Advance Data Structure & Algorithm
Course Code: R1UC503B

Lab File

For

BACHELOR OF

ENGINEERING & TECHNOLOGY

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

GALGOTIAS UNIVERSITY, GREATER NOIDA

UTTAR PRADESH

Student Name:	Jagjeet Singh
Admission No:	22SCSE1120009

Semester : V

STUDENT NAME : Jagjeet Singh **ADMISSION NO. : 22scse1120009**

DATE: **EXPERIMENT NO. : 1**

AIM : **Find the Maximum and Minimum Elements in an Array:**

PROGRAM : Write a function to find the maximum and minimum elements in an array.

Sourcecode:

```
import java.util.Scanner;
public class minmax {
   public static int maxx(int[] a, int m) {
      for (int i = 0; i < a.length; i++) {
         if (a[i] > m) {
            m = a[i];
         }
      }
      return m;
   }
   public static int minn(int[] a, int m) {
      for (int i = 0; i < a.length; i++) {
         if (a[i] < m) {
            m = a[i];
         }
      }
      return m;
   }
   public static void main(String[] args) {
      Scanner obj = new Scanner(System.in);
      System.out.println("Enter the size of the array:");
      int size = obj.nextInt();
      int[] a = new int[size];
      for (int i = 0; i < a.length; i++) {
         System.out.println("Enter the elements:"+i);
         a[i] = obj.nextInt();
      }
      int m = Integer.MIN_VALUE;
      int lr = maxx(a, m);
      System.out.println("Max value in arr" + lr);
      m = Integer.MAX_VALUE;
      int sm = minn(a, m);
      System.out.println("Min value in arr " + sm);

   }
}
```

Output :

```
Enter the size of the array:
 5
Enter the elements:0
1
Enter the elements:1
1
Enter the elements:2
1
Enter the elements:3
1
Enter the elements:4
2
Max value in arr2
Min value in arr 1
```

STUDENT NAME : Jagjeet Singh **ADMISSION NO. : 22scse1120009**

DATE: **EXPERIMENT NO. : 2**

AIM : Reverse an Array

PROGRAM : Write a function to reverse an array in place.

Sourcecode:

```
import java.util.Scanner;
public class reversearr {
   public void rr(int a[]){
      int i=0;
      int j=a.length-1;
      while (i<j){
         int temp=a[i];
         a[i]=a[j];
         a[j]=temp;

         i++;
         j--;
      }
      for(int k=0;k<a.length;k++){
         System.out.print(a[k]+" ");
      }

   }
   public static void main(String[] args) {
      Scanner obj = new Scanner(System.in);
      System.out.println("Enter the size of the array:");
      int size = obj.nextInt();
      int[] a = new int[size];
      for (int i = 0; i < a.length; i++) {
         System.out.println("Enter the elements:"+i);
         a[i] = obj.nextInt();
      }

      reversearr palat = new reversearr();
      palat.rr(a);
   }
}
```

Output :

```
Enter the size of the array:
 5
Enter the elements:0
1
Enter the elements:1
2
Enter the elements:2
3
Enter the elements:3
4
Enter the elements:4
5
5 4 3 2 1
```

STUDENT NAME : Jagjeet Singh **ADMISSION NO. : 22scse1120009**

DATE: **EXPERIMENT NO. : 3**

AIM : **Find the Kth Smallest/Largest Element in an Array:**

PROGRAM : Write a function to find the Kthsmallest or largest element in an array.

Sourcecode:

```
import java.util.Arrays;

public class Main {
   public static int findKthSmallest(int[] arr, int k) {
      Arrays.sort(arr);
      return arr[k - 1];
   }

   public static int findKthLargest(int[] arr, int k) {
      Arrays.sort(arr);
      return arr[arr.length - k];
   }

   public static void main(String[] args) {
      int[] arr = {7, 10, 4, 3, 20, 15};
      int k = 3;

      System.out.println("Kth Smallest Element: " + findKthSmallest(arr, k));
      System.out.println("Kth Largest Element: " + findKthLargest(arr, k));
   }
}
```

STUDENT NAME : Jagjeet Singh **ADMISSION NO. : 22scse1120009**

DATE: **EXPERIMENT NO. : 4**

AIM : Sort an Array of 0s, 1s, and 2s

PROGRAM : Write a function to find the maximum and minimum elements in an array.

