std::function<void(int, int)> quickSort = [&](int left, int right) {
5              if (left >= right) {
6                  return;
7              }
8              int pivotIndex = left + (right - left) / 2;
9              int pivotValue = nums[pivotIndex];
10             int i = left - 1;
11             int j = right + 1;
12             while (i < j) {
13                 do {
14                     i++;
15                 } while (nums[i] < pivotValue);
16                 do {
17                     j--;
18                 } while (nums[j] > pivotValue);
19                 if (i < j) {
20                     std::swap(nums[i], nums[j]);
21                 }
22             }
23             quickSort(left, j);
24             quickSort(j + 1, right);
25         };
26         quickSort(0, nums.size() - 1);
27         return nums;
28     }
29 };

```

53. [Maximum Subarray](#)

```
class Solution {
```

```
public:
```

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

```
        int maxSum = INT_MIN;
```

```
        int currentSum = 0;
```

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

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

```
            if (currentSum > maxSum) {
```

```
                maxSum = currentSum;
```

```

    }

    if (currentSum < 0) {

        currentSum = 0;

    }

}

return maxSum;

}

};

```

The screenshot displays a LeetCode submission for the 'Maximum Subarray' problem. The submission is successful, marked as 'Accepted' on February 13, 2025, with a runtime of 0 ms and memory usage of 71.8 MB. The code is written in C++ and implements Kadane's algorithm to find the maximum subarray sum. The 'Test Result' section shows that the solution passed 'Case 1' with the input array [-2, 1, -3, 4, -1, 2, 1, -5, 4].

```

1 class Solution {
2 public:
3     int maxSubArray(vector<int>& nums) {
4         int maxSum = INT_MIN;
5         int currentSum = 0;
6         for (int i = 0; i < nums.size(); i++) {
7             currentSum += nums[i];
8             if (currentSum > maxSum) {
9                 maxSum = currentSum;
10            }
11            if (currentSum < 0) {
12                currentSum = 0;
13            }
14        }
15        return maxSum;
16    }
17 }

```

Testcase **Accepted** Runtime: 0 ms

Case 1 Case 2 Case 3

Input

nums =

[-2, 1, -3, 4, -1, 2, 1, -5, 4]

932. Beautiful Array

```

class Solution {
public:
    int partition(vector<int> &v, int start, int end, int mask)
    {
        int j = start;
        for(int i = start; i <= end; i++)
        {
            if((v[i] & mask) != 0)
            {
                swap(v[i], v[j]);
            }
        }
    }
};

```

```
        j++;  
    }  
}  
return j;  
}
```

```
void sort(vector<int> &v, int start, int end, int mask)  
{  
    if(start >= end) return;  
    int mid = partition(v, start, end, mask);  
    sort(v, start, mid - 1, mask << 1);  
    sort(v, mid, end, mask << 1);  
}
```

```
vector<int> beautifulArray(int n) {  
    vector<int> ans;  
    for(int i = 0; i < n; i++) ans.push_back(i + 1);  
    sort(ans, 0, n - 1, 1);  
    return ans;  
}  
};
```

Description Editorial Solutions **Submissions**

Status Language Runtime Memory Notes

1 **Accepted** Feb 13, 2025 C++ 0 ms 9.4 MB

```

1 class Solution {
2 public:
3   int partition(vector<int> &v, int start, int end, int mask)
4   {
5       int j = start;
6       for(int i = start; i <= end; i++)
7       {
8           if((v[i] & mask) != 0)
9           {
10              swap(v[i], v[j]);
11              j++;
12          }
13      }
14      return j;
15  }

```

Saved Ln 1, Col

☒ Testcase **Test Result**

Accepted Runtime: 0 ms

• Case 1 • Case 2

Input

n = 4

372. [Super Pow](#)

```

class Solution {
    const int base = 1337;

    int powmod(int a, int k)
    {
        a %= base;

        int result = 1;

        for (int i = 0; i < k; ++i)
            result = (result * a) % base;

        return result;
    }

public:
    int superPow(int a, vector<int>& b) {

        if (b.empty()) return 1;

        int last_digit = b.back();

        b.pop_back();

        return powmod(superPow(a, b), 10) * powmod(a, last_digit) % base;

    }

};

```

The screenshot shows a LeetCode submission for a problem. The submission is 'Accepted' with a runtime of 2 ms and memory of 15.4 MB. The code is in C++ and defines a 'powmod' function and a 'superPow' method. The test result shows 'Accepted' for Case 1 with input 'a = 2'.

```

1 class Solution {
2     const int base = 1337;
3     int powmod(int a, int k)
4     {
5         a %= base;
6         int result = 1;
7         for (int i = 0; i < k; ++i)
8             result = (result * a) % base;
9         return result;
10    }
11 public:
12    int superPow(int a, vector<int>& b) {
13        if (b.empty()) return 1;
14        int last_digit = b.back();
15        b.pop_back();
16    }
17 };

```

Testcase: **Accepted** Runtime: 0 ms

Case 1 Case 2 Case 3

Input

a =
2

218. The Skyline Problem

```
class Solution {
```

```
public:
```

```
vector<vector<int>> getSkyline(vector<vector<int>>& buildings) {
```

```
    vector<vector<int>> ans;
```

```
    multiset<int> pq{0};
```

```
    vector<pair<int, int>> points;
```

```
    for(auto b: buildings){
```

```
        points.push_back({b[0], -b[2]});
```

```
        points.push_back({b[1], b[2]});
```

```
    }
```

```
    sort(points.begin(), points.end());
```

```
    int ongoingHeight = 0;
```

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

```
        int currentPoint = points[i].first;
```

```
        int heightAtCurrentPoint = points[i].second;
```



```

        if(heightAtCurrentPoint < 0){

            pq.insert(-heightAtCurrentPoint);

        } else {

            pq.erase(pq.find(heightAtCurrentPoint));

        }

        auto pqTop = *pq.rbegin();

        if(ongoingHeight != pqTop){

            ongoingHeight = pqTop;

            ans.push_back({currentPoint, ongoingHeight});

        }

    }

    return ans;

}

};

```

Status	Language	Runtime	Memory	Notes
Accepted a few seconds ago	C++	12 ms	28.8 MB	+ Notes
Compile Error a minute ago	Java	N/A	N/A	


```

1 class Solution {
2 public:
3     vector<vector<int>> getSkyline(vector<vector<int>>& buildings) {
4         vector<vector<int>> ans;
5         multiset<int> pq{0};
6
7         vector<pair<int, int>> points;
8
9         for(auto b: buildings){
10             points.push_back({b[0], -b[2]});
11             points.push_back({b[1], b[2]});
12         }
13         sort(points.begin(), points.end());
14
15         int ongoingHeight = 0;
16         for(int i = 0; i < points.size(); i++){
17             int currentPoint = points[i].first;
18             int heightAtCurrentPoint = points[i].second;
19
20             if(heightAtCurrentPoint < 0){
21                 pq.insert(-heightAtCurrentPoint);
22             } else {
23                 pq.erase(pq.find(heightAtCurrentPoint));
24             }
25             auto pqTop = *pq.rbegin();
26             if(ongoingHeight != pqTop){
27                 ongoingHeight = pqTop;

```