Name: Rohan kumar

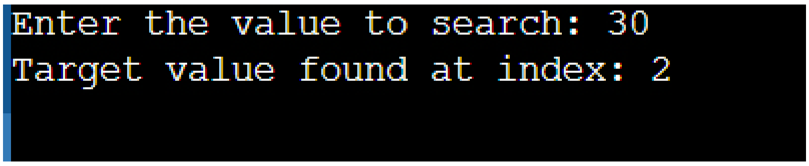
Uid: 22BCS15402

Sec: 620 B

Q1. Implementation of Linear Search

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

int linearSearch(int arr[], int size, int target) {
for (int i = 0; i < size; i++) {        if (arr[i] ==
target) {            return i;
        }
    }
    return -1;
}

int main() {     int arr[] = {10, 20, 30,
40, 50};     int size = sizeof(arr) /
sizeof(arr[0]);     int target = 30;

    int result = linearSearch(arr, size, target);
     if (result != -1) {        cout << "Target value found at index: "
<< result << endl;
    } else {
        cout << "Target value not found in the array." << endl;
    }

    return 0;
}
```

Output:

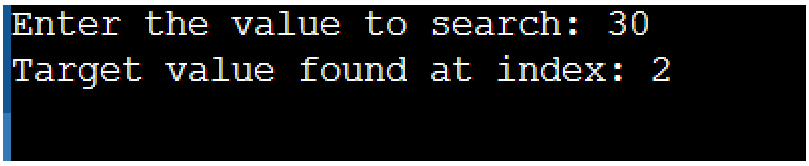


```
Enter the value to search: 30
Target value found at index: 2
```

Q2. Implementation of Binary Search to Find Index Value

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

int binarySearch(int arr[], int size, int target) {
int left = 0;     int right = size - 1;

    while (left <= right) {
        int mid = left + (right - left) / 2;

        if (arr[mid] == target) {
return mid;
        }

        if (arr[mid] < target) {
left = mid + 1;
        } else {
            right = mid - 1;
        }
    }

    return -1;
}

int main() {     int arr[] = {10, 20,
30, 40, 50};     int size =
sizeof(arr) / sizeof(arr[0]);     int
target = 30;

    int result = binarySearch(arr, size, target);
    if (result != -1) {        cout << "Target value found at index: "
<< result << endl;
   } else {
       cout << "Target value not found in the array." << endl;
   }

   return 0;
}
```

Output:

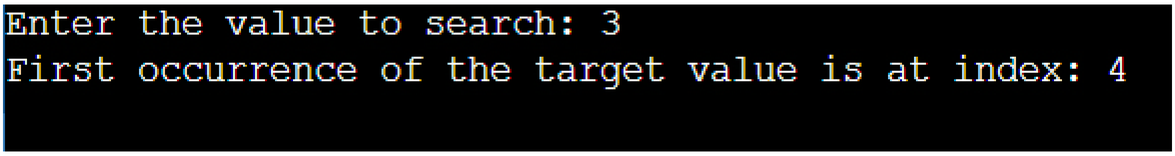


```
Enter the value to search: 30
Target value found at index: 2
```

Q3. Binary Search to Find First Occurrence of Target Value in Sorted Array

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

int binarySearchFirstOccurrence(int arr[], int size, int target) {
int left = 0;    int right = size - 1;    int result = -1;

   while (left <= right) {        int mid
= left + (right - left) / 2;        if
(arr[mid] == target) {            result
= mid;            right = mid - 1;
       } else if (arr[mid] < target) {
left = mid + 1;
       } else {
           right = mid - 1;
       }
   }

   return result;
}

int main() {     int arr[] = {10, 20, 20, 20,
30, 40, 50};     int size = sizeof(arr) /
sizeof(arr[0]);     int target = 20;

    int result = binarySearchFirstOccurrence(arr, size, target);
     if (result != -1) {         cout << "First occurrence of target value found
at index: " << result
<< endl;
    } else {
        cout << "Target value not found in the array." << endl;
    }