Sourcecode:

```
public class Main {
   public static void main(String[] args) {
      int[] nums = {0, 1, 2, 1, 0, 2, 1};
      sortColors(nums);

      System.out.println("Sorted Array:");
      for (int num : nums) {
         System.out.print(num + " ");
      }
   }

   public static void sortColors(int[] nums) {
      int low = 0, mid = 0, high = nums.length - 1;

      while (mid <= high) {
         if (nums[mid] == 0) {
            int temp = nums[low];
            nums[low] = nums[mid];
            nums[mid] = temp;
            low++;
            mid++;
         } else if (nums[mid] == 1) {
            mid++;
         } else {
            int temp = nums[mid];
            nums[mid] = nums[high];
            nums[high] = temp;
            high--;
         }
      }
   }
}
```

```
Sorted Array:
0 0 1 1 1 2 2
```

STUDENT NAME : Jagjeet Singh ADMISSION NO. : 22scse1120009

DATE: EXPERIMENT NO. : 5

AIM : Move All Zeroes to End of Array

PROGRAM : Write a function to find the maximum and minimum elements in an array.

Sourcecode:

```
public class Main {
    public static void main(String[] args) {
        int[] nums = {0, 1, 0, 3, 12};
        moveZeroes(nums);

        System.out.println("Array After Moving Zeroes:");
        for (int num : nums) {
            System.out.print(num + " ");
        }
    }

    public static void moveZeroes(int[] nums) {
        int index = 0;

        for (int num : nums) {
            if (num != 0) {
                nums[index++] = num;
            }
        }

        while (index < nums.length) {
            nums[index++] = 0;
        }
    }
}
```

```
Array After Moving Zeroes:
1 3 12 0 0
```

STUDENT NAME : Jagjeet Singh ADMISSION NO. : 22scse1120009

DATE: EXPERIMENT NO. : 6

AIM : Reverse a Linked List

PROGRAM : Write a function to find the maximum and minimum elements in an array.

Sourcecode:

```
class ListNode {
   int val;
   ListNode next;

   ListNode(int val) {
      this.val = val;
   }
}

public class Main {
   public static void main(String[] args) {
      ListNode head = new ListNode(1);
      head.next = new ListNode(2);
      head.next.next = new ListNode(3);
      head.next.next.next = new ListNode(4);

      System.out.println("Original Linked List:");
      printList(head);

      head = reverseList(head);

      System.out.println("Reversed Linked List:");
      printList(head);
   }

   public static ListNode reverseList(ListNode head) {
      ListNode prev = null;

      while (head != null) {
```

```
            ListNode nextNode = head.next;
            head.next = prev;
            prev = head;
            head = nextNode;
        }

        return prev;
    }

    public static void printList(ListNode head) {
        while (head != null) {
            System.out.print(head.val + " ");
            head = head.next;
        }
        System.out.println();
    }
}
```

```
Original Linked List:
1 2 3 4
Reversed Linked List:
4 3 2 1
```

STUDENT NAME : Jagjeet Singh **ADMISSION NO. : 22scse1120009**

DATE: **EXPERIMENT NO. : 7**

AIM : Detect a Cycle in a Linked List

PROGRAM : Write a function to find the maximum and minimum elements in an array.

Sourcecode:

```
class ListNode {
   int val;
   ListNode next;

   ListNode(int val) {
      this.val = val;
   }
}

public class Main {
   public static void main(String[] args) {
      ListNode head = new ListNode(3);
      head.next = new ListNode(2);
      head.next.next = new ListNode(0);
      head.next.next.next = new ListNode(-4);
      head.next.next.next.next = head.next; // Creates a cycle

      if (hasCycle(head)) {
         System.out.println("Cycle detected in the linked list.");
      } else {
         System.out.println("No cycle detected in the linked list.");
      }
   }

   public static boolean hasCycle(ListNode head) {
      ListNode slow = head, fast = head;

      while (fast != null && fast.next != null) {
         slow = slow.next;
```

```
            fast = fast.next.next;

            if (slow == fast) {
                return true;
            }
        }

        return false;
    }
}
```

```
Cycle detected in the linked list.
```

STUDENT NAME : Jagjeet Singh ADMISSION NO. : 22scse1120009

DATE: EXPERIMENT NO. : 8

AIM : Find the Middle of a Linked List

PROGRAM : Write a function to find the maximum and minimum elements in an array.

Sourcecode:

```
class ListNode {
   int val;
   ListNode next;

   ListNode(int val) {
      this.val = val;
   }
}

public class Main {
   public static void main(String[] args) {
      ListNode head = new ListNode(1);
      head.next = new ListNode(2);
      head.next.next = new ListNode(3);
      head.next.next.next = new ListNode(4);
      head.next.next.next.next = new ListNode(5);

      ListNode middle = findMiddle(head);

      System.out.println("Middle Element of the Linked List: " + middle.val);
   }

   public static ListNode findMiddle(ListNode head) {
      ListNode slow = head, fast = head;

      while (fast != null && fast.next != null) {
         slow = slow.next;
         fast = fast.next.next;
```

```
        }

        return slow;
    }
}
```

STUDENT NAME : Jagjeet Singh **ADMISSION NO. : 22scse1120009**

DATE: **EXPERIMENT NO. : 9**

AIM : Merge Two Sorted Linked Lists

PROGRAM : Write a function to find the maximum and minimum elements in an array.

