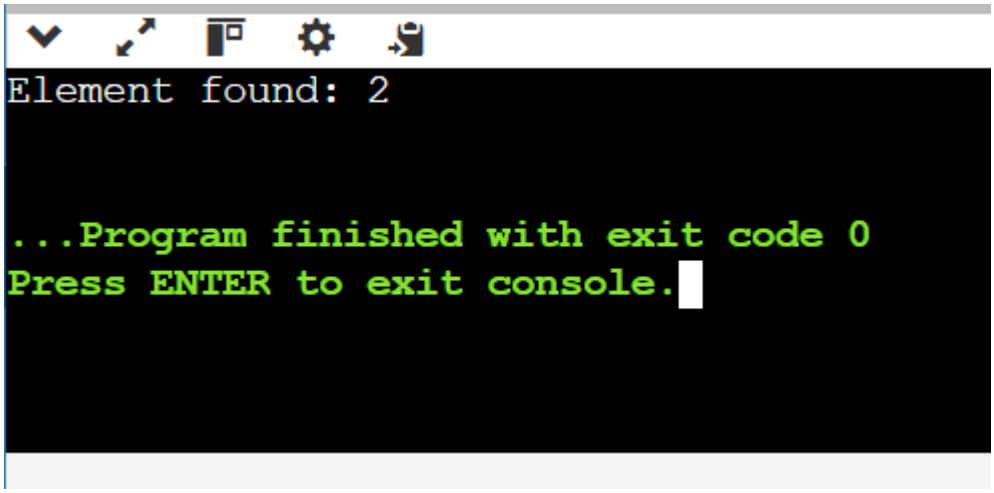
Name – Piyush **UID – 22BCS15351** **Section – 620-B**

Q1. Implementation of linear search

```
#include <iostream>
using namespace std;
int linearSearch(int arr[], int size, int t) {
    for (int i = 0; i < size; ++i) {
        if (arr[i] == t) {
            return i;
        }
    }
    return -1;
}
int main() {
    int arr[] = {1, 5, 14, 18, 25};
    int size = sizeof(arr) / sizeof(arr[0]);
    int t = 14;
    int result = linearSearch(arr, size, t);
    if (result != -1) {
        cout << "Element found: " << result << endl;
    } else {
        cout << "Element not found: " << endl;
    }
    return 0;
}
```

Output

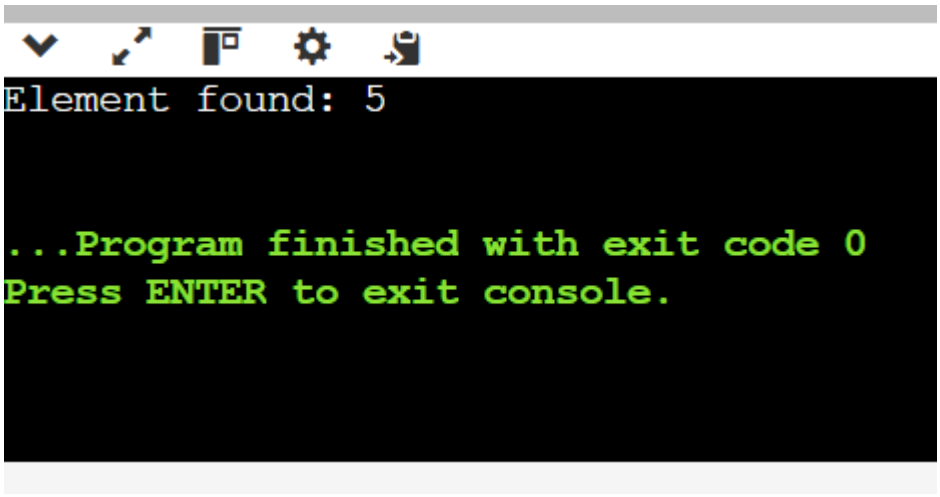


Q2. Implementation of binary search to find index value

```
#include <iostream>

using namespace std;

int binarySearch(int arr[], int size, int t) {
   int l = 0, r = size - 1;
   while (l <= r) {
      int mid = l + (r - l) / 2;
      if (arr[mid] == t) {
         return mid;
      } else if (arr[mid] < t) {
         l = mid + 1;
      } else {
         r = mid - 1;
      }
   }
   return -1;
}
int main() {
```

```
    int arr[] = {1, 4, 14, 25, 4, 18};
    int size = sizeof(arr) / sizeof(arr[0]);
    int t = 18 ;
    int result = binarySearch(arr, size, t);
    if (result != -1) {
        cout << "Element found: " << result << endl;
    } else {
        cout << "Element not found:" << endl;
    }
    return 0;
}
```