    return 0;
}
```

Output:



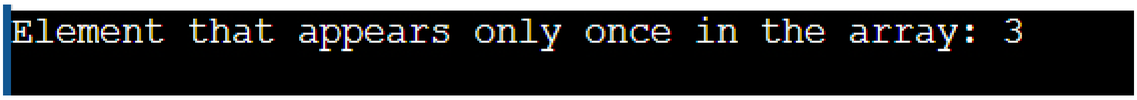
```
Enter the value to search: 3
First occurrence of the target value is at index: 4
```

Q4. Binary Search to Find Element that Appears Only Once in Sorted Array

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

int findUniqueElement(int arr[], int size) {
int left = 0;     int right = size - 1;

    while (left < right) {         int mid
= left + (right - left) / 2;
```

```
        if (mid % 2 == 0) {             if
(arr[mid] == arr[mid + 1]) {
left = mid + 2;
            } else {
right = mid;
            }
        } else {             if (arr[mid] ==
arr[mid - 1]) {                left = mid
+ 1;
            } else {
right = mid - 1;
            }
        }
    }

    return arr[left];
}

int main() {     int arr[] = {1, 1, 2, 2, 3,
4, 4, 5, 5};     int size = sizeof(arr) /
sizeof(arr[0]);

    int result = findUniqueElement(arr, size);

    cout << "Element that appears only once in the array: " << result <<
endl;

    return 0;
}
```

Output:

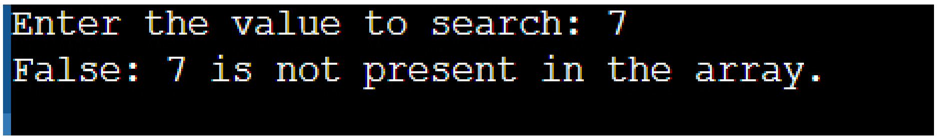


```
Element that appears only once in the array: 3
```

Q5. Given an Array Sorted in Ascending Order and an Integer k, Return True if k is Present in the Array Otherwise False

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

bool binarySearch(int arr[], int size, int target) {
int left = 0;     int right = size - 1;

    while (left <= right) {         int mid
= left + (right - left) / 2;

        if (arr[mid] == target) {
return true;
        }

        if (arr[mid] < target) {
left = mid + 1;
        } else {
            right = mid - 1;
        }
    }

    return false;
}

int main() {     int arr[] = {10, 20, 30,
40, 50};     int size = sizeof(arr) /
sizeof(arr[0]);     int target = 30;

    bool result = binarySearch(arr, size, target);
     if (result)
{
        cout << "Target value is present in the array." << endl;
    } else {          cout << "Target value is not present in the
array." << endl;
    }

    return 0;
}
```

Output:

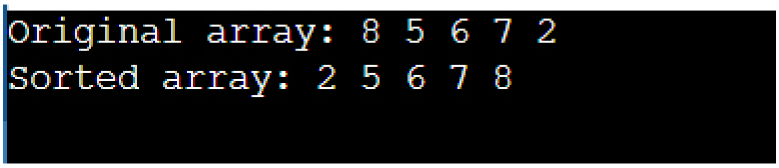


```
Enter the value to search: 7
False: 7 is not present in the array.
```

Q6. Bubble Sort

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

void bubbleSort(int arr[], int size) {
for (int i = 0; i < size - 1; i++) {
for (int j = 0; j < size - i - 1; j++) {
if (arr[j] > arr[j + 1]) {
swap(arr[j], arr[j + 1]);
            }
        }
    }
}

int main() {      int arr[] = {64, 34, 25, 12,
22, 11, 90};      int size = sizeof(arr) /
sizeof(arr[0]);

    bubbleSort(arr, size);
    cout << "Sorted array: ";
for (int i = 0; i < size; i++) {
cout << arr[i] << " ";
    }
    cout << endl;

    return 0;
}
```

Output:

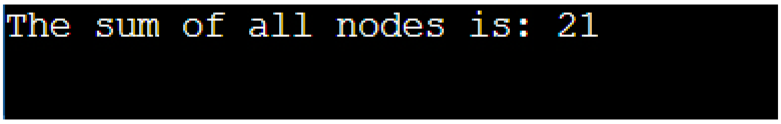


```
Original array: 8 5 6 7 2
Sorted array: 2 5 6 7 8
```

Q7. Sum of Binary Tree Nodes

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

int sumOfNodes(TreeNode* root) {
if (root == NULL) {         return 0;
    }
    return root->val + sumOfNodes(root->left) +
sumOfNodes(root>right);
}

int main() {
    TreeNode* root = new TreeNode(1);
root->left = new TreeNode(2);     root->right
= new TreeNode(3);     root->left->left =
new TreeNode(4);     root->right->left = new
TreeNode(6);

    int sum = sumOfNodes(root);     cout << "Sum of
all node values: " << sum << endl;

