AMAN BANSAL

22BCS13365

KPIT- 901/B

Day 2

Array & Linked list

Very Easy

Q :1 Majority Elements

Given an array num of size n, return the majority element.
The majority element is the element that appears more than ⌊n / 2⌋ times. You may assume that the majority element always exists in the array.

Example 1:
Input: nums = [3,2,3]
Output: 3

Example 2:
Input: nums = [2,2,1,1,1,2,2]
Output: 2

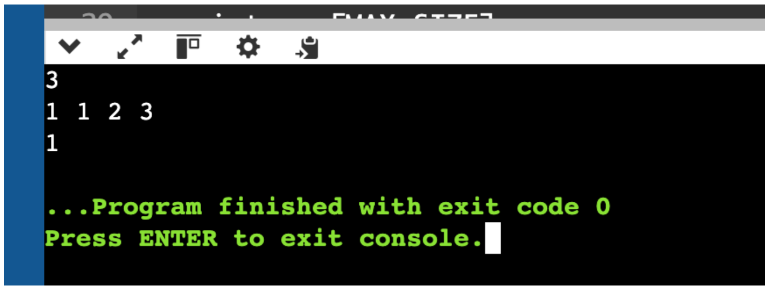
Constraints:
n == nums.length
1 <= n <= 5 * 104
-109 <= nums[i] <= 109

Follow-up: Could you solve the problem in linear time and in O(1) space?

SOLUTION:

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

```
#define ll long long

const int MAX_SIZE = 1000;

int repeatNumbers(int arr[],int size){
    for(int t =0;t<size;t++){
       int count =0;
            for(int i=0;i<size;i++){
           if(arr[t] == arr[i]){
               count++;
           }
            }
            if(count >size/2){
                 return arr[t];
            }
            }
 return -1;
}
int main(){
       int size;
       cin>>size;
        if (size > MAX_SIZE) {
        cout << "Error : Size above maximum"<< MAX_SIZE << endl;
        return 1;
    }
    int arr[MAX_SIZE];
    for (int i = 0; i < size; i++) {
        cin >> arr[i];
    }
    cout<<repeatNumbers(arr,size);

return 0;
}
```



```
3
1 1 2 3
1

...Program finished with exit code 0
Press ENTER to exit console.
```

Easy

Question:. Pascal's Triangle

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

In Pascal's triangle, each number is the sum of the two numbers directly above it as shown:

Example 1:

Input: numRows = 5
Output: [[1],[1,1],[1,2,1],[1,3,3,1],[1,4,6,4,1]]

Example 2:

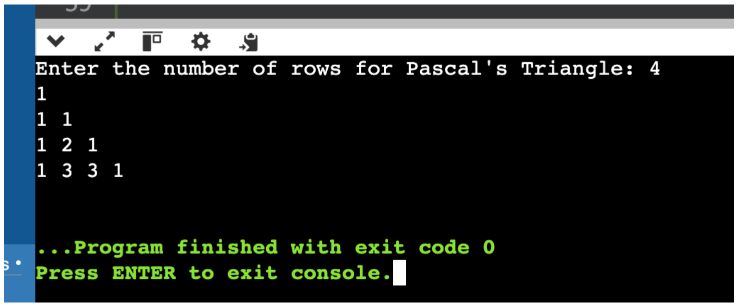
Input: numRows = 1
Output: [[1]]

Constraints:

1 <= numRows <= 30

SOLUTION:

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

void generatePascalsTriangle(int numRows) {
    int pascalsTriangle[30][30] = {0};

    for (int i = 0; i < numRows; i++) {
        pascalsTriangle[i][0] = 1;
        pascalsTriangle[i][i] = 1;

        for (int j = 1; j < i; j++) {
            pascalsTriangle[i][j] = pascalsTriangle[i - 1][j - 1] + pascalsTriangle[i - 1][j];
        }
    }

    for (int i = 0; i < numRows; i++) {
        for (int j = 0; j <= i; j++) {
            cout << pascalsTriangle[i][j] << " ";
        }
        cout << endl;
    }
}

int main() {
    int numRows;
    cout << "Enter the number of rows for Pascal's Triangle: ";
    cin >> numRows;

    if (numRows < 1 || numRows > 30) {
        cout << "Invalid input. Please enter a value between 1 and 30." << endl;
        return 1;
    }

    generatePascalsTriangle(numRows);

    return 0;
}
```



```
Enter the number of rows for Pascal's Triangle: 4
1
1 1
1 2 1
1 3 3 1


...Program finished with exit code 0
Press ENTER to exit console.
```

Medium

Question: 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 ith 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.

Notice that you may not slant the container.

Example 1:

Input: height = [1,8,6,2,5,4,8,3,7]
Output: 49
Explanation: The above vertical lines are represented by array [1,8,6,2,5,4,8,3,7]. In this case, the max area of water (blue section) the container can contain is 49.

Example 2:

Input: height = [1,1]
Output: 1

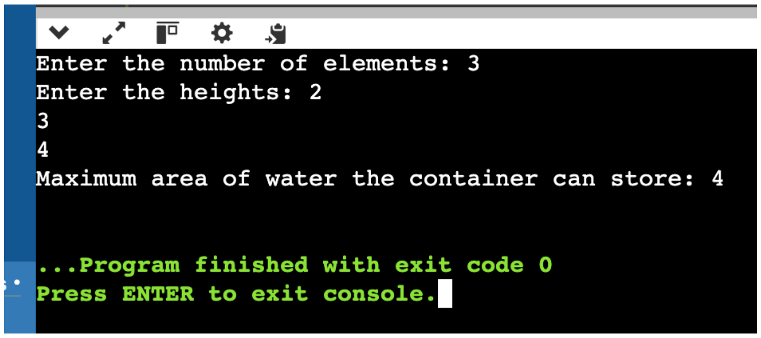
Constraints:

n == height.length
2 <= n <= 105
0 <= height[i] <= 104

SOLUTION:

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

```
int maxArea(int height[], int n) {
    int left = 0;
    int right = n - 1;
    int max_area = 0;
    while (left < right) {
        int width = right - left;
        int area = min(height[left], height[right]) * width;
        max_area = max(max_area, area);

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

    return max_area;
}

int main() {
    int n;
    cout << "Enter the number of elements: ";
    cin >> n;

    if (n < 2 || n > 100000) {
        cout << "Invalid input. Number of elements should be between 2 and 100000." << endl;
        return 1;
    }

    int height[n];
    cout << "Enter the heights: ";
    for (int i = 0; i < n; i++) {
        cin >> height[i];
        if (height[i] < 0 || height[i] > 10000) {
            cout << "Invalid input. Heights should be between 0 and 10000." << endl;
            return 1;
        }
    }
    int result = maxArea(height, n);
    cout << "Maximum area of water the container can store: " << result << endl;

    return 0;
}
```



```
Enter the number of elements: 3
Enter the heights: 2
3
4
Maximum area of water the container can store: 4


...Program finished with exit code 0
Press ENTER to exit console.
```

Hard

Question: Maximum Number of Groups Getting Fresh Donuts

There is a donuts shop that bakes donuts in batches of batchSize. 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 batchSize 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.

You can freely rearrange the ordering of the groups. Return the maximum possible number of happy groups after rearranging the groups.

Example 1:

Input: batchSize = 3, groups = [1,2,3,4,5,6]
Output: 4
Explanation: You can arrange the groups as [6,2,4,5,1,3]. Then the 1st, 2nd, 4th, and 6th groups will be happy.

Example 2:

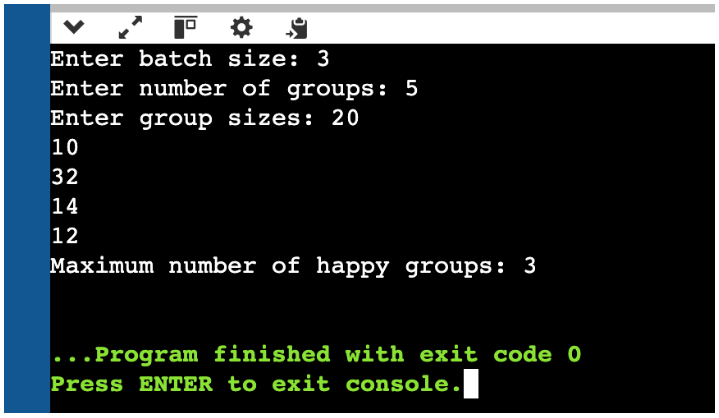
Input: batchSize = 4, groups = [1,3,2,5,2,2,1,6]
Output: 4

Constraints:

1 <= batchSize <= 9
1 <= groups.length <= 30
1 <= groups[i] <= 109

SOLUTION:

```
#include <iostream>
#include <cstring>

using namespace std;

int solve(int batchSize, int remainders[], int remainderSum) {
    int happyGroups = 0;
    happyGroups += remainders[0];
    remainders[0] = 0;

    for (int i = 1; i < batchSize; i++) {
        if (remainders[i] > 0) {
            int complement = batchSize - i;
            if (complement < batchSize) {
                int pairs = min(remainders[i], remainders[complement]);
                happyGroups += pairs;
                remainders[i] -= pairs;
                remainders[complement] -= pairs;
            }
        }
    }

    bool extra = false;
    for (int i = 1; i < batchSize; i++) {
        if (remainders[i] > 0) {
            extra = true;
            happyGroups++;
            break;
        }
    }

    return happyGroups;
}

int maxHappyGroups(int batchSize, int groups[], int n) {
    int remainders[batchSize];
    memset(remainders, 0, sizeof(remainders));

    for (int i = 0; i < n; i++) {
        remainders[groups[i] % batchSize]++;
    }

    return solve(batchSize, remainders, n);
}

int main() {
    int batchSize, n;

    cout << "Enter batch size: ";
    cin >> batchSize;
    cout << "Enter number of groups: ";
    cin >> n;

    int groups[n];
    cout << "Enter group sizes: ";
    for (int i = 0; i < n; i++) {
        cin >> groups[i];
    }

    cout << "Maximum number of happy groups: " << maxHappyGroups(batchSize, groups, n)
<< endl;

    return 0;
}
```



```
Enter batch size: 3
Enter number of groups: 5
Enter group sizes: 20
10
32
14
12
Maximum number of happy groups: 3


...Program finished with exit code 0
Press ENTER to exit console.
```

Very Hard

Question 1. Find Minimum Time to Finish All Jobs

You are given an integer array jobs, where jobs[i] is the amount of time it takes to complete the ith 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.

Example 1:

Input: jobs = [3,2,3], k = 3
Output: 3
Explanation: By assigning each person one job, the maximum time is 3.

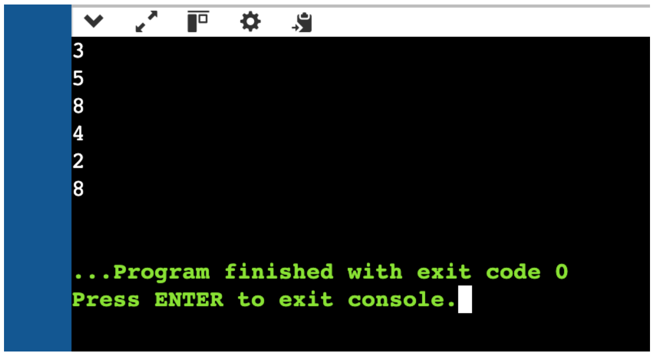
Example 2:

Input: jobs = [1,2,4,7,8], k = 2
Output: 11
Explanation: Assign the jobs the following way:
Worker 1: 1, 2, 8 (working time = 1 + 2 + 8 = 11)
Worker 2: 4, 7 (working time = 4 + 7 = 11)
The maximum working time is 11.
Constraints:

1 <= k <= jobs.length <= 12
1 <= jobs[i] <= 107

SOLUTION:

```cpp
#include <iostream>
#include <algorithm>
#include <numeric> // Include this header for accumulate
#include <cstring>

using namespace std;

bool canAssign(int jobs[], int n, int k, int workerLoad[], int maxTime, int index) {
    if (index == n) return true;
    for (int i = 0; i < k; i++) {
        if (workerLoad[i] + jobs[index] <= maxTime) {
            workerLoad[i] += jobs[index];
            if (canAssign(jobs, n, k, workerLoad, maxTime, index + 1)) return true;
            workerLoad[i] -= jobs[index];
        }
        if (workerLoad[i] == 0) break;
    }
    return false;
}

int findMinimumTime(int jobs[], int n, int k) {
    int low = *max_element(jobs, jobs + n);
    int high = accumulate(jobs, jobs + n, 0);
    int result = high;
    while (low <= high) {
        int mid = low + (high - low) / 2;
        int workerLoad[k];
        memset(workerLoad, 0, sizeof(workerLoad));
        if (canAssign(jobs, n, k, workerLoad, mid, 0)) {
            result = mid;
            high = mid - 1;
        } else {
            low = mid + 1;
        }
    }
    return result;
}

int main() {
    int n, k;
    cin >> n >> k;
    int jobs[n];
    for (int i = 0; i < n; i++) {
        cin >> jobs[i];
    }
    cout << findMinimumTime(jobs, n, k) << endl;
    return 0;
}
```



```
3
5
8
4
2
8


...Program finished with exit code 0
Press ENTER to exit console.
```