NAME: Lakshay Verma UID: 22BCS15481

SECTION: 620/B

DAY - 2

Solution 1. Single Number

Code:

```
#include <iostream>
#include <vector>
using namespace std;
int singleNumber(vector<int>& nums) {
    int result = 0;
    for (int num : nums) {
        result ^= num;
    }
    return result;
}
int main() {
    vector<int> nums = {4, 1, 2, 1, 2};
    cout << "Single Number: " << singleNumber(nums) << endl;
    return 0;
}
```

Output:

```
Single Number: 4
```

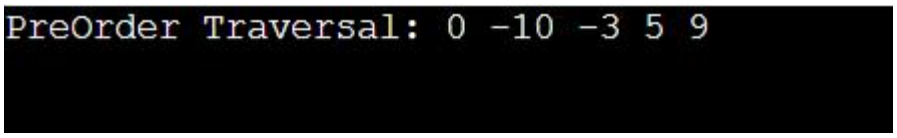
Solution 2. Convert Sorted Array to BST

Code:

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

```
using namespace std;
struct TreeNode {
    int val;
    TreeNode* left;
    TreeNode* right;
    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
};
TreeNode* sortedArrayToBST(vector<int>& nums, int left, int right) {
    if (left > right) return nullptr;
    int mid = left + (right - left) / 2;
    TreeNode* root = new TreeNode(nums[mid]);
    root->left = sortedArrayToBST(nums, left, mid - 1);
    root->right = sortedArrayToBST(nums, mid + 1, right);
    return root;
}
TreeNode* sortedArrayToBST(vector<int>& nums) {
    return sortedArrayToBST(nums, 0, nums.size() - 1);
}
void preOrder(TreeNode* node) {
    if (!node) return;
    cout << node->val << " ";
    preOrder(node->left);
    preOrder(node->right);
}
int main() {
    vector<int> nums = {-10, -3, 0, 5, 9};
    TreeNode* root = sortedArrayToBST(nums);
    cout << "PreOrder Traversal: ";
    preOrder(root);
    cout << endl;
    return 0;
}
```

Output:



Solution 3. Merge Two Sorted Lists

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) return l2;
    if (!l2) return l1;
    if (l1->val < l2->val) {
        l1->next = mergeTwoLists(l1->next, l2);
        return l1;
    } else {
        l2->next = mergeTwoLists(l1, l2->next);
        return l2;
    }
}
void printList(ListNode* head) {
    while (head) {
        cout << head->val << " ";
        head = head->next;
    }
```

```
    cout << endl;
}
int main() {
    ListNode* l1 = new ListNode(1);
    l1->next = new ListNode(2);
    l1->next->next = new ListNode(4);
    ListNode* l2 = new ListNode(1);
    l2->next = new ListNode(3);
    l2->next->next = new ListNode(4);
    ListNode* result = mergeTwoLists(l1, l2);
    cout << "Merged List: ";
    printList(result);
    return 0;
}
```

Output:

```
Merged List: 1 1 2 3 4 4
```

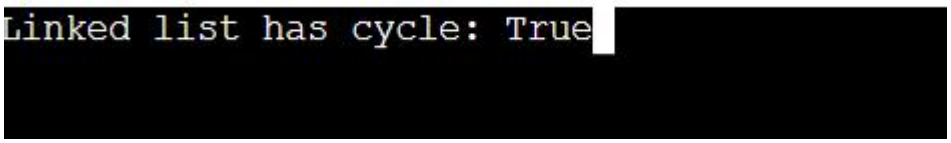
Solution 4. Linked List Cycle

Code:

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

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

bool hasCycle(ListNode* head) {
    ListNode* slow = head;
    ListNode* fast = head;
```

```
    while (fast && fast->next) {
        slow = fast->next;
        fast = fast->next->next;
        if (slow == fast) return true;
    }
    return false;
}

int main() {
    ListNode* head = new ListNode(3);
    head->next = new ListNode(2);
    head->next->next = new ListNode(0);
    head->next->next->next = new ListNode(-4);
    head->next->next->next->next = head->next; // Creating a cycle

    cout << "Linked List has cycle: " << (hasCycle(head) ? "True" :
"False") << endl;
    return 0;
}
```

Output:



Solution 5. Pascal's Triangle

Code:

```
#include <iostream>
#include <vector>
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); // Create a row with 'i+1'
elements initialized to 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;
}
int main() {
    int numRows = 5;
vector<vector<int>> result = generate(numRows);
    for (auto row : result) {
        for (int num : row) {
            cout << num << " ";
        }
        cout << endl;
    }
    return 0;
}
```

OUTPUT:

Solution 6. Container With Most Water

```
#include <iostream>
#include <vector>
#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 area = min(height[left], height[right]) * (right - left);
        max_area = max(max_area, area);
        if (height[left] < height[right]) {
            left++;
        } else {
            right--;
        }
    }
    return max_area;
}

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

Output:

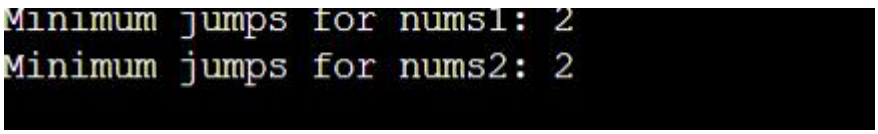
Solution 7. Jump Game II

Code :

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

```
using namespace std;
int jump(vector<int>& nums) {
    int n = nums.size();
    int jumps = 0;
    int farthest = 0;
    int end = 0;
    for (int i = 0; i < n - 1; i++) {
        farthest = max(farthest, i + nums[i]);
        if (i == end) {
            jumps++;
            end = farthest;
            if (end >= n - 1) break;
        }
    }
    return jumps;
}
int main() {
    vector<int> nums1 = {2, 3, 1, 1, 4};
    vector<int> nums2 = {2, 3, 0, 1, 4};
    cout << "Minimum jumps for nums1: " << jump(nums1) << endl;
    cout << "Minimum jumps for nums2: " << jump(nums2) << endl;
    return 0;
}
```

Output:



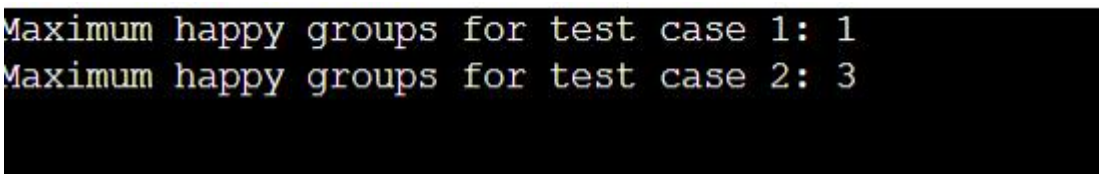
Solution 8. Maximum Number of Groups Getting Fresh Donuts

Code:

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

```
using namespace std;
int maxHappyGroups(int batchSize, vector<int>& groups) {
    int happyCount = 0;
    int currentDonuts = 0;
    sort(groups.begin(), groups.end(), greater<int>());
    for (int groupSize : groups) {
        if (groupSize <= currentDonuts) {
            currentDonuts -= groupSize;
            happyCount++;
        } else {
            currentDonuts = batchSize - groupSize;
        }
    }
    return happyCount;
}
int main() {
    vector<int> groups1 = {1, 2, 3, 4, 5, 6};
    vector<int> groups2 = {1, 3, 2, 5, 2, 2, 1, 6};
    cout << "Maximum happy groups for test case 1: " <<
maxHappyGroups(3, groups1) << endl;
    cout << "Maximum happy groups for test case 2: " <<
maxHappyGroups(4, groups2) << endl;
return 0;
}
```

OUTPUT:



Solution 9. Find Minimum Time to Finish All Jobs

Code :

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

```
#include <algorithm>
using namespace std;
bool canAssignJobs(const vector<int>& jobs, int k, int maxTime) {
    vector<int> workers(k, 0); // worker workload
    return backtrack(jobs, 0, workers, maxTime);
}
bool backtrack(const vector<int>& jobs, int index, vector<int>&
workers, int maxTime) {
    if (index == jobs.size()) return true;
    int job = jobs[index];
    for (int i = 0; i < workers.size(); i++) {
        if (workers[i] + job > maxTime) continue;
        workers[i] += job;
        if (backtrack(jobs, index + 1, workers, maxTime)) return
true;
        workers[i] -= job;
        if (workers[i] == 0) break;
    }
    return false;
}
int minTimeToFinishJobs(vector<int>& jobs, int k) {
    int left = *max_element(jobs.begin(), jobs.end());
    int right = accumulate(jobs.begin(), jobs.end(), 0);
    sort(jobs.begin(), jobs.end(), greater<int>());
    while (left < right) {
        int mid = left + (right - left) / 2;
        if (canAssignJobs(jobs, k, mid)) {
            right = mid;
        } else {
            left = mid + 1;
        }
    }
    return left;
```

```
}
int main() {
    vector<int> jobs1 = {3, 2, 3};
    int k1 = 3;
    vector<int> jobs2 = {1, 2, 4, 7, 8};
    int k2 = 2;
    cout << "Minimum max working time for jobs1: " <<
minTimeToFinishJobs(jobs1, k1) << endl;
    cout << "Minimum max working time for jobs2: " <<
minTimeToFinishJobs(jobs2, k2) << endl;
    return 0;
}
```

OUTPUT:

```
Minimum max working time for jobs1: 3
Minimum max working time for jobs2: 11
```

Solution 10. Majority Element

Code:

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

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

int main() {
    vector<int> nums = {3, 2, 3};
```

```
    cout << "Majority Element: " << majorityElement(nums) << endl;
    return 0;
}
```

Output: