Name : Nipun Chugh
UID   : 22BCS10636
Class : KPIT - 901 / A

Q1. Given an array of size n, return the majority element.
Q2. Given an integer n, return the first n rows of Pascal's triangle.
Q3. 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, 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.
Q4. There is a donut shop that bakes donuts in batches of batch_size. They have a rule where they must serve all of the donuts of a batch before serving any donuts of the next batch. You are given an integer batch_size and an integer array groups, where groups[i] denotes that there is a group of groups[i] customers that will visit the shop. Each customer will get exactly one donut. When a group visits the shop, all customers of the group must be served before serving any of the following groups. A group will be happy if they all get fresh donuts. That is, the first customer of the group does not receive a donut that was left over from the previous group.
Q5. You are given an integer array jobs, where jobs[i] is the amount of time it takes to complete the i^{th} job. There are k workers that you can assign jobs to. Each job should be assigned to exactly one worker. The working time of a worker is the sum of the time it takes to complete all jobs assigned to them. Your goal is to devise an optimal assignment such that the maximum working time of any worker is minimized. Return the minimum possible maximum working time of any assignment.

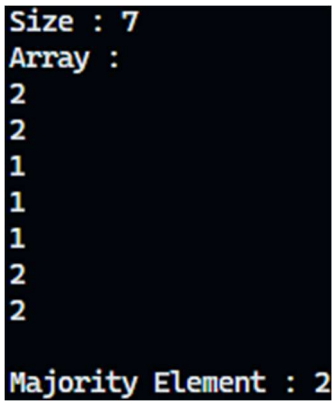
Solutions :

A1. Majority Elements.

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

using namespace std;

int majority_element(vector<int>& arr) {
    int n      = arr.size();
    int count  = 0;
    int retval = 0;

    for (int i = 0; i < n; i++) {
        if (count == 0) {
            retval = arr[i];
            count  = 1;
        } else if (arr[i] == retval) {
            count++;
        } else {
            count--;
        }
    }

    return retval;
}
```

```
int main() {
    vector<int> t;
    int n = 0;
    cout << "Size : ";
    cin >> n;

    cout << "Array : \n";
    for (int i = 0; i < n; i++) {
        int x = 0;
        cin >> x;
        t.emplace_back(x);
    }

    cout << "\nMajority Element : " << majority_element(t) << endl;
    return 0;
}
```

Output :

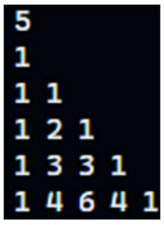


A2. Pascal’s Triangle.

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

using namespace std;

vector<vector<int>> pascal_triangle(int rows) {
    vector<vector<int>> retval;
    for (int i = 0; i < rows; i++) {
        if (i == 0) {
            retval.push_back({1});
        } else if (i == 1) {
            retval.push_back({1, 1});
        } else {
            vector<int> row(i + 1);
            row[0] = row[i] = 1;
            for (int j = 1; j < i; j++) {
                row[j] = retval[i-1][j] + retval[i-1][j-1];
            }
            retval.push_back(row);
        }
    }
    return retval;
}
```

```
int main() {
    int n = 0;
    cin >> n;

    vector<vector<int>> t = pascal_triangle(n);
    for (const auto &row : t) {
        for (const auto &val : row) {
            cout << val << " ";
        }
        cout << endl;
    }

    return 0;
}
```

Output :

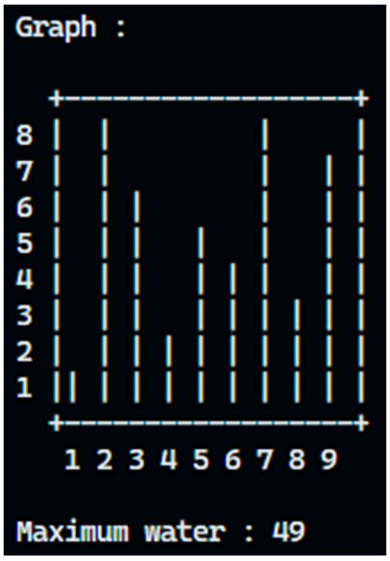


A3. Container With Most Water.

