**Assessment**

Thank you for showing interest in the role of Application Developer. We're looking forward to moving you on to the application process, the Online Assessment which will be followed by an interview.

1. Below mentioned are 10 programming/coding questions and we would like to see your response under each respective question with an explanation/example.
2. You will be given 24 hours from the time you received this document to complete this assessment and send it over at [anilay@turing systems.com](mailto:anilay@turingsystems.com).
3. We will review your document on assessment and get back to you with an update.
4. Should there be any concerns or doubts before taking the test, please feel free to write to us at [anilay@turing systems.com](mailto:anilay@turingsystems.com) to set up a quick call.

**Questions**

1. How do you find the missing number in a given integer array of 1 to 100?
2. How are duplicates removed from a given array in Java?
3. How do you find the middle element of a singly linked list in one pass?
4. How do you find the length of a singly linked list?
5. How do you check if a string contains only digits?
6. How do you reverse words in a given sentence without using any library method?
7. How is a binary search tree implemented?
8. How are all leaves of a binary search tree printed?
9. How is a bubble sort algorithm implemented?
10. How is a merge sort algorithm implemented?

**Solutions**

1. **Algorithm for the missing number in a given integer array:**
   1. Calculate the sum of first n natural numbers as sumtotal= n\*(n+1)/2
   2. Create a variable sum to store the sum of array elements.
   3. Traverse the array from start to end.
   4. Update the value of sum as sum = sum + array[i]
   5. Print the missing number as sumtotal – sum

**Code:**

#include <stdio.h>

/\* getMissingNo takes array and size of array as arguments\*/

int getMissingNo(int a[], int n)

{

int i, total;

total = (n + 1) \* (n + 2) / 2;

for (i = 0; i < n; i++)

total -= a[i];

return total;

}

/\*program to test above function \*/

int main()

{

int a[] = { 1, 2, 4, 5, 6 };

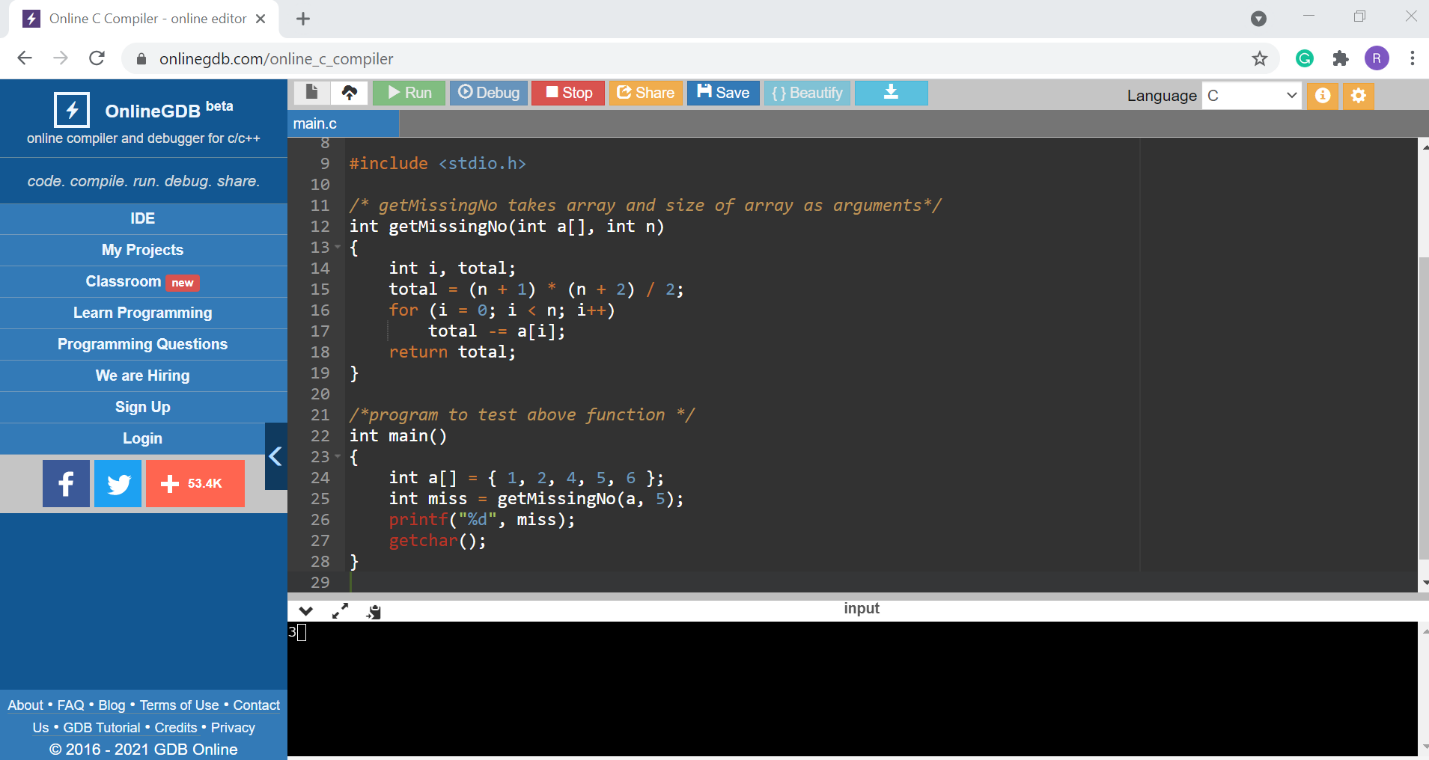
int miss = getMissingNo(a, 5);

