**WEEK 1**

* **Raahat Arora**

**Question 1-**

#include<iostream>

using namespace std;

int binarySearch(int arr[], int low, int high, int x) //Iterative solution

{

while (low <= high) {

int mid = low + (high - low) / 2;

if (arr[mid] == x) //Check if x os present at mid

return mid;

if (arr[mid] < x) //If x greater, ignore left half

low = mid + 1;

else //If x is smaller, ignore right half

high = mid - 1;

}

//If we reach here, element was not present

return -1;

}

int main()

{

int arr[] = { 2, 3, 4, 10, 40 };

int x = 10;

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

int result = binarySearch(arr, 0, n-1, x);

if(result == -1) cout << "Element is not present in array";

else cout << "Element is present at index " << result;

return 0;

}

**Output –**

****

**Question 2-**

#include <iostream>

using namespace std;

class Stack {

private:

int arr[100];

int top;

public:

Stack() {

top = -1;

}

void push(int value) {

if (top == 99) {

cout << "Stack Overflow! Cannot push " << value << " onto the stack.\n";

} else {

arr[++top] = value;

cout << value << " pushed onto the stack.\n";

}

}

void pop() {

if (top == -1) {

cout << "Stack Underflow! No elements to pop.\n";

} else {

cout << arr[top--] << " popped from the stack.\n"; // Return the element and decrement the top

}

}

void displayTop() {

if (top == -1) {

cout << "The stack is empty.\n";

} else {

cout << "Top element is: " << arr[top] << endl;

}

}

bool isEmpty() {

return top == -1; // If top is -1, the stack is empty

}

};

int main() {

Stack s;

s.push(10);

s.push(20);

s.push(30);

s.displayTop();

s.pop();

s.displayTop();

if (s.isEmpty()) {

cout << "The stack is empty.\n";

} else {

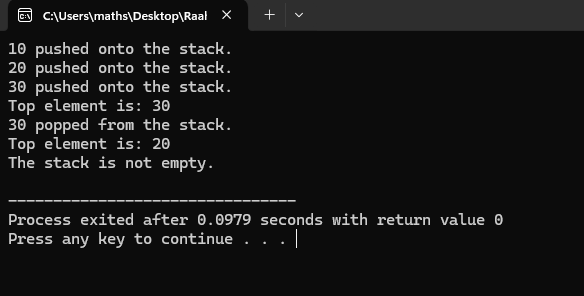
cout << "The stack is not empty.\n";

}

return 0;

}

**Output –**



**Question 3-**

#include <iostream>

using namespace std;

class Queue {

private:

int arr[100];

int front, rear;

public:

Queue() {

front = -1;

rear = -1;

}

void enqueue(int value) {

if (rear == 99) {

cout << "Queue Overflow\n";

} else {

if (front == -1) {

front = 0;

}

arr[++rear] = value;

cout << value << " enqueued\n";

}

}

void dequeue() {

if (front == -1) {

cout << "Queue Underflow\n";

} else {

cout << arr[front++] << " dequeued\n";

if (front > rear) {

front = rear = -1;

}

}

}

void displayFront() {

if (front == -1) {

cout << "Queue is empty\n";

} else {

cout << "Front element is: " << arr[front] << endl;

}

}

bool isEmpty() {

return front == -1;

}

};

int main() {

Queue q;

q.enqueue(10);

q.enqueue(20);

q.enqueue(30);

q.displayFront();

q.dequeue();

q.displayFront();

if (q.isEmpty()) {

cout << "Queue is empty\n";

} else {

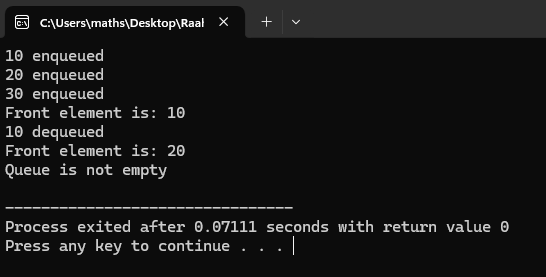
cout << "Queue is not empty\n";

}

return 0;

}

**Output –**



**Question 4-**

#include <iostream>

using namespace std;

class Node {

public:

int data;

Node\* next;

Node\* prev;

Node(int value) {

data = value;

next = prev = NULL;

}

};

class DoublyLinkedList {

private:

Node\* head;

Node\* tail;

public:

DoublyLinkedList() {

head = tail = NULL;

}

void insertAtBeginning(int value) {

Node\* newNode = new Node(value);

if (head == NULL) {

head = tail = newNode;

} else {

newNode->next = head;

head->prev = newNode;

head = newNode;

}

}

void insertAtEnd(int value) {

Node\* newNode = new Node(value);

if (tail == NULL) {

head = tail = newNode;

} else {

tail->next = newNode;

newNode->prev = tail;

tail = newNode;

}

}

void deleteFromBeginning() {

if (head == NULL) {

cout << "List is empty\n";

} else {

Node\* temp = head;

head = head->next;

if (head != NULL) {

head->prev = NULL;

} else {

tail = NULL;

}

delete temp;

}

}

void deleteFromEnd() {

if (tail == NULL) {

cout << "List is empty\n";

} else {

Node\* temp = tail;

tail = tail->prev;

if (tail != NULL) {

tail->next = NULL;

} else {

head = NULL;

}

delete temp;

}

}

void traverseForward() {

Node\* temp = head;

while (temp != NULL) {

cout << temp->data << " ";

temp = temp->next;

}

cout << endl;

}

void traverseBackward() {

Node\* temp = tail;

while (temp != NULL) {

cout << temp->data << " ";

temp = temp->prev;

}

cout << endl;

}

};

int main() {

DoublyLinkedList list;

list.insertAtBeginning(10);

list.insertAtEnd(20);

list.insertAtEnd(30);

list.insertAtBeginning(5);

cout << "Forward traversal: ";

list.traverseForward();

cout << "Backward traversal: ";

list.traverseBackward();

list.deleteFromBeginning();

list.deleteFromEnd();

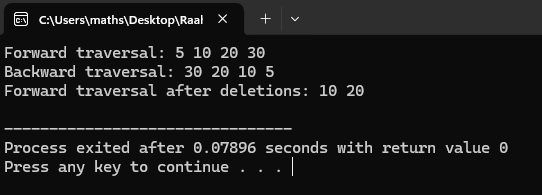
cout << "Forward traversal after deletions: ";

list.traverseForward();

return 0;

}

**Output –**



**Question 5-**

#include <iostream>

using namespace std;

class Graph {

private:

int matrix[5][5];

int vertices;

public:

Graph(int v) {

vertices = v;

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

for (int j = 0; j < vertices; j++) {

matrix[i][j] = 0;

}

}

}

void addEdge(int u, int v) {

if (u >= 0 && u < vertices && v >= 0 && v < vertices) {

matrix[u][v] = 1;

matrix[v][u] = 1;

}

}

void removeEdge(int u, int v) {

if (u >= 0 && u < vertices && v >= 0 && v < vertices) {

matrix[u][v] = 0;

matrix[v][u] = 0;

}

}

void displayGraph() {

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

for (int j = 0; j < vertices; j++) {

cout << matrix[i][j] << " ";

}

cout << endl;

}

}

};

int main() {

Graph g(5);

g.addEdge(0, 1);

g.addEdge(1, 2);

g.addEdge(2, 3);

g.addEdge(3, 4);

cout << "Graph after adding edges:" << endl;

g.displayGraph();

g.removeEdge(1, 2);

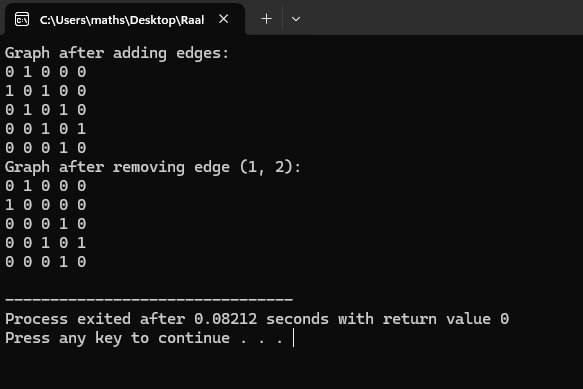
cout << "Graph after removing edge (1, 2):" << endl;

g.displayGraph();

return 0;

}

**Output-**



**Question 6-**

#include <iostream>

using namespace std;

class Node {

public:

int data;

Node\* left;

Node\* right;

Node(int value) {

data = value;

left = right = NULL;

}

};

class BST {

private:

Node\* root;

Node\* insert(Node\* node, int value) {

if (node == NULL) {

return new Node(value);

}

if (value < node->data) {

node->left = insert(node->left, value);

} else {

node->right = insert(node->right, value);

}

return node;

}

Node\* deleteNode(Node\* root, int value) {

if (root == NULL) {

return root;

}

if (value < root->data) {

root->left = deleteNode(root->left, value);

} else if (value > root->data) {

root->right = deleteNode(root->right, value);

} else {

if (root->left == NULL) {

Node\* temp = root->right;

delete root;

return temp;

} else if (root->right == NULL) {

Node\* temp = root->left;

delete root;

return temp;

}

Node\* temp = minValueNode(root->right);

root->data = temp->data;

root->right = deleteNode(root->right, temp->data);

}

return root;

}

Node\* minValueNode(Node\* node) {

Node\* current = node;

while (current && current->left != NULL) {

current = current->left;

}

return current;

}

bool search(Node\* root, int value) {

if (root == NULL) {

return false;

}

if (value == root->data) {

return true;

}

if (value < root->data) {

return search(root->left, value);

}

return search(root->right, value);

}

void inorder(Node\* root) {

if (root != NULL) {

inorder(root->left);

cout << root->data << " ";

inorder(root->right);

}

}

public:

BST() {

root = NULL;

}

void insert(int value) {

root = insert(root, value);

}

void deleteNode(int value) {

root = deleteNode(root, value);

}

bool search(int value) {

return search(root, value);

}

void inorder() {

inorder(root);

cout << endl;

}

};

int main() {

BST tree;

tree.insert(50);

tree.insert(30);

tree.insert(20);

tree.insert(40);

tree.insert(70);

tree.insert(60);

tree.insert(80);

cout << "Inorder traversal: ";

tree.inorder();

cout << "Searching for 40: " << (tree.search(40) ? "Found" : "Not Found") << endl;

cout << "Searching for 25: " << (tree.search(25) ? "Found" : "Not Found") << endl;

tree.deleteNode(20);

cout << "Inorder traversal after deleting 20: ";

tree.inorder();

tree.deleteNode(30);

cout << "Inorder traversal after deleting 30: ";

tree.inorder();

tree.deleteNode(50);

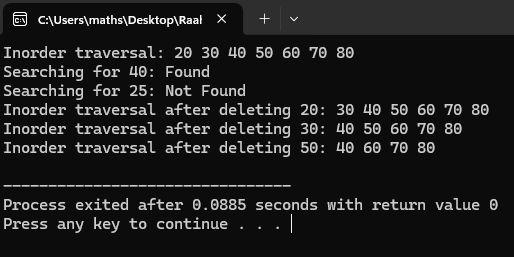
cout << "Inorder traversal after deleting 50: ";

tree.inorder();

return 0;

}

**Output –**



**Question 7-**

#include <iostream>

using namespace std;

int gcd(int a, int b) {

while (b != 0) {

int temp = b;

b = a % b;

a = temp;

}

return a;

}

int main() {

int a, b;

cout << "Enter two integers: ";

cin >> a >> b;

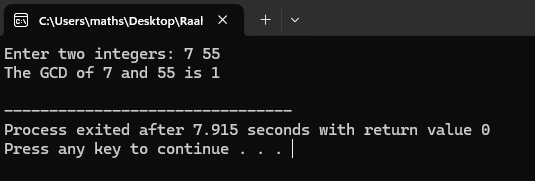
int result = gcd(a, b);

cout << "The GCD of " << a << " and " << b << " is " << result << endl;

return 0;

}

**Output –**



**Question 8-**

#include <iostream>

using namespace std;

int gcd(int a, int b) {

int min\_val = (a < b) ? a : b;

for (int i = min\_val; i > 0; i--) {

if (a % i == 0 && b % i == 0) {

return i;

}

}

return 1; // If no common divisor, GCD is 1

}

int main() {

int a, b;

cout << "Enter two integers: ";

cin >> a >> b;

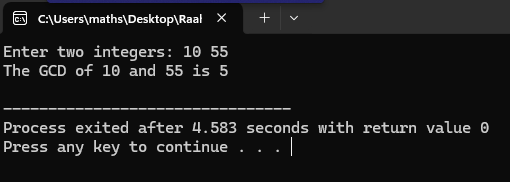
int result = gcd(a, b);

cout << "The GCD of " << a << " and " << b << " is " << result << endl;

return 0;

}

**Output –**



**Question 9-**

#include <iostream>

using namespace std;

void selectionSort(int arr[], int n) {

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

int minIdx = i;

for (int j = i + 1; j < n; j++) {

if (arr[j] < arr[minIdx]) {

minIdx = j;

}

}

if (minIdx != i) {

swap(arr[i], arr[minIdx]);

}

}

}

void bubbleSort(int arr[], int n) {

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

bool swapped = false;

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

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

swap(arr[j], arr[j + 1]);

swapped = true;

}

}

if (!swapped) {

break; // If no two elements were swapped, the array is sorted

}

}

}

void printArray(int arr[], int n) {

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

cout << arr[i] << " ";

}

cout << endl;

}

int main() {

int arr1[] = {64, 25, 12, 22, 11};

int arr2[] = {64, 25, 12, 22, 11};

int n = sizeof(arr1) / sizeof(arr1[0]);

cout << "Original array: ";

printArray(arr1, n);

selectionSort(arr1, n);

cout << "Sorted array using Selection Sort: ";

printArray(arr1, n);

bubbleSort(arr2, n);

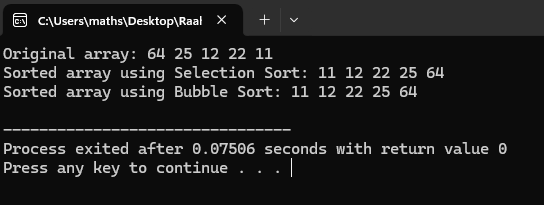
cout << "Sorted array using Bubble Sort: ";

printArray(arr2, n);

return 0;

}

**Output –**



**Question 10-**

#include <iostream>

#include <string>

using namespace std;

int bruteForceSearch(const string& text, const string& pattern) {

int n = text.length();

int m = pattern.length();

// Traverse the text

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

int j = 0;

// Compare the pattern with the substring of text

while (j < m && text[i + j] == pattern[j]) {

j++;

}

// If j equals m, it means a match is found

if (j == m) {

return i; // Match found at index i

}

}

return -1; // No match found

}

int main() {

string text, pattern;

// Input text and pattern

cout << "Enter the text: ";

getline(cin, text); // Using getline to handle input with spaces

cout << "Enter the pattern: ";

getline(cin, pattern); // Using getline to handle input with spaces

int result = bruteForceSearch(text, pattern);

if (result != -1) {

cout << "Pattern found at index " << result << endl;

} else {

cout << "Pattern not found!" << endl;

}

return 0;

}

**Output –**