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

using namespace std;

int max_water(vector<int>& heights) {
    int retval = 0;
    int i      = 0;
    int j      = heights.size() - 1;

    while (i < j) {
        int water = (j - i) * min(heights[i], heights[j]);
        retval = max(retval, water);

        if (heights[i] < heights[j]) { i++; }
        else                         { j--; }
    }

    return retval;
}

void plot_graph(vector<int>& heights) {
    int max = *max_element(heights.begin(), heights.end()) + 1;
    int heights_size = heights.size();

    char symbols[max][heights_size];

    for (int j = heights_size - 1; j > -1; j--) {
        int height = heights[j];
        for (int i = max - 1; i > -1; i--) {
```

```
            if (height > 0) { symbols[i][j] = '|'; }
            else            { symbols[i][j] = ' '; }
            height--;
        }
    }
    cout << endl;
    cout << "  +";
    for (int j = 0; j < heights_size * 2; j++) {
        cout << "-";
    }
    cout << "+" << endl;
    for (int i = 1; i < max; i++) {
        cout << max - i << " |";
        for (int j = 0; j < heights_size; j++) {
            cout << symbols[i][j] << " ";
        }
        cout << "|" << endl;
    }
    cout << "  +";
    for (int j = 0; j < heights_size * 2; j++) {
        cout << "-";
    }
    cout << "+" << endl;
    cout << "    ";
    for (int j = 0; j < heights_size; j++) {
        cout << j + 1 << " ";
    }
}

int main(int argc, char* argv[]) {
    cout << "\nGraph : \n";
    vector<int> heights = { 1, 8, 6, 2, 5, 4, 8, 3, 7 };
    plot_graph(heights);
    cout << "\n\nMaximum water : " << max_water(heights) << endl << endl;
    return 0;
}
```

Output :



A4. Maximum Number of Groups Getting Fresh Donuts

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

int dfs(vector<int>& arr, int left, int K) {
    int q = 0;

    if (left == 0) {
        for (int i = 1; i < K; ++i) {
            if (arr[i] > 0) {
                arr[i]--;
                q = max(q, 1 + dfs(arr, K - i, K));
                arr[i]++;
            }
        }
    } else {
        for (int i = 1; i < K; ++i) {
            if (arr[i] > 0) {
                arr[i]--;
                int nleft = (i <= left ? left - i : K + left - i);
                q = max(q, dfs(arr, nleft, K));
                arr[i]++;
            }
        }
    }
    return q;
}

int maxGroups(int K, const vector<int>& arr) {
    vector<int> V(K, 0);

    for (int x : arr) { V[x % K]++; }

    int ans = V[0] + dfs(V, 0, K);
    return ans;
}

int main() {
    vector<int> arr = {1, 2, 3, 4, 5, 6};
    int K = 3;

    cout << maxGroups(K, arr) << endl;
    return 0;
}
```

Output :

```
Satisfied Groups : 4
```

A5. Find Minimum Time to Finish All Jobs

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

using namespace std;

bool can_assign_jobs(vector<int>& jobs, int k, long long maxTime) {
    int workers = 1;
    long long current_load = 0;

    for (int job : jobs) {
        if (job > maxTime) return false;

        if (current_load + job > maxTime) {
            workers++;
            current_load = job;
            if (workers > k) return false;
        } else {
            current_load += job;
        }
    }
    return true;
}

long long minimum_time_required(vector<int>& jobs, int k) {
    sort(jobs.rbegin(), jobs.rend());

    long long left = jobs[0];  // Minimum possible time is the largest job
    long long right = accumulate(jobs.begin(), jobs.end(), 0LL);

    while (left < right) {
        long long mid = left + (right - left) / 2;
        if (can_assign_jobs(jobs, k, mid)) {
            right = mid;
        } else {
            left = mid + 1;
        }
    }
    return left;
}

int main() {
    vector<int> jobs1 = {3, 2, 3};
    int k1 = 3;
    cout << minimum_time_required(jobs1, k1) << endl;
    return 0;
}
```

Output :

```
Minimum time required : 3
```