printf("%d", miss);

getchar();

}

**Output:**



# 2. **Algorithm to remove duplicate element in an Array**

Using temporary array or using separate index. To remove the duplicate element from array, the array must be in sorted order. If array is not sorted, you can sort it by calling Arrays.sort(arr) method.

**Code:**

public class RemoveDuplicateInArrayExample{

public static int removeDuplicateElements(int arr[], int n){

        if (n==0 || n==1){

            return n;

        }

        int[] temp = new int[n];

        int j = 0;

        for (int i=0; i<n-1; i++){

            if (arr[i] != arr[i+1]){

                temp[j++] = arr[i];

            }

         }

        temp[j++] = arr[n-1];

        // Changing original array

        for (int i=0; i<j; i++){

            arr[i] = temp[i];

        }

        return j;

    }

    public static void main (String[] args) {

        int arr[] = {10,20,20,30,30,40,50,50};

        int length = arr.length;

        length = removeDuplicateElements(arr, length);

        //printing array elements

        for (int i=0; i<length; i++)

           System.out.print(arr[i]+" ");

    }

}

**Output:**



3.**Algorithm to find the middle element of a singly linked list:**

Traverse linked list using two pointers. Move one pointer by one and the other pointers by two. When the fast pointer reaches the end slow pointer will reach the middle of the linked list.

**Code:**

// C program to find middle of linked list

#include<stdio.h>

#include<stdlib.h>

/\* Link list node \*/

struct Node

{

int data;

struct Node\* next;

};

/\* Function to get the middle of the linked list\*/

void printMiddle(struct Node \*head)

{

struct Node \*slow\_ptr = head;

struct Node \*fast\_ptr = head;

if (head!=NULL)

{

while (fast\_ptr != NULL && fast\_ptr->next != NULL)

{

fast\_ptr = fast\_ptr->next->next;

slow\_ptr = slow\_ptr->next;

}

printf("The middle element is [%d]\n\n", slow\_ptr->data);

}

}

void push(struct Node\*\* head\_ref, int new\_data)

{

/\* allocate node \*/

struct Node\* new\_node =

(struct Node\*) malloc(sizeof(struct Node));

/\* put in the data \*/

new\_node->data = new\_data;

/\* link the old list off the new node \*/

new\_node->next = (\*head\_ref);

/\* move the head to point to the new node \*/

(\*head\_ref) = new\_node;

}

// A utility function to print a given linked list

void printList(struct Node \*ptr)

{

while (ptr != NULL)

{

printf("%d->", ptr->data);

ptr = ptr->next;

}

printf("NULL\n");

}

/\* Driver program to test above function\*/

int main()

{

/\* Start with the empty list \*/

struct Node\* head = NULL;

int i;

for (i=5; i>0; i--)

{

push(&head, i);

printList(head);

printMiddle(head);

}

return 0;

}

**Output:**



## 4. **Length of Linked List using Iterative Approach**

## We will use the Linked list traversal to find the length of a linked list.

* Head Points to the First Node of The List.
* Initialize the count variable with value 0
* Initialize the temp variable with Head
* As we access each Node, the value of count variable is increased by 1.
* Stop The process when we reach null.
* Do not change the head reference.

**Code:**

public class MyLinkedList {

public class Node {

int data;

Node next;

}

public Node head;

public Node tail;

public int size;

public int getFirst() throws Exception {

if (this.size == 0) {

throw new Exception("linked list is empty");

}

return this.head.data;

}

public int getLast() throws Exception {

if (this.size == 0) {

throw new Exception("linked list is empty");

}

return this.tail.data;

}

public void display() {

Node temp = this.head;

while (temp != null) {

System.out.println(temp.data + " ");

temp = temp.next;

}

}

public void addFirst(int item) {

Node nn = new Node();

nn.data = item;

if (this.size == 0) {

this.head = nn;

this.tail = nn;

this.size = this.size + 1;

} else {

nn.next = this.head;

this.head = nn;

this.size = this.size + 1;

}

}

public int length() {

Node temp = this.head;

int count = 0;

while (temp != null) {

count++;

temp = temp.next;

}

return count;

}

public static void main(String[] args) {

MyLinkedList ll = new MyLinkedList();

ll.addFirst(10);

ll.addFirst(20);

ll.addFirst(30);

ll.addFirst(40);

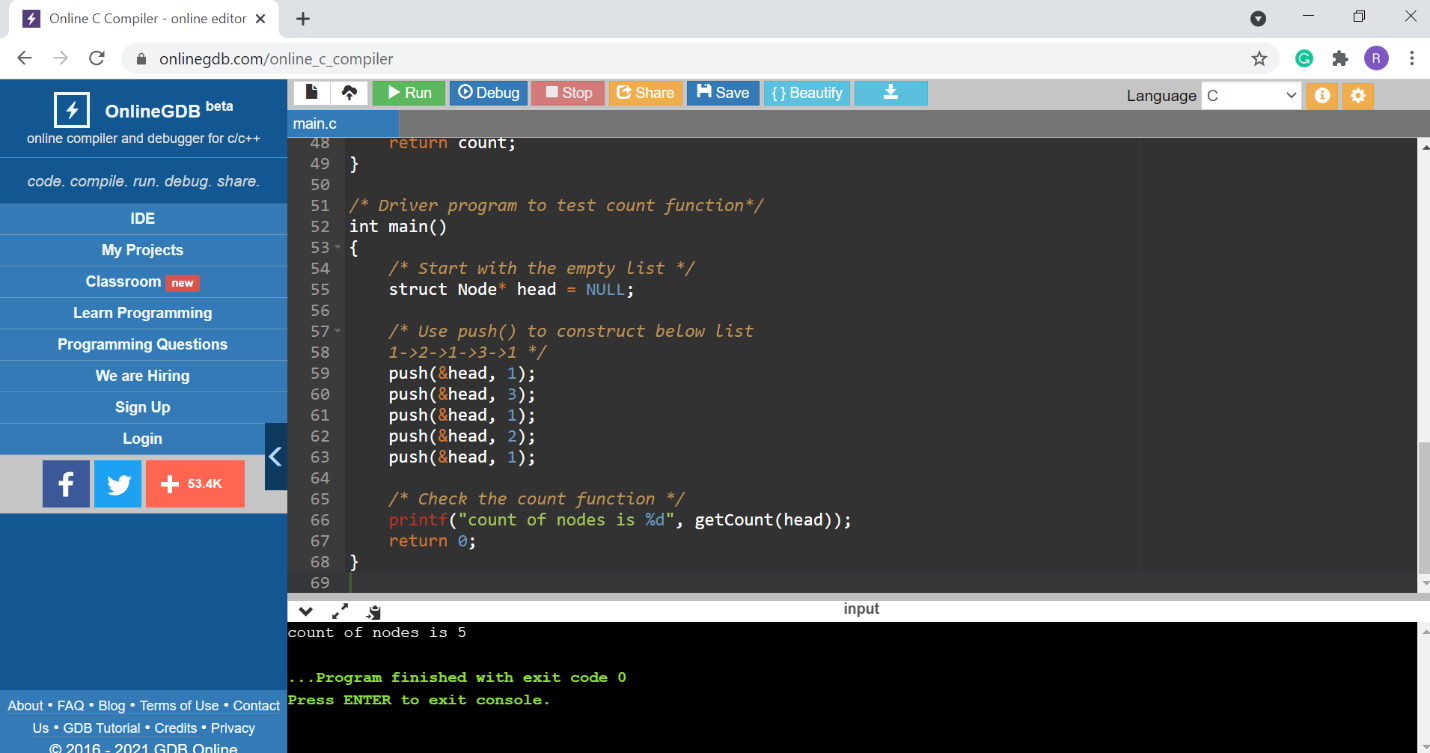
ll.addFirst(50);

System.out.println("Length of Linked List is " + ll.length());

}

}

**Output:**



5. **Algorithm to check if a string contains only digits Using Traversal:**

The idea is to traverse each character in the string and check if the character of the string contains only digits from **0 to 9**. If all the character of the string contains only digits then return true, otherwise, return false.