Sourcecode:

```
class ListNode {
   int val;
   ListNode next;

   ListNode(int val) {
      this.val = val;
   }
}

public class Main {
   public static void main(String[] args) {
      ListNode l1 = new ListNode(1);
      l1.next = new ListNode(3);
      l1.next.next = new ListNode(5);

      ListNode l2 = new ListNode(2);
      l2.next = new ListNode(4);
      l2.next.next = new ListNode(6);

      ListNode merged = mergeTwoLists(l1, l2);

      System.out.println("Merged Linked List:");
      printList(merged);
   }

   public static ListNode mergeTwoLists(ListNode l1, ListNode l2) {
      ListNode dummy = new ListNode(-1);
      ListNode current = dummy;

      while (l1 != null && l2 != null) {
```

```
            if (l1.val <= l2.val) {
                current.next = l1;
                l1 = l1.next;
            } else {
                current.next = l2;
                l2 = l2.next;
            }
            current = current.next;
        }

        current.next = (l1 != null) ? l1 : l2;

        return dummy.next;
    }

    public static void printList(ListNode head) {
        while (head != null) {
            System.out.print(head.val + " ");
            head = head.next;
        }
        System.out.println();
    }
}
```

```
Merged Linked List:
1 2 3 4 5 6
```

STUDENT NAME : Jagjeet Singh ADMISSION NO. : 22scse1120009

DATE: EXPERIMENT NO. : 10

AIM : Remove Nth Node from End of List

PROGRAM : Write a function to find the maximum and minimum elements in an array.

Sourcecode:

```
public class Main {
   public static void main(String[] args) {
      ListNode head = new ListNode(1);
      head.next = new ListNode(2);
      head.next.next = new ListNode(3);
      head.next.next.next = new ListNode(4);
      head.next.next.next.next = new ListNode(5);

      int n = 2;
      head = removeNthFromEnd(head, n);

      System.out.println("Linked List After Removing " + n + "th Node From End:");
      printList(head);
   }

   public static ListNode removeNthFromEnd(ListNode head, int n) {
      ListNode dummy = new ListNode(0);
      dummy.next = head;
      ListNode first = dummy;
      ListNode second = dummy;

      for (int i = 0; i <= n; i++) {
         first = first.next;
      }

      while (first != null) {
         first = first.next;
         second = second.next;
      }
```

```
        second.next = second.next.next;

        return dummy.next;
    }

    public static void printList(ListNode head) {
        while (head != null) {
            System.out.print(head.val + " ");
            head = head.next;
        }
        System.out.println();
    }
}
```

```
Linked List After Removing 2th Node From End:
1 2 3 5
```

STUDENT NAME : Jagjeet Singh **ADMISSION NO. : 22scse1120009**

DATE: **EXPERIMENT NO. : 11**

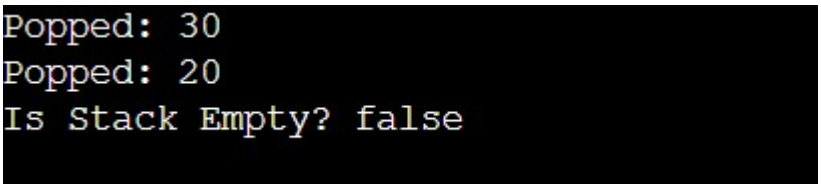
AIM : Implement a Stack Using Arrays

PROGRAM : Write a function to find the maximum and minimum elements in an array.

Sourcecode:

```
class Stack {
    private int[] stack;
    private int top;

    public Stack(int size) {
        stack = new int[size];
        top = -1;
    }

    public void push(int x) {
        if (top == stack.length - 1) {
            System.out.println("Stack Overflow");
            return;
        }
        stack[++top] = x;
    }

    public int pop() {
        if (top == -1) {
            System.out.println("Stack Underflow");
            return -1;
        }
        return stack[top--];
    }

    public int peek() {
        if (top == -1) {
            System.out.println("Stack is Empty");
            return -1;
        }
        return stack[top];
```

```
    }

    public boolean isEmpty() {
        return top == -1;
    }
}

public class Main {
    public static void main(String[] args) {
        Stack stack = new Stack(5);

        stack.push(10);
        stack.push(20);
        stack.push(30);

        System.out.println("Top Element: " + stack.peek());

        System.out.println("Popped: " + stack.pop());
        System.out.println("Popped: " + stack.pop());

        System.out.println("Is Stack Empty? " + stack.isEmpty());
    }
}
```



STUDENT NAME : Jagjeet Singh **ADMISSION NO. : 22scse1120009**

DATE: **EXPERIMENT NO. : 12**

AIM : Implement a Stack Using Arrays

PROGRAM : Write a function to implement a stack using an array or list with basic operations: push, pop, peek, and isEmpty.

