Worksheet 6

Student Name: Sagar Sharma   **UID:** 22CS11764

Branch:CSE   **Section/Group:** 605-B

Semester: 6   **Date of Performance:** 19/03/25

Subject Name: AP   **Subject Code:** 22CSP-351

1. Given an integer array nums where the elements are sorted in ascending order, convert it to a height-balanced binary search tree.

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

    if (left > right) return nullptr;  // Base case


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

    TreeNode* root = new TreeNode(nums[mid]);  // Create root node


    // Recursively build left and right subtrees

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

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


    return root;

}


// Helper function to start recursion
```
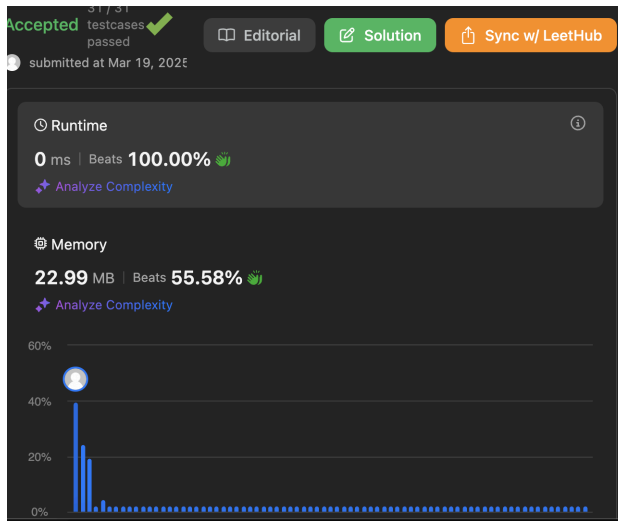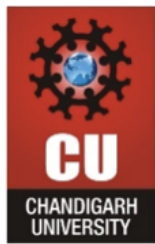
```
TreeNode* sortedArrayToBST(vector<int>& nums) {

    return sortedArrayToBST(nums, 0, nums.size() - 1);

}
```


2. Given a positive integer n, write a function that returns the number of set bits in its binary representation (also known as the Hamming weight).
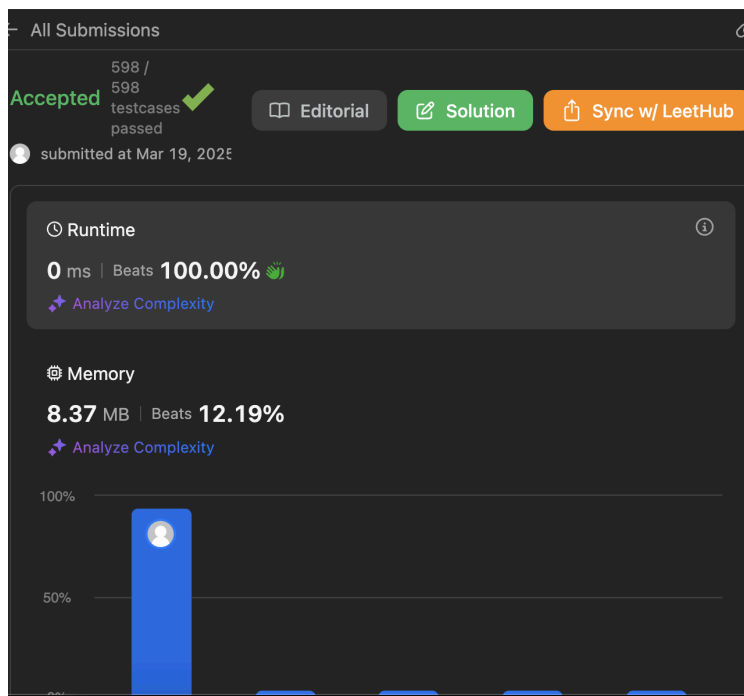
```
class Solution {

public:

    int hammingWeight(uint32_t n) {

    int count = 0;

    while (n != 0) {

        count += (n & 1);

        n >>= 1;

    }
```

```
        return count;

    }

};
```

3. Given an array of integers nums, sort the array in ascending order and return it.

   You must solve the problem without using any built-in functions in O(nlog(n)) time complexity and with the smallest space complexity possible.

```cpp
void merge(vector<int>& nums, int left, int mid, int right) {

    int n1 = mid - left + 1;  // Size of left subarray

    int n2 = right - mid;  // Size of right subarray


    // Create temporary arrays
```

```cpp
vector<int> leftArr(n1);

vector<int> rightArr(n2);


// Copy data to temporary arrays

for (int i = 0; i < n1; i++)

    leftArr[i] = nums[left + i];

for (int i = 0; i < n2; i++)

    rightArr[i] = nums[mid + 1 + i];


// Merge two sorted subarrays

int i = 0, j = 0, k = left;

while (i < n1 && j < n2) {

    if (leftArr[i] <= rightArr[j])

        nums[k++] = leftArr[i++];

    else

        nums[k++] = rightArr[j++];