Below is the implementation of the above approach:

**Code:**

// Java program for the above approach

// contains only digits

class GFG {

// Function to check if a string

// contains only digits

public static boolean

onlyDigits(String str, int n)

{

// Traverse the string from

// start to end

for (int i = 0; i < n; i++) {

// Check if character is

// digit from 0-9

// then return true

// else false

if (str.charAt(i) >= '0'

&& str.charAt(i) <= '9') {

return true;

}

else {

return false;

}

}

return false;

}

// Driver Code

public static void main(String args[])

{

// Given string str

String str = "1234";

int len = str.length();

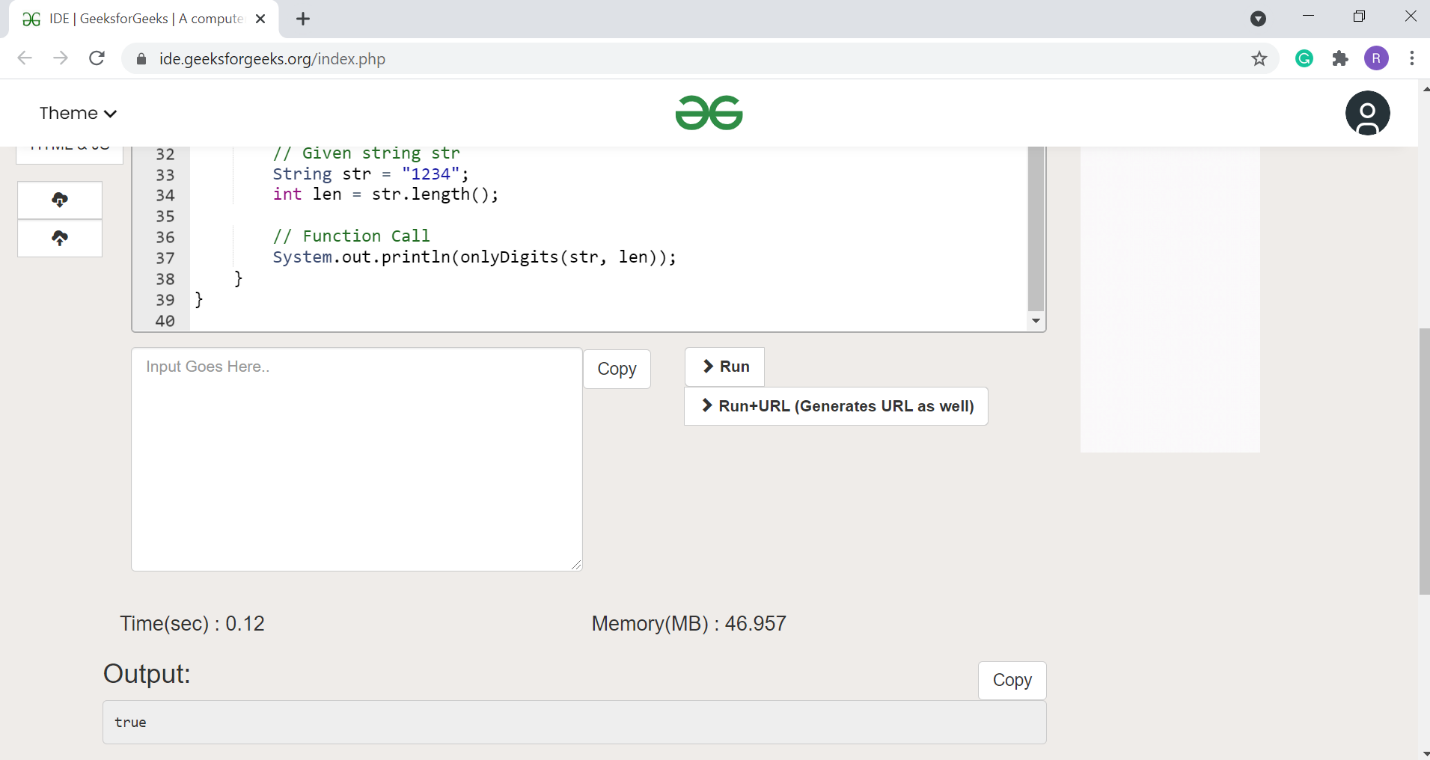
// Function Call

System.out.println(onlyDigits(str, len));

}

}

Output:



## 6. **Algorithm to reverse words in a given sentence**

## Initially, reverse the individual words of the given string one by one

* [Reverse the whole string](https://www.geeksforgeeks.org/write-a-program-to-reverse-an-array-or-string/) from start to end to get the desired output

**Code:**

// Java program to

// reverse a String

import java.util.\*;

class GFG{

// Reverse the letters

// of the word

static void reverse(char str[],int start, int end)

{

// Temporary variable

// to store character

char temp;

while (start <= end)

{

// Swapping the first

// and last character

temp = str[start];

str[start] = str[end];

str[end] = temp;

start++;

end--;

}

}

// Function to reverse words

static char[] reverseWords(char []s)

{

// Reversing individual words as

// explained in the first step

int start = 0;

for (int end = 0; end < s.length; end++)

{

// If we see a space, we

// reverse the previous

// word (word between

// the indexes start and end-1

// i.e., s[start..end-1]

if (s[end] == ' ')

{

reverse(s, start, end);

start = end + 1;

}

}

// Reverse the last word

reverse(s, start, s.length - 1);