Output

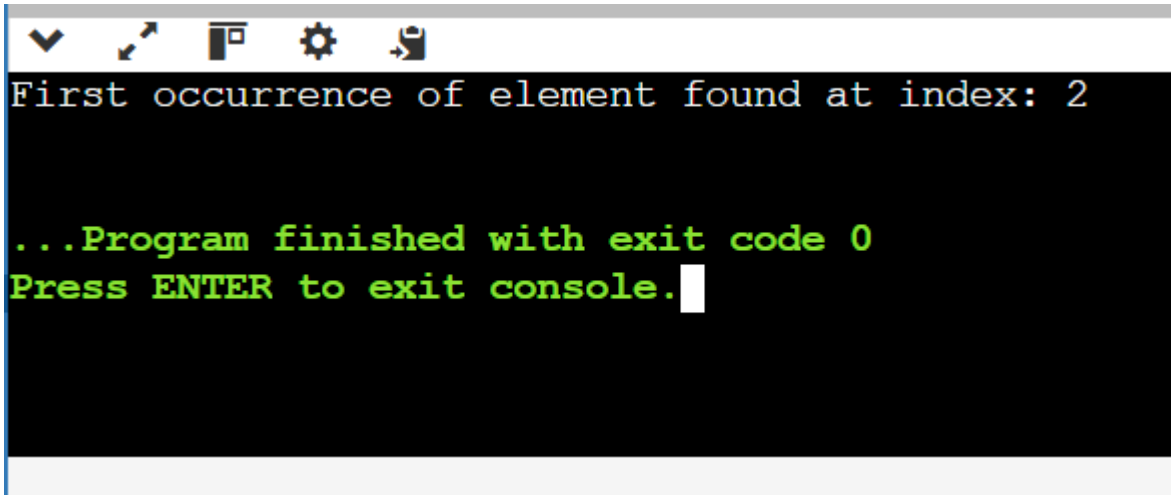


Q3. Binary search to find first occurance of target value in sorted array

```
#include <iostream>
using namespace std;
int firstOccurrenceBinarySearch(int arr[], int size, int t) {
    int l = 0, r = size - 1;
```

```
    int result = -1;
    while (l <= r) {
        int mid = l + (r - l) / 2;
        if (arr[mid] == t) {
            result = mid;
            r = mid - 1;
        } else if (arr[mid] < t) {
            l = mid + 1;
        } else {
            r = mid - 1;
        }
    }

    return result;
}
int main() {
    int arr[] = {5, 10, 15, 20, 25, 30};
    int size = sizeof(arr) / sizeof(arr[0]);
    int t = 12;
    int result = firstOccurrenceBinarySearch(arr, size, t);
    if (result != -1) {
        cout << "First occurrence of element found at index: " << result << endl; // Output: 2
    } else {
        cout << "Element not found:" << endl;
    }
    return 0;
}
```