}


// Copy remaining elements

while (i < n1) nums[k++] = leftArr[i++];

while (j < n2) nums[k++] = rightArr[j++];

}
```
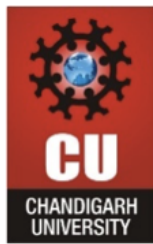
```
// Merge Sort function

void mergeSort(vector<int>& nums, int left, int right) {

    if (left < right) {

        int mid = left + (right - left) / 2;  // Find the middle point

        mergeSort(nums, left, mid);  // Sort left half

        mergeSort(nums, mid + 1, right);  // Sort right half

        merge(nums, left, mid, right);  // Merge sorted halves

    }

}


// Sorting function

vector<int> sortArray(vector<int>& nums) {

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

    return nums;}
```
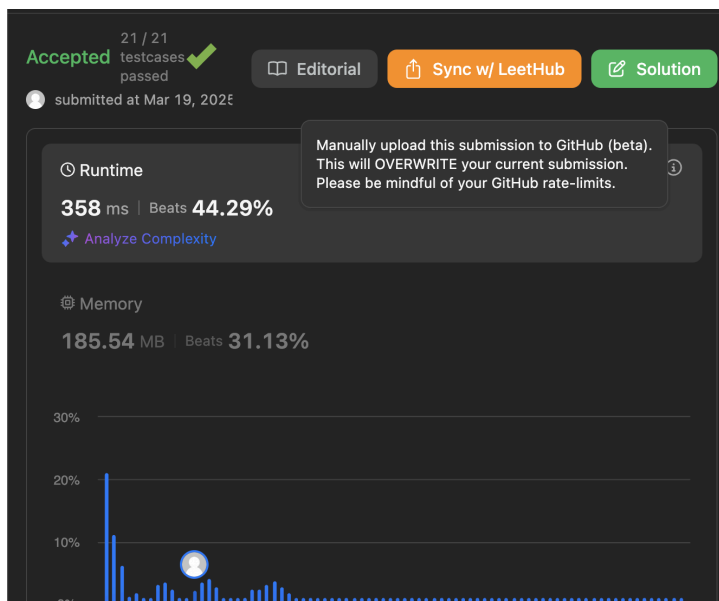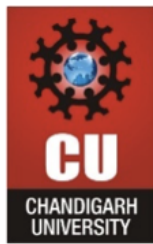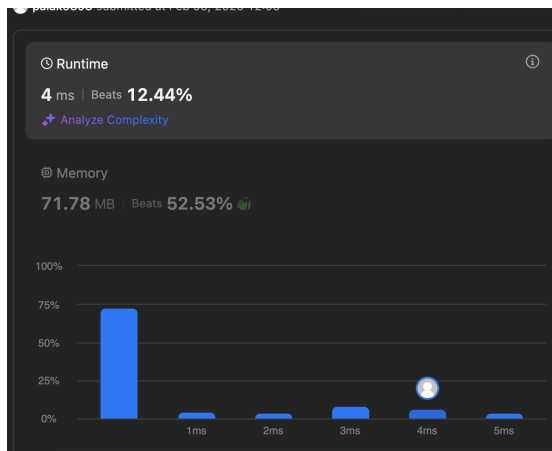
4. Given an integer array nums, find the subarray with the largest sum, and return its sum.

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

    int maxSum = nums[0];

    int currentSum = nums[0];


    for (int i = 1; i < nums.size(); ++i) {

        currentSum = max(nums[i], currentSum + nums[i]);  // Extend the subarray or start a new one

        maxSum = max(maxSum, currentSum);             // Update the max sum

    }


    return maxSum;

}
```


5. An array nums of length n is beautiful if: nums is a permutation of the integers in the range [1, n].

For every $0 <= i < j < n$, there is no index k with $i < k < j$ where $2 * nums[k] == nums[i] + nums[j]$. Given the integer n, return any beautiful array nums of length n. There will be at least one valid answer for the given n.
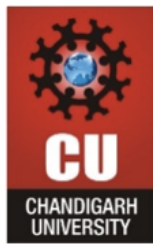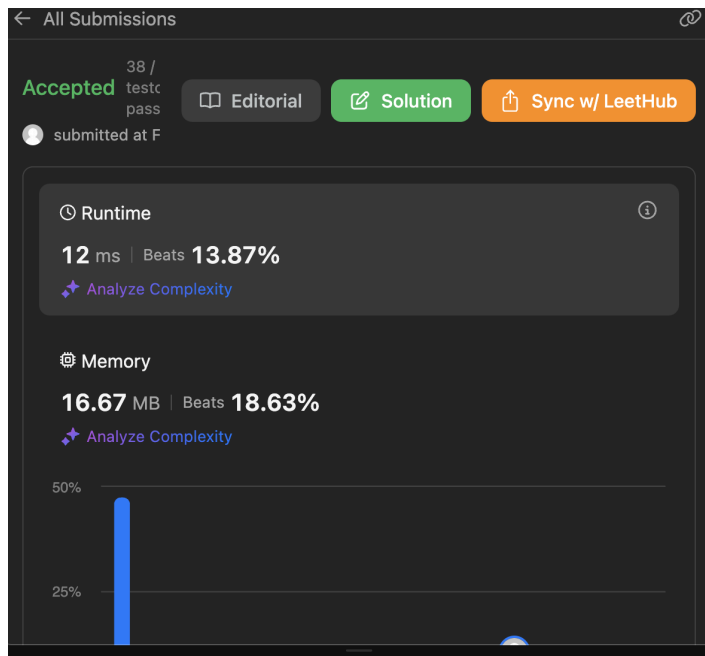
```cpp
class Solution {

public:

    vector<int> beautifulArray(int n) {

    if (n == 1) return {1};


        vector<int> odd = beautifulArray((n + 1) / 2);  // Construct for odd indices

        vector<int> even = beautifulArray(n / 2);        // Construct for even indices


        vector<int> result;

        for (int x : odd) result.push_back(2 * x - 1);  // Map odd part: 2*x - 1

        for (int x : even) result.push_back(2 * x);    // Map even part: 2*x


        return result;
    }

};
```
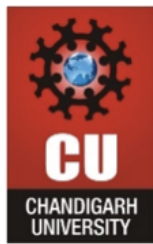
6. Your task is to calculate ab mod 1337 where a is a positive integer and b is an extremely large positive integer given in the form of an array.

```
int modPow(int a, int b, int mod) {

    int result = 1;

    a %= mod;

    while (b > 0) {

        if (b % 2 == 1) {

            result = (result * a) % mod;

        }

        a = (a * a) % mod;

        b /= 2;

    }

    return result;

}
```
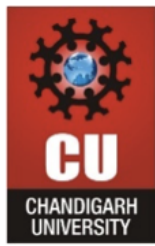
```
// Function to calculate a^b % 1337 where b is given as a vector of digits

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

    if (b.empty()) return 1;



    int lastDigit = b.back();

    b.pop_back();



    int part1 = modPow(superPow(a, b), 10, MOD);  // a^(remaining digits * 10) % MOD

    int part2 = modPow(a, lastDigit, MOD);        // a^lastDigit % MOD



    return (part1 * part2) % MOD;

}
```
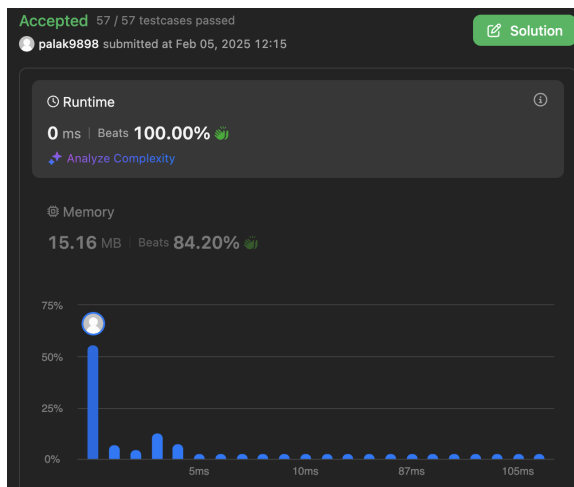
Accepted 57 / 57 testcases passed

☑ Solution

palak9898 submitted at Feb 05, 2025 12:15

⏱ Runtime

0 ms │ Beats **100.00%** 🙌

✦ Analyze Complexity

⚙ Memory

15.16 MB │ Beats **84.20%** 🙌

```
75%

50%

25%

0%
        5ms        10ms       87ms       105ms
```

7. An array nums of length n is beautiful if: nums is a permutation of the integers in the range [1, n].
   For every $0 <= i < j < n$, there is no index k with $i < k < j$ where $2 * nums[k] == nums[i] + nums[j]$.
   Given the integer n, return any beautiful array nums of length n. There will be at least one valid
   answer for the given n.

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

    vector<pair<int, int>> events;  // Stores {x, height} events

    multiset<int> heights = {0};    // Max heap using multiset

    vector<vector<int>> result;


    // Convert buildings into events

    for (auto& b : buildings) {

        events.push_back({b[0], -b[2]}); // Start event (negative height for max heap)

        events.push_back({b[1], b[2]});  // End event (positive height)

    }


    // Sort events

    sort(events.begin(), events.end());


    int prevMaxHeight = 0;  // Previous max height


    // Process events

    for (auto& [x, h] : events) {

        if (h < 0) {

            heights.insert(-h); // Add building height

        } else {

            heights.erase(heights.find(h)); // Remove building height

        }
```
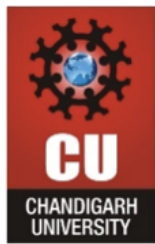
```
int currMaxHeight = *heights.rbegin(); // Get max height from set


if (currMaxHeight != prevMaxHeight) {

    result.push_back({x, currMaxHeight});

    prevMaxHeight = currMaxHeight;

  }

}


return result;

}
```