    return 0;
}
```

Output:

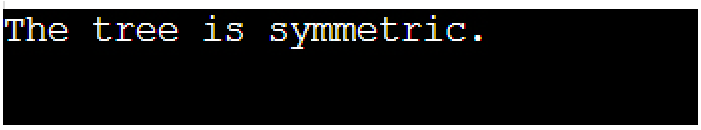


```
The sum of all nodes is: 21
```

Q8. Find if the Tree is Symmetric or Not

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

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

bool isMirror(TreeNode* left, TreeNode* right) {
if (left == NULL && right == NULL) {
return true;
    }
    if (left == NULL || right == NULL) {
return false;
    }
    return (left->val == right->val) && isMirror(left->left, right->right)
&& isMirror(left->right, right->left);
}

bool isSymmetric(TreeNode* root) {
if (root == NULL) {         return true;
    }
    return isMirror(root->left, root->right);
}

int main() {
    TreeNode* root = new TreeNode(1);
root->left = new TreeNode(2);     root->right
= new TreeNode(2);     root->left->left =
new TreeNode(3);     root->left->right = new
TreeNode(4);     root->right->left = new
TreeNode(4);     root->right->right = new
TreeNode(3);     bool result =
isSymmetric(root);
     if (result)
{
cout << "The tree is symmetric." << endl; } else { cout << "The tree is
not symmetric." << endl; } return 0;
}



        cout << "The tree is symmetric." << endl;

    } else {

        cout << "The tree is not symmetric." << endl;

    }


    return 0;

}
```

Output:



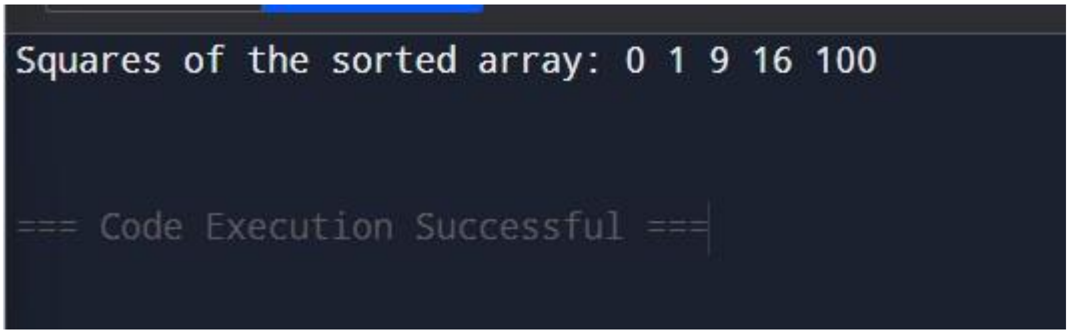
```
The tree is symmetric.
```

Q9. Squares of a Sorted Array

```
#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;


vector<int> sortedSquares(vector<int>& nums) {

for (int& num : nums) {        num = num * num;

   }

   sort(nums.begin(), nums.end());

return nums;

}


int main() {     vector<int> nums = {-4, -1, 0,

3, 10};     vector<int> result =

sortedSquares(nums);


   cout << "Squares of the sorted array: ";

for (int num : result) {        cout << num

<< " ";
```

```
    }
    cout << endl;


    return 0;
}
```

Output:



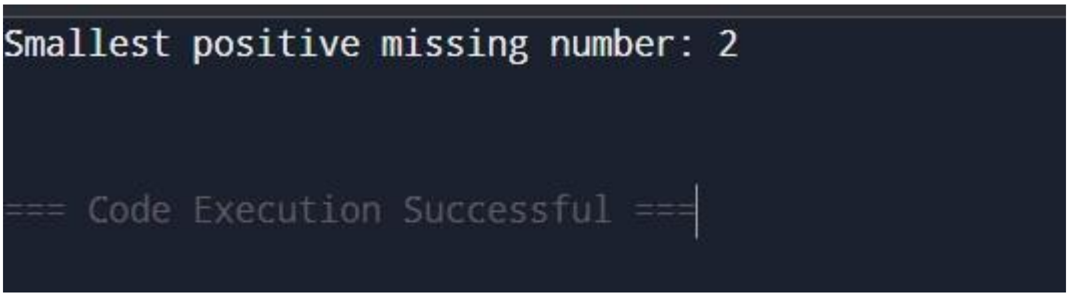
```
Squares of the sorted array: 0 1 9 16 100


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

Q10. Smallest Positive Missing Number

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

int smallestMissingPositive(vector<int>& nums) {
sort(nums.begin(), nums.end());     int smallest =
1;
```

```
    for (int num : nums) {
if (num == smallest) {
smallest++;
        }
    }

    return smallest;
}

int main() {     vector<int> nums = {3, 4, -1,
1};     int result =
smallestMissingPositive(nums);

    cout << "Smallest positive missing number: " << result << endl;

    return 0;
}
```

Output:



```
Smallest positive missing number: 2


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