Day-2

Q1- Majority Elements

```
#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;
   }

   count = 0;     for (int num
: nums) {         if (num ==
candidate) {
count++;
      }
   }

   if (count > nums.size() / 2)
      { return candidate;
   }

   return -1;
}

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

```
   vector<int> nums(n);      cout
<< "Enter the elements: ";      for
(int i = 0; i < n; ++i) {
       cin >> nums[i];
   }

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

Output-

```
Enter the number of elements: 3
Enter the elements: 2 2 1
Majority Element: 2
```

Q2- Single Number

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

   vector<int> nums(n);      cout
<< "Enter the elements: ";      for
(int i = 0; i < n; ++i) {
       cin >> nums[i];
   }

   cout << "Single Number: " << singleNumber(nums) << endl;
return 0;
}
```

Output-

```
Enter the number of elements: 3
Enter the elements: 2 1 1
Single Number: 2
```

Q3- 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(nullptr), right(nullptr) {}
};

TreeNode* sortedArrayToBST(vector<int>& nums)
{ if (nums.empty()) return nullptr;

   int mid = nums.size() / 2;
   TreeNode* root = new TreeNode(nums[mid]);

   vector<int> leftNums(nums.begin(), nums.begin() + mid);
vector<int> rightNums(nums.begin() + mid + 1, nums.end());

   root->left = sortedArrayToBST(leftNums);
   root->right = sortedArrayToBST(rightNums);

   return root;
}

void printInOrder(TreeNode* root)
{ if (!root) return;
printInOrder(root->left);       cout <<
root->val << " ";
printInOrder(root->right);
}
```

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

   vector<int> nums(n);
   cout << "Enter the sorted elements: ";
   for (int i = 0; i < n; ++i) {
cin >> nums[i];
   }

   TreeNode* root = sortedArrayToBST(nums);
cout << "In-order traversal of the BST: ";
printInOrder(root);      cout << endl;

   return 0;
}
```

Output-

```
Enter the number of elements: 6
Enter the sorted elements: 1 2 5 7 8 9
In-order traversal of the BST: 1 2 5 7 8 9
```

Q4- Merge Two Sorted Lists

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

struct ListNode
   { int val;
   ListNode* next;
   ListNode() : val(0), next(nullptr) {}
   ListNode(int x) : val(x), next(nullptr) {}
   ListNode(int x, ListNode* next) : val(x), next(next) {}
};

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;
}

int main()
{ int n1, n2;
    cout << "Enter the number of elements in the first list: ";
cin >> n1;
    ListNode* list1 = nullptr;
ListNode* tail1 = nullptr;
    cout << "Enter the sorted elements for the first list: ";
for (int i = 0; i < n1; ++i) {
        int val;
cin >> val;
        if (!list1) {              list1 =
new ListNode(val);
            tail1 = list1;
} else {
            tail1->next = new ListNode(val);
            tail1 = tail1->next;
        }
    }

    cout << "Enter the number of elements in the second list: ";
cin >> n2;
    ListNode* list2 = nullptr;
ListNode* tail2 = nullptr;
    cout << "Enter the sorted elements for the second list: ";
    for (int i = 0; i < n2; ++i) {
        int val;          cin >> val;
if (!list2) {               list2 = new
ListNode(val);
            tail2 = list2;
} else {
            tail2->next = new ListNode(val);
            tail2 = tail2->next;
        }
```

```
    }

    ListNode* mergedList = mergeTwoLists(list1, list2);
    cout << "Merged sorted list: ";
    printList(mergedList);

    return 0;
}
```

Output-

```
Enter the number of elements in the first list: 3
Enter the sorted elements for the first list: 1 2 4
Enter the number of elements in the second list: 3
Enter the sorted elements for the second list: 1 3 4
Merged sorted list: 1 1 2 3 4 4
```

Q5- Linked List Cycle

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

struct ListNode
    { int val;
    ListNode* next;
    ListNode() : val(0), next(nullptr) {}
    ListNode(int x) : val(x), next(nullptr) {}
    ListNode(int x, ListNode* next) : val(x), next(next) {}
};

bool hasCycle(ListNode* head) {
    if (!head || !head->next) return false;

    ListNode* slow = head;
    ListNode* fast = head->next;

    while (slow != fast) {
        if (!fast || !fast->next) return false;
        slow = slow->next;
        fast = fast->next->next;
    }

    return true;
}
```

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

   ListNode* head = nullptr;
ListNode* tail = nullptr;
   cout << "Enter the elements of the list: ";
   for (int i = 0; i < n; ++i) {
      int val;
cin >> val;
if (!head) {
         head = new ListNode(val);
         tail = head;
      } else {
         tail->next = new ListNode(val);
         tail = tail->next;
      }
   }

   if (hasCycle(head)) {
      cout << "The list has a cycle." << endl;
   } else {
      cout << "The list does not have a cycle." << endl;
   }

   return 0;
}
```

Output-

```
Enter the number of elements in the list: 2
Enter the elements of the list: 1 2
The list does not have a cycle.
```