Output

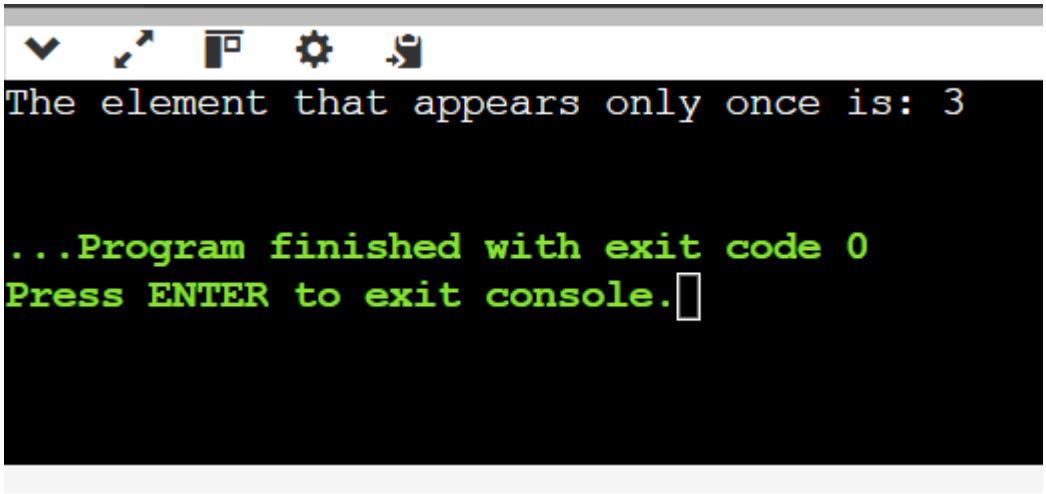


Q4. appears only once in sorted array (bs)

```
#include <iostream>

using namespace std;

int singleNonDuplicate(int arr[], int size) {
    int left = 0, right = size - 1;
    while (left < right) {
        int mid = left + (right - left) / 2;
        if (mid % 2 == 1) {
            mid--;
        }
        if (arr[mid] == arr[mid + 1]) {
            left = mid + 2;
        } else {
            right = mid;
        }
    }
    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 = singleNonDuplicate(arr, size);
    cout << "The element that appears only once is: " << result << endl;
    return 0;
}
```

Output

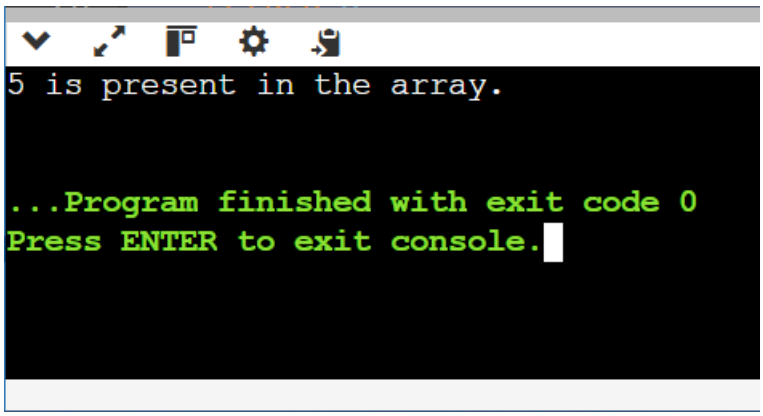


Q5. given an array sorted in ascending order and an integer k return true if k is present in the array otherwise false

```
#include <iostream>
using namespace std;
int binarySearch(int arr[], int size, int k) {
    int left = 0, right = size - 1;
    while (left <= right) {
        int mid = left + (right - left) / 2;
        if (arr[mid] == k) {
```

```
            return true;
        } else if (arr[mid] < k) {
            left = mid + 1;
        } else {
            right = mid - 1;
        }
    }
    return false;
}
int main() {
    int arr[] = {1, 3, 5, 7, 9, 11, 13};
    int size = sizeof(arr) / sizeof(arr[0]);
    int k = 5;
    if (binarySearch(arr, size, k)) {
        cout << k << " is present in the array." << endl;
    } else {
        cout << k << " is not present in the array." << endl;
    }
    return 0;
}
```

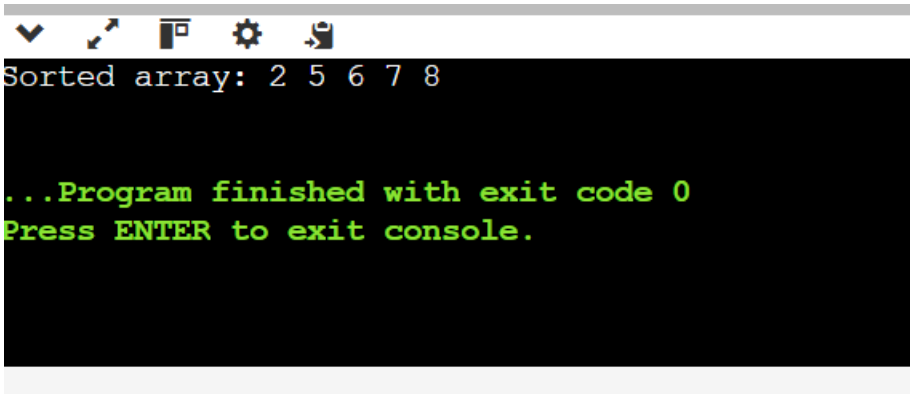
Output



Q6. Bubble sort

```
#include <iostream>
using namespace std;
void bubbleSort(int arr[], int n) {
   for (int i = 0; i < n - 1; ++i) {
      for (int j = 0; j < n - i - 1; ++j) {
         if (arr[j] > arr[j + 1]) {
            int temp = arr[j];
            arr[j] = arr[j + 1];
            arr[j + 1] = temp;
         }
      }
   }
}
int main() {
   int arr[] = {8, 5, 7, 6, 2};
   int n = sizeof(arr) / sizeof(arr[0]);
   bubbleSort(arr, n);
   cout << "Sorted array: ";
   for (int i = 0; i < n; ++i) {
      cout << arr[i] << " ";
   }
   cout << endl;
   return 0;
}
```

Output

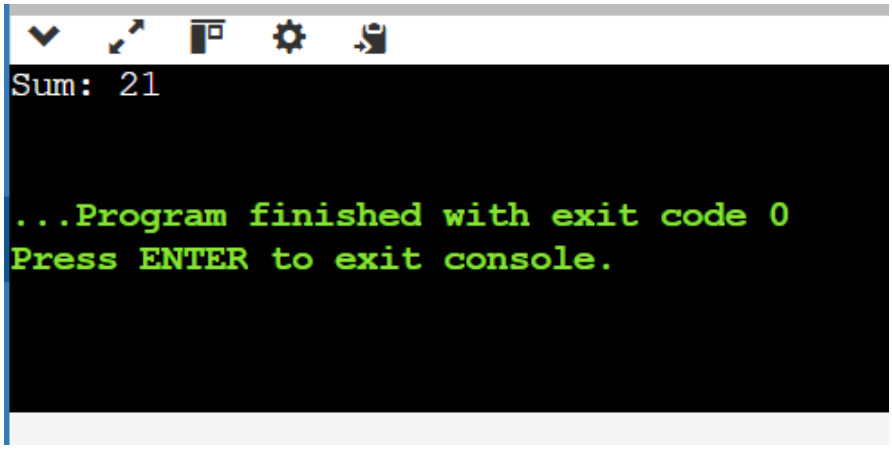


Q7. Sum of binary tree nodes

```
#include <iostream>

using namespace std;

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

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

TreeNode* createExampleTree() {
    TreeNode* root = new TreeNode(1);
    root->left = new TreeNode(2);
    root->right = new TreeNode(3);
    root->left->left = new TreeNode(4);
```

```
    root->left->right = new TreeNode(5);
    root->right->right = new TreeNode(6);
    return root;
}
int main() {
    TreeNode* root = createExampleTree();
    int sum = sumOfNodes(root);
    cout << "Sum: " << sum << endl;
    return 0;
}
```

Output



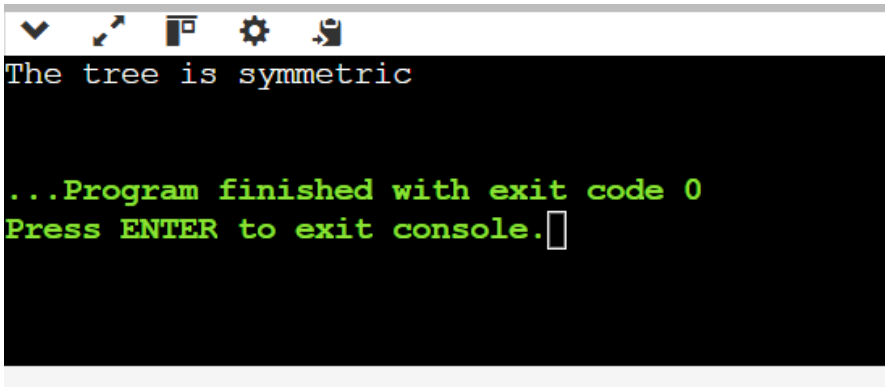
Q8. Find the tree is symmetric or not .Input – [1,2,2,3,4,4,3]

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

```
};
int isMirror(TreeNode* left, TreeNode* right) {
    if (left == nullptr && right == nullptr) {
        return true;
    }
    if (left == nullptr || right == nullptr) {
        return false;
    }
    return (left->val == right->val) && isMirror(left->left, right->right) && isMirror(left->right,
right->left);
}
int isSymmetric(TreeNode* root) {
    if (root == nullptr) {
        return true;
    }
    return isMirror(root->left, root->right);
}
TreeNode* createExampleTree() {
    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);
    return root;
}
```

```
int main() {
    TreeNode* root = createExampleTree();
    bool symmetric = isSymmetric(root);
    cout << "The tree is " << (symmetric ? "symmetric" : "not symmetric") << endl;
    return 0;
}
```

Output

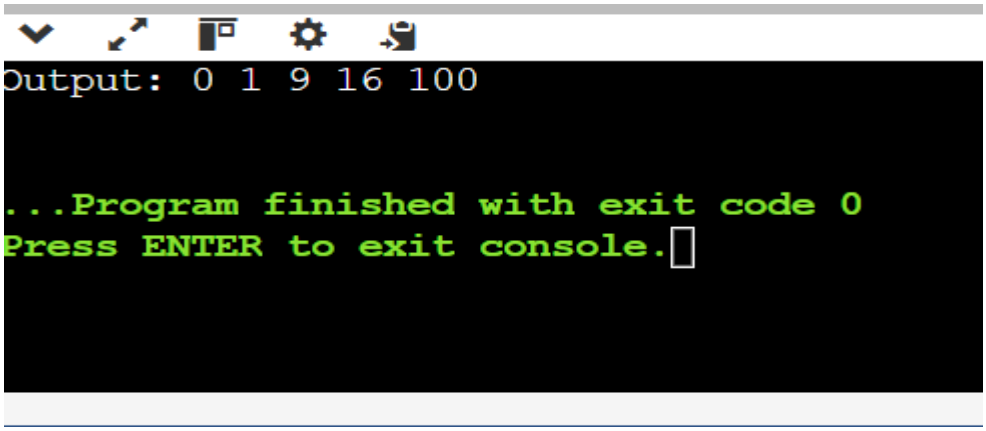


Q9. **Squares of a Sorted Array**

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
vector<int> sortedSquares(vector<int>& nums) {
    int n = nums.size();
    vector<int> result(n);
    int left = 0, right = n - 1;
    int pos = n - 1;
    while (left <= right) {
        if (abs(nums[left]) > abs(nums[right])) {
            result[pos] = nums[left] * nums[left];
```

```
            left++;
        } else {
            result[pos] = nums[right] * nums[right];
            right--;
        }
        pos--;
    }
    return result;
}
int main() {
    vector<int> nums = {-4, -1, 0, 3, 10};
    vector<int> result = sortedSquares(nums);

    cout << "Output: ";
    for (int x : result) {
        cout << x << " ";
    }
    cout << endl;
    return 0;
}
```

Output



Q10. Smallest positive missing number.

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
int smallestMissingPositive(vector<int>& arr) {
    int n = arr.size();
    int j = 0;
    for (int i = 0; i < n; i++) {
        if (arr[i] <= 0) {
            swap(arr[i], arr[j]);
            j++;
        }
    }
    for (int i = j; i < n; i++) {
        int val = abs(arr[i]);
        if (val - 1 + j < n && arr[val - 1 + j] > 0) {
            arr[val - 1 + j] = -arr[val - 1 + j];
        }
    }
    for (int i = j; i < n; i++) {
        if (arr[i] > 0) {
            return i - j + 1;
        }
    }
    return n - j + 1;
}
```

```
int main() {
    vector<int> arr = {3, 4, -1, 1};

    int result = smallestMissingPositive(arr);
    cout << "The smallest positive missing number is: " << result << endl;

    return 0;
}
```

Output