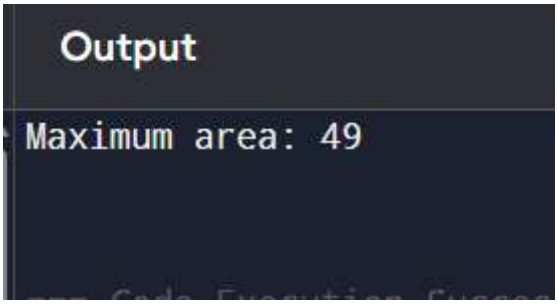
Day 2 UID- 22BCS10188

Q1 Container With Most Water

```cpp
/* You are given an integer array height of length n. There are n vertical lines drawn s
Find two lines that together with the x-axis form a container, such that the container c
Return the maximum amount of water a container can store.
Notice that you may not slant the container.  */

#include <iostream>
#include <vector>

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

    while (left < right) {
        int currentArea = (right - left) * std::min(height[left], height[right]);
        maxArea = std::max(maxArea, currentArea);

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

    return maxArea;
}

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

Output



Q2 Convert Sorted Array to Binary Search Tree

Day 2 > Convert Sorted Array to Binary Search Tree.cpp

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

struct TreeNode {
    int val;
    TreeNode *left;
    TreeNode *right;
    TreeNode(int x) : val(x), left(NULL), right(NULL) {}
};

class Solution {
public:
    TreeNode* sortedArrayToBST(std::vector<int>& nums) {
        return sortedArrayToBST(nums, 0, nums.size() - 1);
    }

private:
    TreeNode* sortedArrayToBST(const std::vector<int>& nums, int left, int right) {
        if (left > right) {
            return NULL;
        }

        int mid = left + (right - left) / 2;
        TreeNode* node = new TreeNode(nums[mid]);
        node->left = sortedArrayToBST(nums, left, mid - 1);
        node->right = sortedArrayToBST(nums, mid + 1, right);

        return node;
    }
};
```

```cpp
};

void printInOrder(TreeNode* node) {
    if (node == NULL) {
        return;
    }
    printInOrder(node->left);
    std::cout << node->val << " ";
    printInOrder(node->right);
}

int main() {
    Solution solution;
    std::vector<int> nums = {-10, -3, 0, 5, 9};
    TreeNode* root = solution.sortedArrayToBST(nums);

    std::cout << "In-order traversal of the constructed BST: ";
    printInOrder(root);
    std::cout << std::endl;

    return 0;
}
```

Output

```
Output

In-order traversal of the constructed BST: -10 -3 0 5 9


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

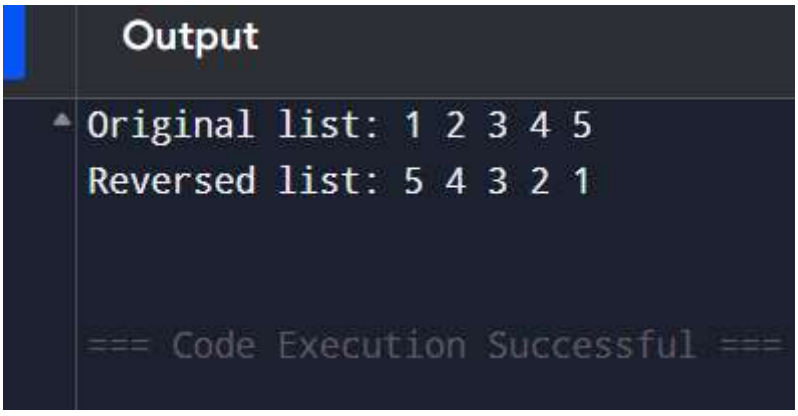
Q3 Reverse Linked List

Day 2 > Reverse Linked List.cpp

```cpp
#include <iostream>

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

ListNode* reverseList(ListNode* head) {
    ListNode* prev = nullptr;
    ListNode* curr = head;
    while (curr != nullptr) {
        ListNode* nextTemp = curr->next;
        curr->next = prev;
        prev = curr;
        curr = nextTemp;
    }
    return prev;
}

void printList(ListNode* head) {
    ListNode* temp = head;
    while (temp != nullptr) {
        std::cout << temp->val << " ";
        temp = temp->next;
    }
    std::cout << std::endl;
}

ListNode* createNode(int val) {
    return new ListNode(val);
}
```

```
33
34  int main() {
35      ListNode* head = createNode(1);
36      head->next = createNode(2);
37      head->next->next = createNode(3);
38      head->next->next->next = createNode(4);
39      head->next->next->next->next = createNode(5);
40
41      std::cout << "Original list: ";
42      printList(head);
43
44      vListNode* reversedHead = reverseList(head);
45
46      std::cout << "Reversed list: ";
47      printList(reversedHead);
48
49      return 0;
50  }
```

Output

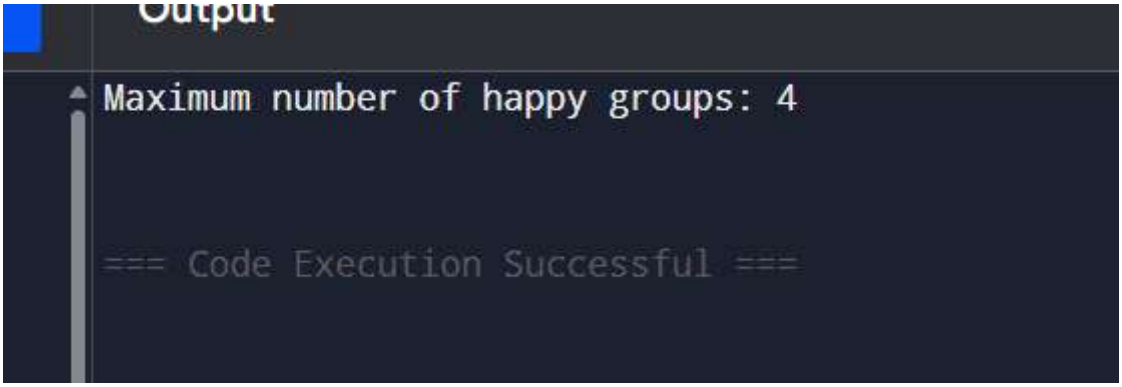


Q4 Maximum Number of Groups Getting Fresh Donuts

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

using namespace std;

class Solution {
public:
    int maxHappyGroups(int batchSize, vector<int>& groups) {
        unordered_map<int, int> remainderCount;
        for (int group : groups) {
            remainderCount[group % batchSize]++;
        }

        int happyGroups = remainderCount[0];
        remainderCount[0] = 0;

        for (int i = 1; i <= batchSize / 2; ++i) {
            if (i == batchSize - i) {
                happyGroups += remainderCount[i] / 2;
            } else {
                int minCount = min(remainderCount[i], remainderCount[batchSize -
                happyGroups += minCount;
                remainderCount[i] -= minCount;
                remainderCount[batchSize - i] -= minCount;
            }
        }

        vector<int> remainders;
        for (auto& [remainder, count] : remainderCount) {
            for (int i = 0; i < count; ++i) {
                remainders.push_back(remainder);
            }
        }
```

```
        sort(remainders.begin(), remainders.end(), greater<int>());

        int currentSum = 0;
        for (int remainder : remainders) {
            if (currentSum + remainder <= batchSize) {
                currentSum += remainder;
            } else {
                happyGroups++;
                currentSum = remainder;
            }
        }

        if (currentSum > 0) {
            happyGroups++;
        }

        return happyGroups;
    }
};

int main() {
    Solution solution;
    int batchSize = 3;
    vector<int> groups = {1, 2, 3, 4, 5, 6};
    cout << "Maximum number of happy groups: " << solution.maxHappyGroups(batchSize, groups) << endl;
    return 0;
}
```

Output



Q5 Jump Game II

Day 2 > Jump Game II.cpp

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

using namespace std;

int jump(vector<int>& nums) {
    int n = nums.size();
    if (n <= 1) return 0;

    int jumps = 0, current_end = 0, farthest = 0;

    for (int i = 0; i < n - 1; ++i) {
        farthest = max(farthest, i + nums[i]);
        if (i == current_end) {
            jumps++;
            current_end = farthest;
        }
    }

    return jumps;
}

int main() {
    vector<int> nums = {2, 3, 1, 1, 4};
    cout << "Minimum number of jumps: " << jump(nums) << endl;
    return 0;
}
```

Output