Assignment 2

Name – Ishita Faculty Name- Er. Rajni Devi

UID- 22BCS15353 Date- 20 Dec ,2024

Section – 620-B

Very Easy

Ques 1:

Majority Elements

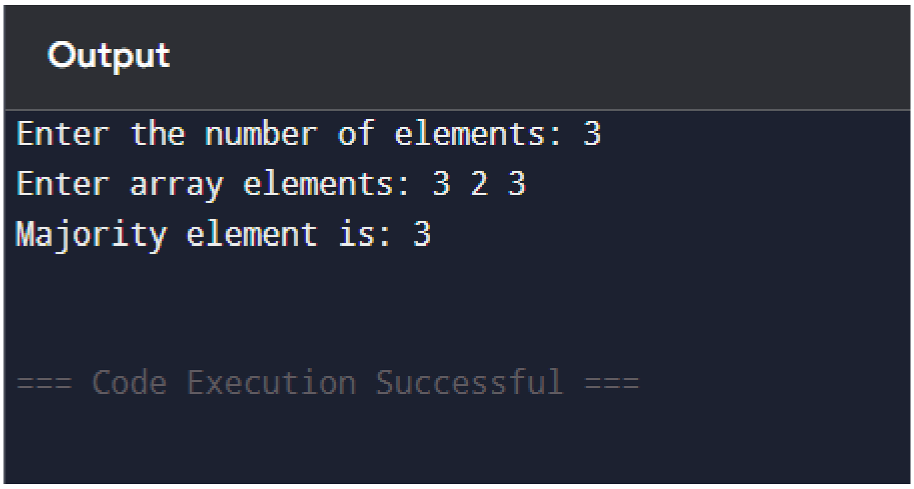
Given an array nums of size n, return the majority element.

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.

Code:

```
#include <iostream>
using namespace std;
int main() {
    int n;
    cout << "Enter the number of elements: ";
    cin >> n;
    int arr[n];
    cout << "Enter array elements: ";
    for (int i = 0; i < n; i++) {
        cin >> arr[i];
    }
    int majorityElement = -1;
    int maxCount = 0;
    for (int i = 0; i < n; i++) {
        int count = 0;
        for (int j = 0; j < n; j++) {
            if (arr[i] == arr[j]) {
                count++;
            }
        }
        if (count > n / 2) {
            majorityElement = arr[i];
            break;
        }
    }
    if (majorityElement != -1) {
        cout << "Majority element is: " << majorityElement << endl;
    } else {
        cout << "No majority element found." << endl;
    }
    return 0;
}
```

Output:



Output

```
Enter the number of elements: 3
Enter array elements: 3 2 3
Majority element is: 3


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

<u>Ques 2:</u>

Merge Two Sorted Lists

You are given the heads of two sorted linked lists list1 and list2.

Merge the two lists into one **sorted** list. The list should be made by splicing together the nodes of the first two lists.

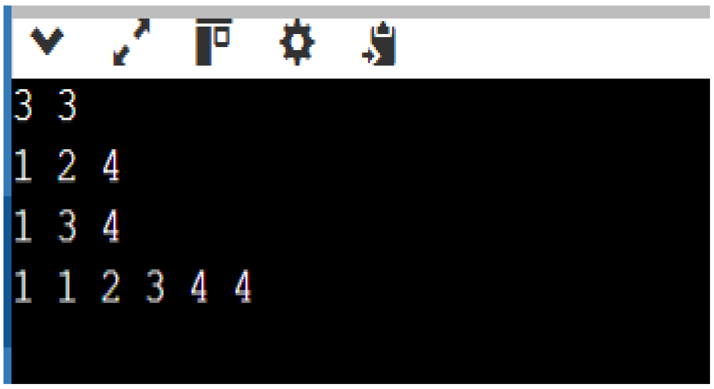
Return *the head of the merged linked list*.

Code:

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

```
struct ListNode {
    int val;
    ListNode* next;
    ListNode(int x) : val(x), next(nullptr) {}
};
ListNode* mergeTwoLists(ListNode* l1, ListNode* l2) {
    if (!l1 || !l2) return l1 ? l1 : l2;
    if (l1->val < l2->val) return l1->next = mergeTwoLists(l1->next, l2), l1;
    return l2->next = mergeTwoLists(l1, l2->next), l2;}
ListNode* createList(int n) {
    ListNode *head = nullptr, *tail = nullptr;
    while (n--) {
        int x; cin >> x;
        if (!head) head = tail = new ListNode(x);
        else tail = tail->next = new ListNode(x);}
    return head;}
void printList(ListNode* head) {
    while (head) cout << head->val << " ", head = head->next;}
int main() {
    int n, m;
    cin >> n >> m;
    ListNode* l1 = createList(n), *l2 = createList(m);
    printList(mergeTwoLists(l1, l2));
}
```

Output:



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

Easy

Ques 3:

Pascal's Triangle

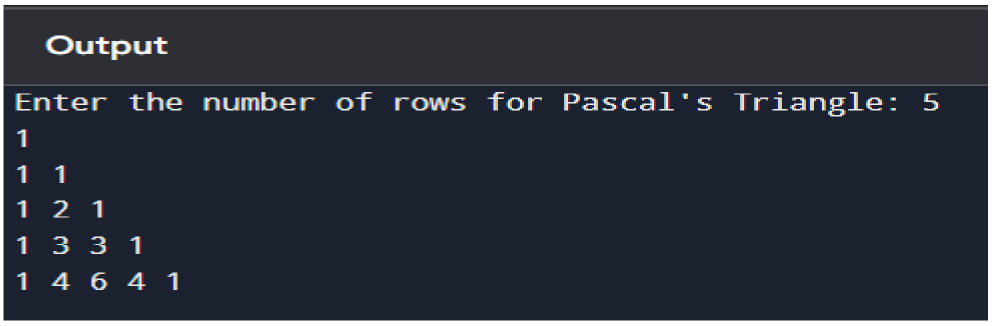
Given an integer numRows, return the first numRows of Pascal's triangle.

In Pascal's triangle, each number is the sum of the two numbers directly .

Code:

```
#include <iostream>

using namespace std;

void printPascalsTriangle(int rows) {

    for (int i = 0; i < rows; i++) {

        int num = 1;

        for (int j = 0; j <= i; j++) {

            cout << num << " ";

            num = num * (i - j) / (j + 1);

        }

        cout << endl;}}

int main() {

    int rows;

    cout << "Enter the number of rows for Pascal's Triangle: ";

    cin >> rows;

    printPascalsTriangle(rows);

    return 0;}
```

Output:



```
Enter the number of rows for Pascal's Triangle: 5
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
```

Ques 4:

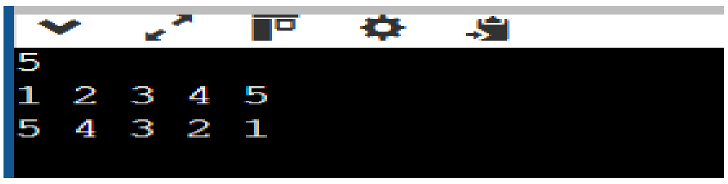
Reverse Linked List

Given the head of a singly linked list, reverse the list, and return *the reversed list*.

Code:

```
#include <iostream>
using namespace std;
struct ListNode {
    int val;
    ListNode* next;
    ListNode(int x) : val(x), next(nullptr) {}};
ListNode* reverseList(ListNode* head) {
    ListNode *prev = nullptr, *next = nullptr;
    while (head) {
        next = head->next;
        head->next = prev; prev = head;
        head = next;}
    return prev;}
void printList(ListNode* head) {
    while (head) cout << head->val << " ", head = head->next}
int main() {
    int n, x; cin >> n;
    ListNode *head = nullptr, *tail = nullptr;
    while (n--) {
        cin >> x;
        if (!head) head = tail = new ListNode(x);
        else tail = tail->next = new ListNode(x);}
    head = reverseList(head);
    printList(head);}
```

Output:



```
5
1 2 3 4 5
5 4 3 2 1
```

<u>**Medium**</u>

Ques 5:

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.

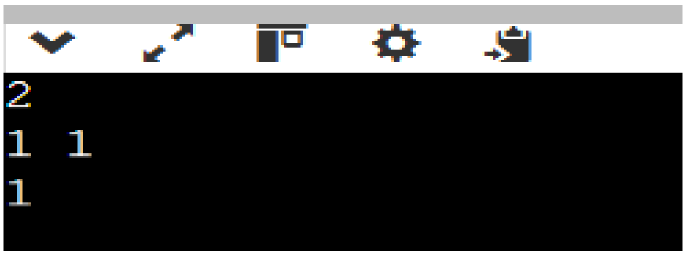
Return the maximum amount of water a container can store.

Notice that you may not slant the container.

Code:

```
#include <iostream>

using namespace std;

int maxArea(int height[], int n) {

    int left = 0, right = n - 1, maxWater = 0;

    while (left < right) {

        int h = min(height[left], height[right]);

        maxWater = max(maxWater, h * (right - left));

        if (height[left] < height[right]) left++;

        else right--;}

    return maxWater;}

     int main() {

    int n;

    cin >> n; // Number of lines

    int height[n];

    for (int i = 0; i < n; i++) cin >> height[i];

    cout << maxArea(height, n);

    return 0;}
```

Output:



```
2
1 1
1
```

Ques 6:

Jump Game II

You are given a 0-indexed array of integers nums of length n. You are initially positioned at nums[0]. Each element nums[i] represents the maximum length of a forward jump from index i. In other words, if you are at nums[i], you can jump to any nums[i + j]

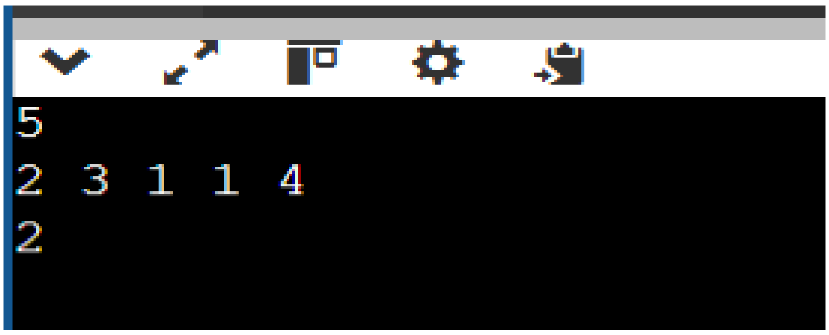
where: 0 <= j <= nums[i] and i + j < n

Return the minimum number of jumps to reach nums[n - 1]. The test cases are generated such that you can reach nums[n - 1].

Code:

```
#include <iostream>
using namespace std;
int jump(int nums[], int n) {
   int jumps = 0, farthest = 0, end = 0;
   for (int i = 0; i < n - 1; i++) {
      farthest = max(farthest, i + nums[i]);
      if (i == end) {
         jumps++;
         end = farthest:}}
   return jumps;}
  int main() {
   int n;
   cin >> n; // Size of the array
   int nums[n];
   for (int i = 0; i < n; i++) cin >> nums[i];
   cout << jump(nums, n);
   return 0;}
```

Output:



```
5
2 3 1 1 4
2
```

Hard

Ques 7:

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.

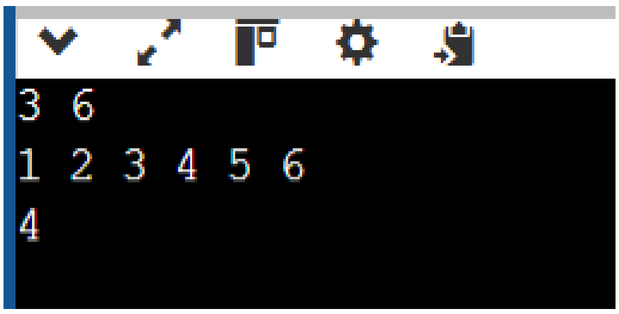
When a group visits the shop, all customers of the group must be served before serving any of the following groups. A group will be happy if they all get fresh donuts. That is, the first customer of the group does not receive a donut that was left over from the previous group.

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

Code:

```
#include <iostream>

using namespace std;

int maxHappyGroups(int batchSize, int groups[], int n) {

    int happy = 0;

    int remainder[batchSize] = {0};

    for (int i = 0; i < n; i++) {

        if (groups[i] % batchSize == 0) {

            happy++;  // These groups will always be happy

        } else {

            remainder[groups[i] % batchSize]++;  // Track remainder groups

        }

    }

    for (int i = 1; i < batchSize; i++) {

        int complement = batchSize - i;

        if (remainder[i] > 0 && remainder[complement] > 0) {

            int pairs = min(remainder[i], remainder[complement]);

            happy += pairs;

            remainder[i] -= pairs;
            remainder[complement] -= pairs;
        }
    }
    happy += remainder[0] / batchSize;
    return happy;
}
 int main() {
    int n, batchSize;
    cin >> batchSize >> n;  // Read batchSize and number of groups
    int groups[n];
    for (int i = 0; i < n; i++) cin >> groups[i];
    cout << maxHappyGroups(batchSize, groups, n);
    return 0;
}
```

Output:



```
3 6
1 2 3 4 5 6
4
```

Ques 8:

Maximum Number of Darts Inside of a Circular Dartboard

Alice is throwing n darts on a very large wall. You are given an array darts where darts[i] = [xi, yi] is the position of the ith dart that Alice threw on the wall.

Bob knows the positions of the n darts on the wall. He wants to place a dartboard of radius r on the wall so that the maximum number of darts that Alice throws lie on the dartboard.

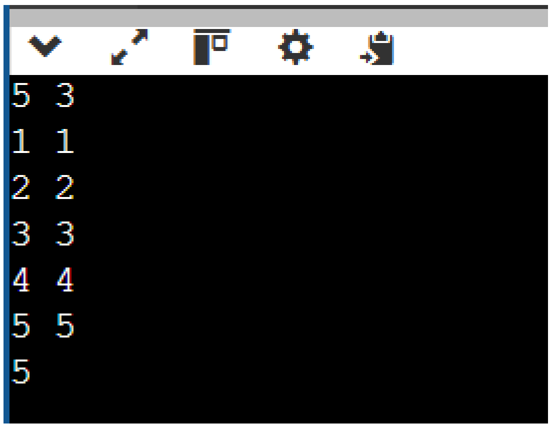
Given the integer r, return the maximum number of darts that can lie on the dartboard.

Code:

```
#include <iostream>
```

```
#include <cmath>
using namespace std;
int maxDartsInDartboard(int darts[][2], int n, int r) {
    int maxDarts = 0;
    for (int i = 0; i < n; i++) {
        int count = 0;
        for (int j = 0; j < n; j++) {
            // Calculate distance from dart i to dart j
            int dist = pow(darts[i][0] - darts[j][0], 2) + pow(darts[i][1] - darts[j][1], 2);
            if (dist <= r * r) count++;
        }
        maxDarts = max(maxDarts, count);}
    return maxDarts;}
     int main() {
    int n, r;
    cin >> n >> r;  // Number of darts and radius of dartboard
    int darts[n][2]; // Array to store the positions of darts
    for (int i = 0; i < n; i++) cin >> darts[i][0] >> darts[i][1];
    cout << maxDartsInDartboard(darts, n, r);
    return 0;
}
```

Output:



```
5 3
1 1
2 2
3 3
4 4
5 5
5
```

Very Hard

Ques 9:

Insert Greatest Common Divisors in Linked List

Given the head of a linked list head, in which each node contains an integer value.

Between every pair of adjacent nodes, insert a new node with a value equal to the greatest common divisor of them.

Return *the linked list after insertion*.

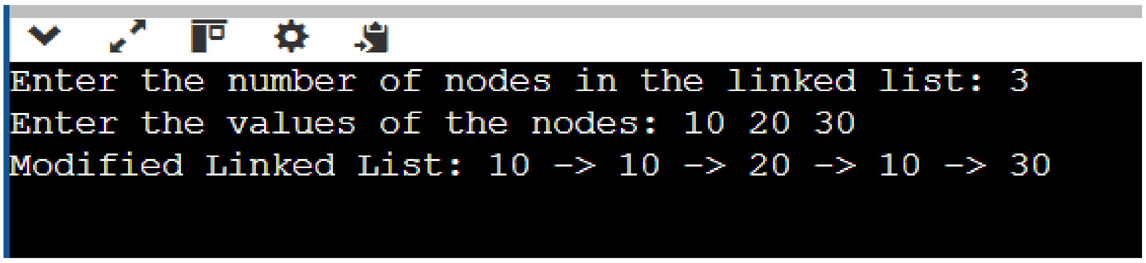
The greatest common divisor of two numbers is the largest positive integer that evenly divides both numbers.

Code:

```
#include <iostream>

#include <algorithm>

using namespace std;

struct ListNode {

   int val;

   ListNode* next;

   ListNode(int x) : val(x), next(nullptr) {}};

int gcd(int a, int b) {

   while (b != 0) {

      int temp = b;

      b = a % b;

      a = temp;}

   return a;}

   ListNode* insertGCDNodes(ListNode* head) {

   ListNode* curr = head;

   while (curr != nullptr && curr->next != nullptr) {

      int gcdValue = gcd(curr->val, curr->next->val);

      ListNode* newNode = new ListNode(gcdValue);

      newNode->next = curr->next;

      curr->next = newNode;
```

```
        curr = curr->next->next; // Move two steps ahead
    return head;}
     void printList(ListNode* head) {
    cout << "Modified Linked List: ";
    while (head != nullptr) {
        cout << head->val;
        if (head->next != nullptr) cout << " -> ";
        head = head->next;}
    cout << endl;}
    int main() {
    int n, val;
    cout << "Enter the number of nodes in the linked list: ";
    cin >> n; // Number of nodes
    cout << "Enter the values of the nodes: ";
    cin >> val;
    ListNode* head = new ListNode(val);
    ListNode* tail = head;
    for (int i = 1; i < n; i++) {
        cin >> val;
        tail->next = new ListNode(val);
        tail = tail->next;}
    head = insertGCDNodes(head);
    printList(head);  // Print the modified list
    return 0;}
```

Output:



```
Enter the number of nodes in the linked list: 3
Enter the values of the nodes: 10 20 30
Modified Linked List: 10 -> 10 -> 20 -> 10 -> 30
```

Ques 10:

Find Minimum Time to Finish All Jobs

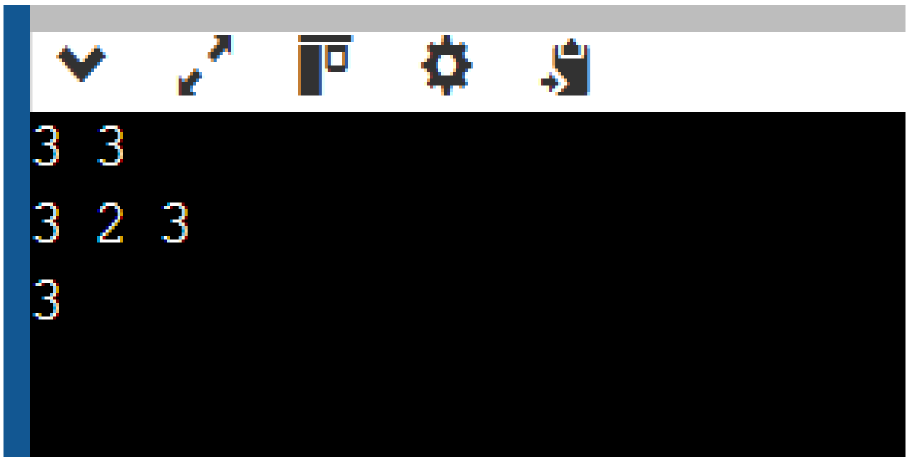
You are given an integer array jobs, where jobs[i] is the amount of time it takes to complete the ith job.There are k workers that you can assign jobs to. Each job should be assigned to exactly one worker. The working time of a worker is the sum of the time it takes to complete all jobs assigned to them. 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.

Code:

```
#include <iostream>
using namespace std;
bool dfs(int* jobs, int* workers, int n, int k, int maxTime, int i) {
    if (i == n) return true;
    for (int j = 0; j < k; j++) {
        if (workers[j] + jobs[i] <= maxTime) {
            workers[j] += jobs[i];
            if (dfs(jobs, workers, n, k, maxTime, i + 1)) return true;
            workers[j] -= jobs[i];
        }
        if (workers[j] == 0) break;  // Optimization to avoid unnecessary checks
    }
    return false;
}
bool canAssignJobs(int* jobs, int n, int k, int maxTime) {
    int* workers = new int[k]();  // Array to track work assigned to each worker
    bool result = dfs(jobs, workers, n, k, maxTime, 0);
    delete[] workers;  // Clean up dynamically allocated memory
    return result;
}
int minimumWorkingTime(int* jobs, int n, int k) {
    int maxJobTime = jobs[0], totalJobTime = 0;
    for (int i = 0; i < n; i++) {
        if (jobs[i] > maxJobTime) maxJobTime = jobs[i];
        totalJobTime += jobs[i];
    }
    int left = maxJobTime, right = totalJobTime;
    while (left < right) {
        int mid = left + (right - left) / 2;
        if (canAssignJobs(jobs, n, k, mid)) right = mid;
        else left = mid + 1;
    }
    return left;
}

int main() {
    int k, n;
    cin >> k >> n;  // Number of workers and number of jobs
    int* jobs = new int[n];  // Dynamic array for jobs
    for (int i = 0; i < n; i++) cin >> jobs[i];  // Input job times

    cout << minimumWorkingTime(jobs, n, k) << endl;  // Output the result
    delete[] jobs;  // Clean up dynamically allocated memory
    return 0;
}
```

Output:



```
3 3
3 2 3
3
```