// Reverse the entire String

reverse(s, 0, s.length - 1);

return s;

}

// Driver Code

public static void main(String[] args)

{

String s = "i like this program very much ";

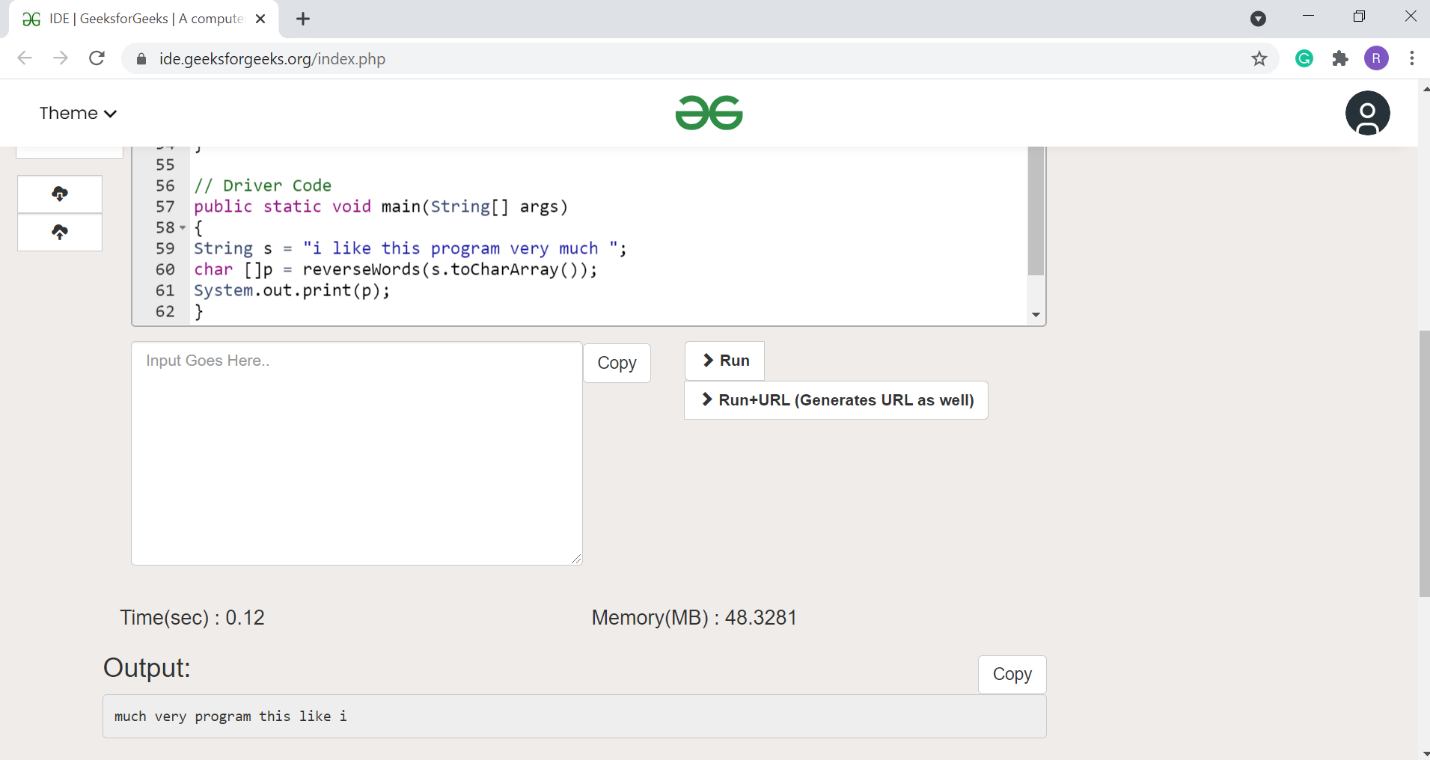
char []p = reverseWords(s.toCharArray());

System.out.print(p);

}

}

**Output**:



7.**Algorithm to implement Binary Search Tree**

* Binary Search tree can be defined as a class of binary trees, in which the nodes are arranged in a specific order. This is also called ordered binary tree.
* In a binary search tree, the value of all the nodes in the left sub-tree is less than the value of the root.
* Similarly, value of all the nodes in the right sub-tree is greater than or equal to the value of the root.
* This rule will be recursively applied to all the left and right sub-trees of the root.

**Code: Program to implement BST operations**

#include <iostream>

#include <stdlib.h>

using namespace std;

struct Node {

    int data;

    Node \*left;

    Node \*right;

};

Node\* create(int item)

{

    Node\* node = new Node;

    node->data = item;

    node->left = node->right = NULL;

    return node;

}

void inorder(Node \*root)

{

    if (root == NULL)

        return;

    inorder(root->left);

    cout<< root->data << "   ";

    inorder(root->right);

}

Node\* findMinimum(Node\* cur)

{

    while(cur->left != NULL) {

        cur = cur->left;

    }

    return cur;

}

Node\* insertion(Node\* root, int item)

{

    if (root == NULL)

        return create(item);

    if (item < root->data)

        root->left = insertion(root->left, item);

    else

        root->right = insertion(root->right, item);

    return root;

}

void search(Node\* &cur, int item, Node\* &parent)

{

    while (cur != NULL && cur->data != item)

    {

        parent = cur;

        if (item < cur->data)

            cur = cur->left;

        else

            cur = cur->right;

    }

}

void deletion(Node\*& root, int item)

{

    Node\* parent = NULL;

    Node\* cur = root;

    search(cur, item, parent);

    if (cur == NULL)

        return;

    if (cur->left == NULL && cur->right == NULL)

    {

        if (cur != root)

        {

            if (parent->left == cur)

                parent->left = NULL;

            else

                parent->right = NULL;

        }

        else

            root = NULL;

        free(cur);

    }

    else if (cur->left && cur->right)

    {

        Node\* succ  = findMinimum(cur- >right);

        int val = succ->data;

        deletion(root, succ->data);

        cur->data = val;

    }

    else

    {

        Node\* child = (cur->left)? Cur- >left: cur->right;

        if (cur != root)

        {

            if (cur == parent->left)

                parent->left = child;

            else

                parent->right = child;

        }

        else

            root = child;

        free(cur);

    }

}

int main()

{

   Node\* root = NULL;

   int keys[8];

   for(int i=0;i<8;i++)

    {

    cout << "Enter value to be inserted";

    cin>>keys[i];

        root = insertion(root, keys[i]);

    }

    inorder(root);

    cout<<"\n";

    deletion(root, 10);

    inorder(root);

    return 0;

}

8. **Algorithm to print all leaves of a binary search tree:**

* Check if the given node is null. If null, then return from the function.
* Check if it is a leaf node. If the node is a leaf node, then print its data.
* If in the above step, the node is not a leaf node then check if the left and right children of node exist. If yes then call the function for left and right child of the node recursively.

**Code:**

// Java program to print leaf nodes

// from left to right

import java.util.\*;

class GFG{

// A Binary Tree Node

static class Node

{

public int data;

public Node left, right;

};

// Function to print leaf

// nodes from left to right

static void printLeafNodes(Node root)

{

// If node is null, return

if (root == null)

return;

// If node is leaf node, print its data

if (root.left == null &&

root.right == null)

{

System.out.print(root.data + " ");

return;

}

// If left child exists, check for leaf

// recursively

if (root.left != null)

printLeafNodes(root.left);

// If right child exists, check for leaf

// recursively

if (root.right != null)

printLeafNodes(root.right);

}

// Utility function to create a new tree node

static Node newNode(int data)

{

Node temp = new Node();

temp.data = data;

temp.left = null;

temp.right = null;

return temp;

}

// Driver code

public static void main(String []args)

{

// Let us create binary tree shown in

// above diagram

Node root = newNode(1);

root.left = newNode(2);

root.right = newNode(3);

root.left.left = newNode(4);

root.right.left = newNode(5);

root.right.right = newNode(8);

root.right.left.left = newNode(6);

root.right.left.right = newNode(7);

root.right.right.left = newNode(9);

root.right.right.right = newNode(10);

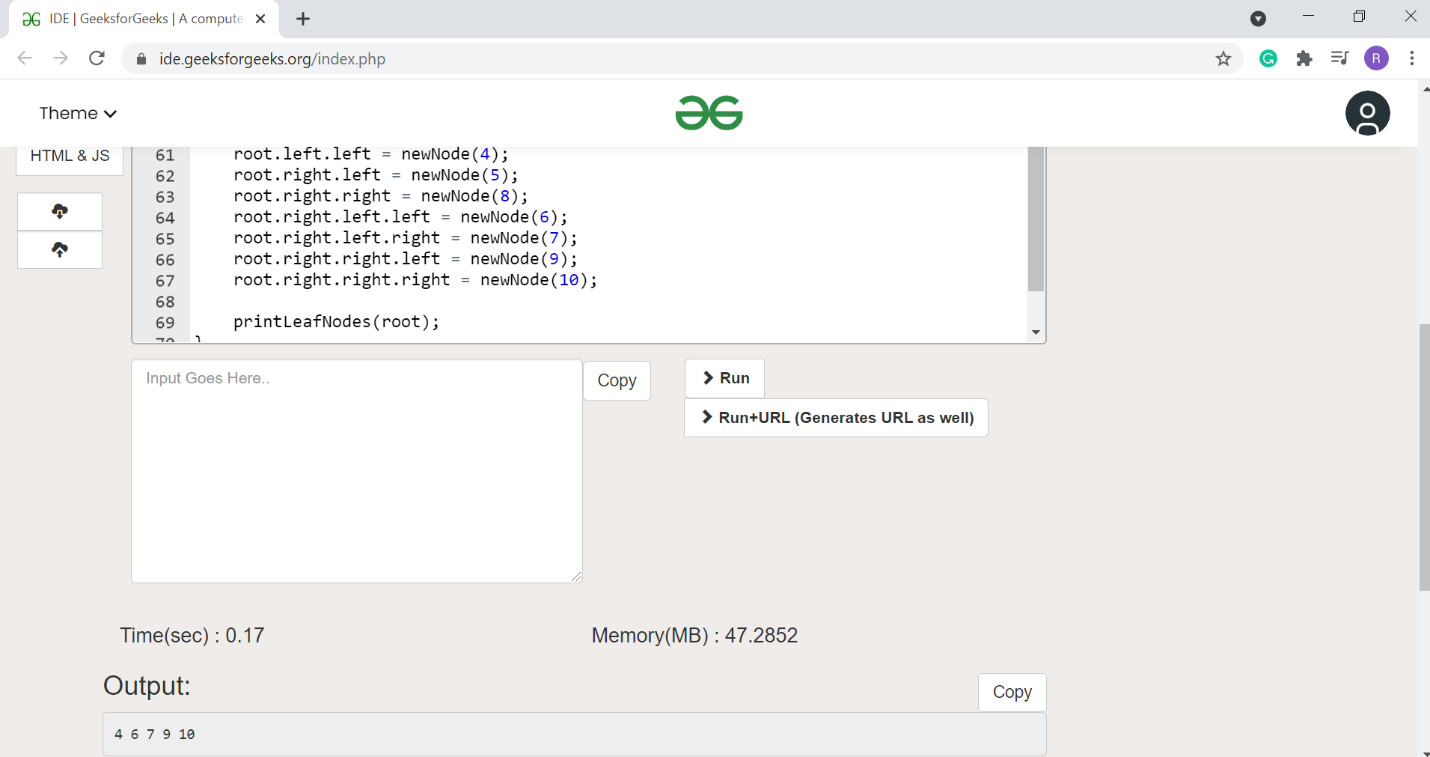
// Print leaf nodes of the given tree

printLeafNodes(root);

}

}

**Output:**



9.**Implementing Bubble Sort Algorithm**

Following are the steps involved in bubble sort(for sorting a given array in ascending order):

* Starting with the first element(index = 0), compare the current element with the next element of the array.
* If the current element is greater than the next element of the array, swap them.
* If the current element is less than the next element, move to the next element. **Repeat Step 1**.

**Code:**

// below we have a simple C program for bubble sort

#include <stdio.h>

void bubbleSort(int arr[], int n)

{

int i, j, temp, flag=0;

for(i = 0; i < n; i++)

{

for(j = 0; j < n-i-1; j++)

{

// introducing a flag to monitor swapping

if( arr[j] > arr[j+1])

{

// swap the elements

temp = arr[j];

arr[j] = arr[j+1];

arr[j+1] = temp;

// if swapping happens update flag to 1

flag = 1;

}

}

// if value of flag is zero after all the iterations of inner loop

// then break out

if(flag==0)

{

break;

}

}

// print the sorted array

printf("Sorted Array: ");

for(i = 0; i < n; i++)

{

printf("%d ", arr[i]);

}

}

int main()

{

int arr[100], i, n, step, temp;

// ask user for number of elements to be sorted

printf("Enter the number of elements to be sorted: ");

scanf("%d", &n);

// input elements if the array

for(i = 0; i < n; i++)

{

printf("Enter element no. %d: ", i+1);

scanf("%d", &arr[i]);

}

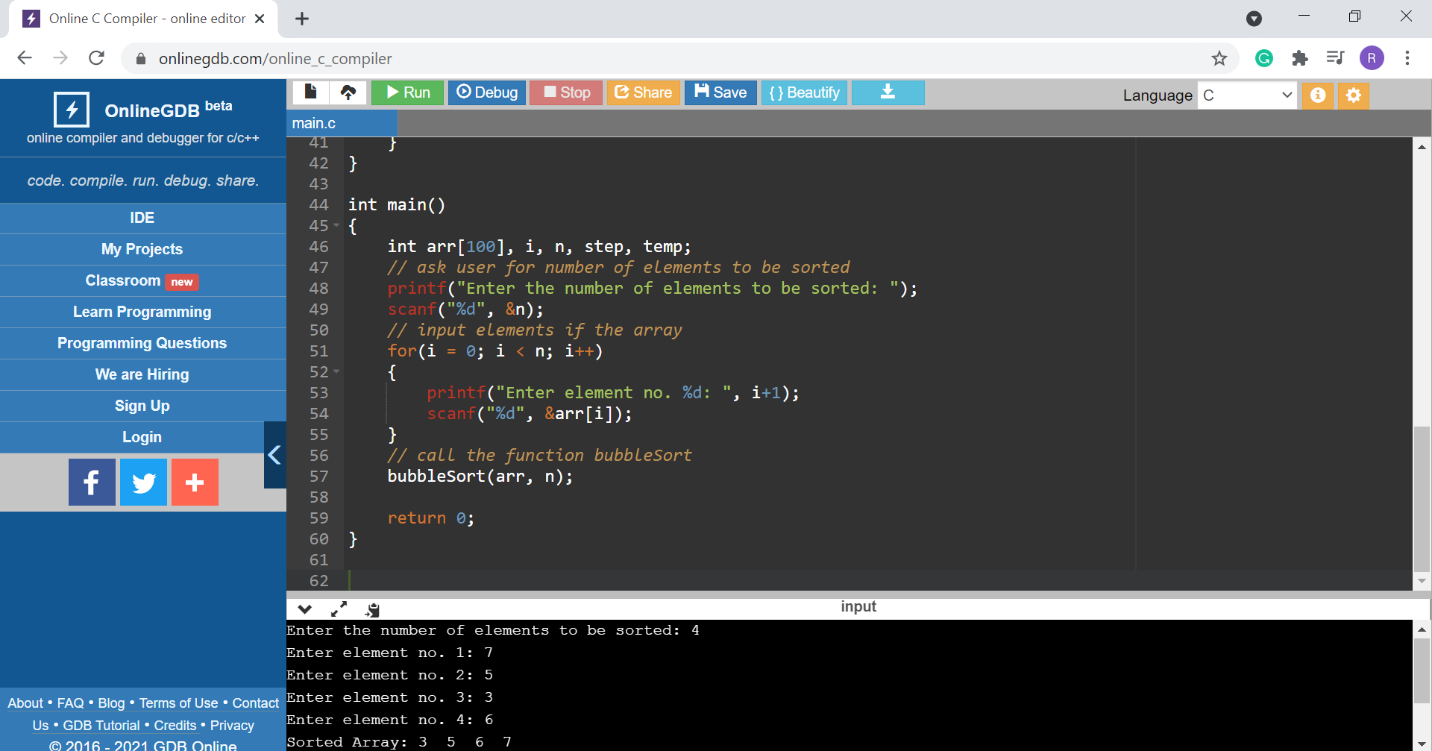
// call the function bubbleSort

bubbleSort(arr, n);

return 0;

}

**Output:**



10. **Algorithm to implement Merge Sort:**

Merge Sort follows the rule of **Divide and Conquer** to sort a given set of numbers/elements, recursively, hence consuming less time.

In Merge Sort, the given unsorted array with n elements, is divided into n subarrays, each having one element, because a single element is always sorted in itself. Then, it repeatedly merges these subarrays, to produce new sorted subarrays, and in the end, one complete sorted array is produced.

The concept of Divide and Conquer involves three steps:

* Divide the problem into multiple small problems.
* Conquer the subproblems by solving them. The idea is to break down the problem into atomic subproblems, where they are actually solved.
* Combine the solutions of the subproblems to find the solution of the actual problem.

**Code:**

#include <stdio.h>

// lets take a[5] = {32, 45, 67, 2, 7} as the array to be sorted.

// merge sort function

void mergeSort(int a[], int p, int r)

{

int q;

if(p < r)

{

q = (p + r) / 2;

mergeSort(a, p, q);

mergeSort(a, q+1, r);

merge(a, p, q, r);

}

}

// function to merge the subarrays

void merge(int a[], int p, int q, int r)

{

int b[5]; //same size of a[]

int i, j, k;

k = 0;

i = p;

j = q + 1;

while(i <= q && j <= r)

{

if(a[i] < a[j])

{

b[k++] = a[i++]; // same as b[k]=a[i]; k++; i++;

}

else

{

b[k++] = a[j++];

}

}

while(i <= q)

{

b[k++] = a[i++];

}

while(j <= r)

{

b[k++] = a[j++];

}

for(i=r; i >= p; i--)

{

a[i] = b[--k]; // copying back the sorted list to a[]

}

}

// function to print the array

void printArray(int a[], int size)

{

int i;

for (i=0; i < size; i++) {

printf("%d ", a[i]);

}

printf("\n");

}

int main()

{

int arr[] = {32, 45, 67, 2, 7};

int len = sizeof(arr)/sizeof(arr[0]);

printf("Given array: \n");

printArray(arr, len);

// calling merge sort

mergeSort(arr, 0, len - 1);

printf("\nSorted array: \n");

printArray(arr, len);

return 0;

}

**Output**:

