Winter Winning Camp

Name: -Priyanshu Raj

UID: - 22BCS15295

Section: - 620-A

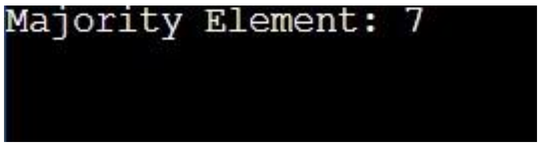
Date: -20/12/24

1) **Majority element**

```
#include <iostream> #include <vector>
using namespace std; int
majorityElement(vector<int>& nums)
{
int count = 0; int
candidate = 0; for
(int num : nums)
{
if (count == 0)
{
candidate = num;
}
count += (num == candidate) ? 1 : -1;
}
return candidate;
}
int main()
```

```
{
vector<int> nums = {3,7,3,7,3,1,7,6,7};
cout << "Majority Element: " << majorityElement(nums) << endl;
return 0;
}
```

OUTPUT: -

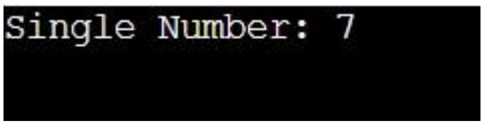


```
Majority Element: 7
```

2) Single Number

```
#include <iostream>

using namespace std;
int singleNumber(int nums[], int n)
{
int result = 0;
for (int i = 0; i < n; i++)
{
result ^= nums[i];
}
return result;
}
int main()
{
int nums[] = {7, 1, 2, 1, 2}; int n =
sizeof(nums) / sizeof(nums[0]);
cout << "Single Number: " << singleNumber(nums, n) << endl;
return 0;
}
```

OUTPUT: -



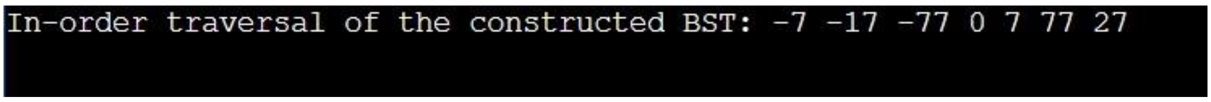
```
Single Number: 7
```

3) Convert Sorted Array to Binary Search Tree

```
#include <iostream>

#include <vector> using

namespace std; struct

TreeNode

{ int

val;

TreeNode* left;

TreeNode* right;

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

};

TreeNode* sortedArrayToBSTHelper(vector<int>& nums, int left, int
right)

{

if (left > right) return NULL; int

mid = left + (right - left) / 2;

TreeNode* root = new TreeNode(nums[mid]);


root->left = sortedArrayToBSTHelper(nums, left, mid - 1); root-

>right = sortedArrayToBSTHelper(nums, mid + 1, right); return

root;

}
```

```
TreeNode* sortedArrayToBST(vector<int>& nums)
{
return sortedArrayToBSTHelper(nums, 0, nums.size() - 1);
}
void printInOrder(TreeNode* root)
{
if (root == NULL) return;
printInOrder(root->left); cout
<< root->val << " ";
printInOrder(root->right);
}
int main()
{
vector<int> nums = {-7, -17,-77, 0, 7, 77,27};
TreeNode* root = sortedArrayToBST(nums); cout <<
"In-order traversal of the constructed BST: ";
printInOrder(root);  cout << endl; return 0;
}
```

OUTPUT: -



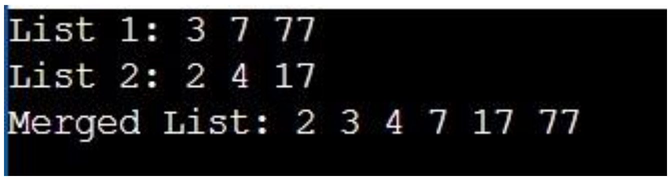
```
In-order traversal of the constructed BST: -7 -17 -77 0 7 77 27
```

4) Merge Two Sorted Lists

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

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

};
ListNode* mergeTwoLists(ListNode* list1, ListNode* list2) {
if (!list1) return list2; if (!list2) return list1; if (list1->val <
list2->val) {
list1->next = mergeTwoLists(list1->next, list2);
return list1; } else {
list2->next = mergeTwoLists(list1, list2->next); return
list2;
}
}
void printList(ListNode* head) {
while (head) { cout << head-
>val << " "; head = head->next;
}
cout << endl;
}
ListNode* createList(int arr[], int n) { if
(n == 0) return NULL;
ListNode* head = new ListNode(arr[0]); ListNode*
current = head;
for (int i = 1; i < n; ++i) { current->next
= new ListNode(arr[i]); current =
current->next;
}
return head;
}
int main() { int arr1[]
= {3, 7, 77}; int arr2[]
= {2, 4, 17};
ListNode* list1 = createList(arr1, 3);
ListNode* list2 = createList(arr2, 3);
cout << "List 1: "; printList(list1);
cout << "List 2: "; printList(list2);
ListNode* mergedList = mergeTwoLists(list1, list2); cout << "Merged
List: "; printList(mergedList); return 0;
}
```

OUTPUT: -



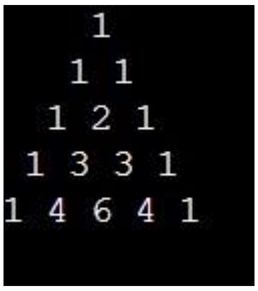
```
List 1: 3 7 77
List 2: 2 4 17
Merged List: 2 3 4 7 17 77
```

5) Pascal's Triangle

```
#include<iostream>
```

```
using namespace std;
int main()
{ int n = 5; for (int i =
0; i < n; i++)
{
int value = 1;

for (int j = 0; j < n - i - 1; j++)
{
cout << " ";
}
for (int j = 0; j <= i; j++)
{
cout << value << " "; value =
value * (i - j) / (j + 1);
}
cout << endl;
}
return 0;
}
```

OUTPUT: -



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

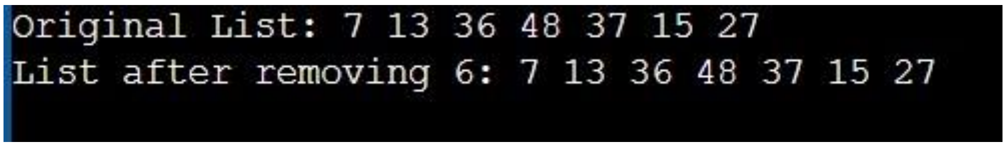
6) Remove Linked List Elements: -

```
#include <iostream> using
namespace std; struct
ListNode { int val;
ListNode* next;
ListNode(int x) : val(x), next(NULL) {}
};

ListNode* removeElements(ListNode* head, int val) { while
(head != NULL && head->val == val) {
ListNode* temp = head;
head = head->next; delete
temp;
}
ListNode* current = head;
while (current != NULL && current->next != NULL) {
if (current->next->val == val) { ListNode* temp =
current->next; current->next = current->next-
>next; delete temp;
} else { current =
current->next;
}
}
return head;
}
void printList(ListNode* head) {
while (head != NULL) { cout <<
head->val << " "; head = head-
>next;
}
cout << endl;
}
int main() {
ListNode* head = new ListNode(7);
head->next = new ListNode(13); head->next->next = new
ListNode(36); head->next->next->next = new ListNode(48);
head->next->next->next->next = new ListNode(37); head->next-
>next->next->next->next = new ListNode(15); head->next-
>next->next->next->next->next = new ListNode(27); cout <<
"Original List: "; printList(head); int val = 6;
head = removeElements(head, val); cout
<< "List after removing " << val << ": ";
printList(head);
```

return 0;

}

OUTPUT: -

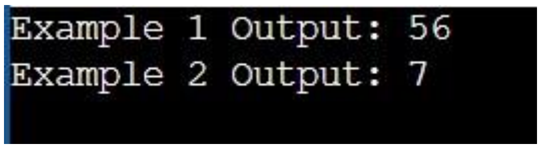


```
Original List: 7 13 36 48 37 15 27
List after removing 6: 7 13 36 48 37 15 27
```

7) Container With Most Water: -

```
#include <iostream> #include
<vector> using namespace std; int
maxArea(vector<int>& height) { int
left = 0, right = height.size() - 1; int
max_area = 0; while (left < right) {
int width = right - left;
int current_area = min(height[left], height[right]) * width;
max_area = max(max_area, current_area); if (height[left]
< height[right]) { left++; } else { right--;
}
}
return max_area;
}
int main() {
vector<int> height1 = {7, 3, 5, 6, 3, 9, 1, 13, 17}; cout <<
"Example 1 Output: " << maxArea(height1) << endl;
vector<int> height2 = {7, 8};
cout << "Example 2 Output: " << maxArea(height2) << endl; return
0;
}
```

OUTPUT: -

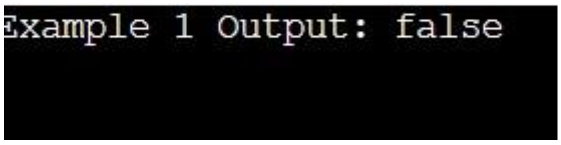


```
Example 1 Output: 56
Example 2 Output: 7
```

8) Valid Sudoku: -

```
#include <iostream>
#include <vector> #include
<unordered_set> using
namespace std;
bool isValidSudoku(vector<vector<char>>& board) {
vector<unordered_set<char>> rows(9), cols(9),
boxes(9); for (int i = 0; i < 9; i++) { for (int j = 0; j < 9; j++)
{ char num = board[i][j]; if (num == '.') continue; int
boxIndex = (i / 3) * 3 + j / 3;
if (rows[i].count(num) || cols[j].count(num) ||
boxes[boxIndex].count(num)) { return false;
}
rows[i].insert(num); cols[j].insert(num);
boxes[boxIndex].insert(num);
}
}
return true;

}

int main() {

vector<vector<char>> board1 = {

{'1', '3', '.', '.', '7', '.', '.', '.', '.'},

{'6', '.', '.', '1', '9', '5', '.', '.', '.'},

{'.', '9', '8', '.', '.', '.', '.', '6', '.'},

{'8', '.', '.', '.', '6', '.', '.', '.', '3'},

{'4', '.', '.', '8', '.', '3', '.', '.', '1'},


{'7', '.', '.', '.', '2', '.', '.', '.', '6'},

{'.', '6', '.', '.', '.', '.', '2', '8', '.'},

{'.', '.', '.', '4', '1', '9', '.', '.', '5'},

{'.', '.', '.', '.', '6', '.', '.', '7', '9'}

};

cout << "Example 1 Output: " << (isValidSudoku(board1) ? "true" :
"false") << endl; return

0;

}
```

OUTPUT: -



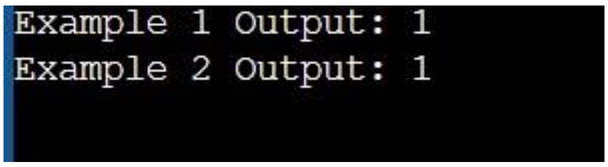
```
Example 1 Output: false
```

9) Jump Game II: - #include

<iostream> #include <vector>

```
using namespace std; int
jump(vector<int>& nums) {
int n = nums.size(); if
(n == 1) return 0;
int jumps = 0, currentEnd = 0, farthest = 0;
for (int i = 0; i < n - 1; i++) { farthest =
max(farthest, i + nums[i]); if (i ==
currentEnd) { jumps++; currentEnd =
farthest;
}
}
return jumps;
}
int main() {
vector<int> nums1 = {7, 8, 3, 6, 3};
cout << "Example 1 Output: " << jump(nums1) << endl;
vector<int> nums2 = {7, 4, 6, 0, 9};

cout << "Example 2 Output: " << jump(nums2) << endl; return
0;
}
```

OUTPUT: -



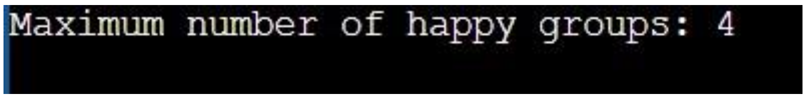
```
Example 1 Output: 1
Example 2 Output: 1
```

10) Maximum Number of Groups Getting Fresh Donuts: -

```
#include <iostream>

#include <vector> #include

<unordered_map> using

namespace std;

int maxHappyGroups(int batchSize, vector<int>& groups) {

unordered_map<int, int> remainderCount; for (int group :

groups) {

remainderCount[group % batchSize]++;

}

int happyGroups = remainderCount[0];

for (int i = 1; i <= batchSize / 2; i++) { if

(i == batchSize - i) {

happyGroups += remainderCount[i] / 2;

} else {

happyGroups += min(remainderCount[i], remainderCount[batchSize
- i]);


}

}

return happyGroups;

}
```

```
int main() { int
batchSize = 3;
vector<int>
groups = {7, 8, 9,
4, 9, 6}; cout <<
"Maximum
number of
happy groups: "
<<
maxHappyGrou
ps(batchSize,
groups) << endl;
return 0;
}
```

OUTPUT: -



```
Maximum number of happy groups: 4
```