

Kaushik Dhruv Alok
22BCS10210
KPIT-901/A
Day 2 Questions

Q1. Majority Elements:

Given an array nums of size n, return the majority element.

The majority element is the element that appears more than $\lfloor n / 2 \rfloor$ times. You may assume that the majority element always exists in the array.

Code:

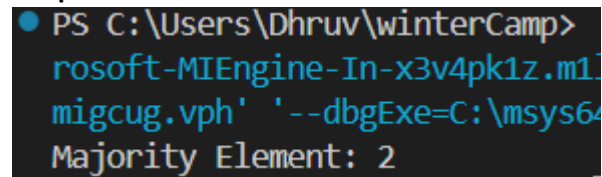
```
#include <iostream>
#include <vector>
using namespace std;

int majorityNum(const vector<int>& nums) {
    int max = 0;
    int count = 0;

    for (int num : nums) {
        if (count == 0) {
            max = num;
            count = 1;
        } else if (num == max) {
            count++;
        } else {
            count--;
        }
    }
    return max;
}

int main() {
    vector<int> nums = {2,2,1,1,1,2,2};
    cout << "Majority Element: " << majorityNum(nums) << endl;
    return 0;
}
```

Output:



```
PS C:\Users\Dhruv\winterCamp>
g++ -std=c++11 -g -o majority majority.cpp
Majority Element: 2
```

Q2. Pascal's Triangle:

Given an integer numRows, return the first numRows of Pascal's triangle.

Code:

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

```

vector<vector<int>> generate(int numRows) {
    vector<vector<int>> result;
    result.reserve(numRows);

    for (int i = 0; i < numRows; ++i) {
        vector<int> row(i + 1, 1); // Initialize all to 1
        for (int j = 1; j < i; ++j) {
            row[j] = result[i-1][j-1] + result[i-1][j];
        }
        result.push_back(row);
    }

    return result;
}

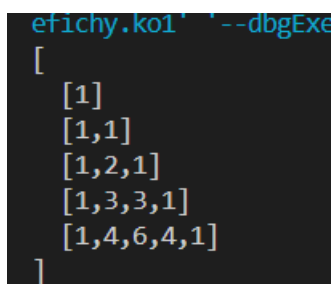
int main() {
    int numRows = 5;
    vector<vector<int>> pascal = generate(numRows);

    cout << "[\n";
    for (const auto& row : pascal) {
        cout << " [";
        for (int i = 0; i < (int)row.size(); ++i) {
            cout << row[i];
            if (i < (int)row.size() - 1) cout << ",";
        }
        cout << "]\n";
    }
    cout << "]" << endl;

    return 0;
}

```

Output:



```

[
  [1]
  [1,1]
  [1,2,1]
  [1,3,3,1]
  [1,4,6,4,1]
]

```

Q3. Container With Most Water

You are given an integer array `height` of length `n`. There are `n` vertical lines drawn such that the two endpoints of the `i`th line are $(i, 0)$ and $(i, \text{height}[i])$.

Find two lines that together with the x-axis form a container, such that the container contains the most water.

Return the maximum amount of water a container can store.

Code:

```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

class Solution {
public:
    int maxArea(vector<int>& height) {
        int left = 0;
        int right = (int)height.size() - 1;
        int max_water = 0;

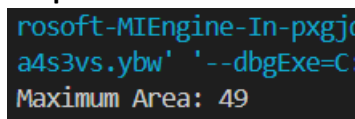
        while (left < right) {
            int current_height = min(height[left], height[right]);
            int current_width = right - left;
            int current_area = current_height * current_width;

            max_water = max(max_water, current_area);

            if (height[left] < height[right]) {
                left++;
            } else {
                right--;
            }
        }

        return max_water;
    }
};

int main() {
    vector<int> height = {1,8,6,2,5,4,8,3,7};
    Solution sol;
    cout << "Maximum Area: " << sol.maxArea(height) << endl;
    return 0;
}
```

Output:

```
rossoft-MIEngine-In-pxgjc
a4s3vs.ybw' '--dbgExe=C
Maximum Area: 49
```

Q4. Maximum Number of Groups Getting Fresh Donuts.

Q5. Find Minimum Time to Finish All Jobs.

Code:

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <unordered_set>
#include <functional>
```

```

using namespace std;

class Solution {
public:
    int maxHappyGroups(int batchSize, vector<int>& groups) {
        int ans = 0;
        vector<int> cnt(batchSize, 0);
        for (int g : groups) {
            int r = g % batchSize;
            if (r == 0) ans++;
            else cnt[r]++;
        }
        vector<int> freq;
        for (int i = 1; i < batchSize; i++) {
            for (int j = 0; j < cnt[i]; j++)
                freq.push_back(i);
        }
        if (freq.empty()) return ans;
        int n = (int)freq.size();
        vector<vector<int>> memo(1 << n, vector<int>(batchSize, -1));
        function<int(int,int)> dfs = [&](int mask, int r) -> int {
            if (mask == (1 << n) - 1) return 0;
            if (memo[mask][r] != -1) return memo[mask][r];
            int res = 0;
            for (int i = 0; i < n; i++) {
                if ((mask & (1 << i)) == 0) {
                    int nr = (r + freq[i]) % batchSize;
                    res = max(res, (nr == 0 ? 1 : 0) + dfs(mask | (1 << i), nr));
                }
            }
            memo[mask][r] = res;
            return res;
        };
        ans += dfs(0, 0);
        return ans;
    }
};

```

```

class Solution2 {
public:
    int res;
    void backtrack(vector<int>& jobs, vector<int>& workerTime, int idx, int k, int curMax) {
        if (idx == (int)jobs.size()) {
            res = min(res, curMax);
            return;
        }
        if (curMax >= res) return;
        unordered_set<int> seen;
        for (int i = 0; i < k; i++) {
            if (seen.count(workerTime[i])) continue;
            seen.insert(workerTime[i]);
            workerTime[i] += jobs[idx];
            backtrack(jobs, workerTime, idx + 1, k, max(curMax, workerTime[i]));
            workerTime[i] -= jobs[idx];
            if (workerTime[i] == 0) break;
        }
    }
};

```

```

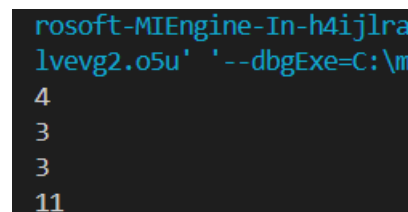
int minimumTimeRequired(vector<int>& jobs, int k) {
    sort(jobs.begin(), jobs.end(), greater<int>());
    int sum = 0;
    for (auto j : jobs) sum += j;
    res = sum;
    vector<int> workerTime(k, 0);
    backtrack(jobs, workerTime, 0, k, 0);
    return res;
}

};

int main() {
    {
        Solution sol;
        int batchSize = 3;
        vector<int> groups = {1,2,3,4,5,6};
        cout << sol.maxHappyGroups(batchSize, groups) << endl;
    }
    {
        Solution sol;
        int batchSize = 4;
        vector<int> groups = {1,3,2,5,2,2,1,6};
        cout << sol.maxHappyGroups(batchSize, groups) << endl;
    }
    {
        Solution2 sol2;
        vector<int> jobs = {3,2,3};
        int k = 3;
        cout << sol2.minimumTimeRequired(jobs, k) << endl;
    }
    {
        Solution2 sol2;
        vector<int> jobs = {1,2,4,7,8};
        int k = 2;
        cout << sol2.minimumTimeRequired(jobs, k) << endl;
    }
    return 0;
}

```

Output:



```

rossoft-MIEngine-In-h4ijlra
lvevg2.o5u' '--dbgExe=C:\m
4
3
3
11

```