main.cpp

Share Run

```cpp
#include <vector>
#include <iostream>
using namespace std;

int majorityElement(vector<int>& nums) {
    int candidate = 0, count = 0;
    for (int num : nums) {
        if (count == 0) {
            candidate = num;
        }
        count += (num == candidate) ? 1 : -1;
    }
    return candidate;
}

int main() {
    vector<int> nums1 = {3, 2, 3};
    vector<int> nums2 = {2, 2, 1, 1, 1, 2, 2};
    cout << majorityElement(nums1) << endl; // Output: 3
    cout << majorityElement(nums2) << endl; // Output: 2
    return 0;
}
```

Output

```
3
2


=== Code Execution Successful ===
```

main.cpp

Share Run

```cpp
#include <vector>
#include <iostream>
using namespace std;

vector<vector<int>> generate(int numRows) {
    vector<vector<int>> triangle;
    for (int i = 0; i < numRows; ++i) {
        vector<int> row(i + 1, 1);
        for (int j = 1; j < i; ++j) {
            row[j] = triangle[i - 1][j - 1] + triangle[i - 1][j];
        }
        triangle.push_back(row);
    }
    return triangle;
}

void printTriangle(const vector<vector<int>>& triangle) {
    for (const auto& row : triangle) {
        for (int num : row) {
            cout << num << " ";
        }
        cout << endl;
    }
}

int main() {
    int numRows = 5;
    vector<vector<int>> triangle = generate(numRows);
    printTriangle(triangle);
```

Output

```
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1


=== Code Execution Successful ===
```

main.cpp

Share Run

Output

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

int maxArea(vector<int>& height) {
    int left = 0, right = height.size() - 1;
    int max_area = 0;
    while (left < right) {
        int width = right - left;
        max_area = max(max_area, min(height[left], height[right]) * width);
        if (height[left] < height[right]) {
            ++left;
        } else {
            --right;
        }
    }
    return max_area;
}

int main() {
    vector<int> height1 = {1, 8, 6, 2, 5, 4, 8, 3, 7};
    vector<int> height2 = {1, 1};
    cout << maxArea(height1) << endl; // Output: 49
    cout << maxArea(height2) << endl; // Output: 1
    return 0;
}

```

```
49
1


=== Code Execution Successful ===
```

main.cpp

Share Run

```
        }
        int hash = carry;
        for (int r : state) {
            hash = hash * 31 + r;
        }
        static unordered_map<int, int> memo;
        if (memo.count(hash)) return memo[hash];

        int max_happy = 0;
        for (int i = 0; i < batchSize; ++i) {
            if (state[i] > 0) {
                --state[i];
                max_happy = max(max_happy, dfs(state, (carry + i) % batchSize)
                    + (carry == 0 ? 1 : 0));
                ++state[i];
            }
        }
        return memo[hash] = max_happy;
    };

    return dfs(remainders, 0) + remainders[0];
}

int main() {
    vector<int> groups1 = {1, 2, 3, 4, 5, 6};
    vector<int> groups2 = {1, 3, 2, 5, 2, 2, 1, 6};
    cout << maxHappyGroups(3, groups1) << endl; // Output: 4
    cout << maxHappyGroups(4, groups2) << endl; // Output: 4
    return 0;
}
```

Output

```
6
5


=== Code Execution Successful ===
```

main.cpp

Share Run

```cpp
#include <vector>
#include <iostream>
#include <algorithm>
#include <numeric> // For accumulate
#include <functional> // For function

using namespace std;

int minimumTimeRequired(vector<int>& jobs, int k) {
    vector<int> workers(k, 0);
    int n = jobs.size();
    sort(jobs.rbegin(), jobs.rend()); // Sort jobs in descending order

    // Helper function using lambda
    function<bool(int, int)> canDistribute = [&](int idx, int limit) {
        if (idx == n) return true;
        for (int i = 0; i < k; ++i) {
            if (workers[i] + jobs[idx] <= limit) {
                workers[i] += jobs[idx];
                if (canDistribute(idx + 1, limit)) return true;
                workers[i] -= jobs[idx];
            }
            if (workers[i] == 0) break; // Prune redundant cases
        }
        return false;
    };

    int left = jobs[0], right = accumulate(jobs.begin(), jobs.end(), 0);
    while (left < right) {
        int mid = left + (right - left) / 2;
```

Output

```
5
15


=== Code Execution Successful ===
```