Sourcecode:

```
#include <iostream>
#include <vector>
using namespace std;
class Stack {
private:
vector<int> arr;
public:
void push(int x) {
arr.push_back(x);
}
int pop() {
if (isEmpty()) {
cout << "Stack Underflow!" << endl;
return -1;
}
int topElement = arr.back();
arr.pop_back();
return topElement;
}
int peek() {
if (isEmpty()) {
cout << "Stack is empty!" << endl;
return -1;
}
return arr.back();
}
bool isEmpty() {
return arr.empty();
}
};
```

Output

```
Top Element: 30
Popped: 30
Popped: 20
Is Stack Empty? false
```

STUDENT NAME : Jagjeet Singh ADMISSION NO. : 22scse1120009

DATE: EXPERIMENT NO. : 13

AIM : Implement a Stack Using Linked List

PROGRAM : Write a function to implement a stack using a linked list with basic operations: push, pop, peek, and isEmpty.

Sourcecode:

```
#include <iostream>
using namespace std;
struct Node {
int data;
Node* next;
};
class Stack {
private:
Node* top;
public:
Stack() {
top = nullptr;
}
void push(int x) {
Node* newNode = new Node();
newNode->data = x;
newNode->next = top;
top = newNode;
}
int pop() {
if (isEmpty()) {
cout << "Stack Underflow!" << endl;
return -1;
}
int poppedValue = top->data;
Node* temp = top;
top = top->next;
delete temp;
```

```
return poppedValue;
}
int peek() {
if (isEmpty()) {
cout << "Stack is empty!" << endl;
return -1;
}
return top->data;
}
bool isEmpty() {
return top == nullptr;
}
};
```

Output

```
Stack Underflow!
Top Element: 30
Popped Element: 30
```

STUDENT NAME : Jagjeet Singh **ADMISSION NO. : 22scse1120009**

DATE: **EXPERIMENT NO. : 14**

AIM : Check for Balanced Parentheses

Sourcecode:

```
#include <iostream>
#include <stack>
using namespace std;
bool isBalanced(string expr) {
stack<char> s;
for (char ch : expr) {
if (ch == '(' || ch == '{' || ch == '[') {
s.push(ch);
} else if (ch == ')' || ch == '}' || ch == ']') {
if (s.empty() || (ch == ')' && s.top() != '(') || (ch == '}' && s.top() !=
'{') || (ch == ']' && s.top() != '[')) {
return false;
}
s.pop();
}
}
return s.empty();}
```

Output

```
Balanced
```

STUDENT NAME : Jagjeet Singh **ADMISSION NO. : 22scse1120009**

DATE: **EXPERIMENT NO. : 15**

AIM : Evaluate Postfix Expression

Sourcecode:

```
#include <iostream>
#include <stack>
#include <cctype>
using namespace std;
int evaluatePostfix(string expr) {
stack<int> s;
for (char ch : expr) {
if (isdigit(ch)) {
s.push(ch - '0');
} else {
int b = s.top(); s.pop();
int a = s.top(); s.pop();
switch (ch) {
case '+': s.push(a + b); break;
case '-': s.push(a - b); break;
case '*': s.push(a * b); break;
case '/': s.push(a / b); break;
}
}
}
return s.top();
}
```

Output

```
Result: 9
```

STUDENT NAME : Jagjeet Singh ADMISSION NO. : 22scse1120009

DATE: EXPERIMENT NO. : 16

AIM : Next Greater Element

Sourcecode:

```
#include <iostream>
#include <vector>
#include <stack>
using namespace std;
vector<int> nextGreaterElement(vector<int>& nums) {
vector<int> result(nums.size(), -1);
stack<int> s;
for (int i = 0; i < nums.size(); ++i) {
while (!s.empty() && nums[s.top()] < nums[i]) {
result[s.top()] = nums[i];
s.pop();
}
s.push(i);
}
return result;
}
};
```

Output

```
Next Greater Element: 2 3 -1
```

STUDENT NAME : Jagjeet Singh **ADMISSION NO. : 22scse1120009**

DATE: **EXPERIMENT NO. : 17**

AIM : Implement a Queue Using Arrays/Lists

Sourcecode:

```
#include <iostream>
#include <vector>
using namespace std;
class Queue {
private:
vector<int> arr;
public:
void enqueue(int x) {
arr.push_back(x);
}
int dequeue() {
if (isEmpty()) {
cout << "Queue Underflow!" << endl;
return -1;
}
int frontElement = arr[0];
arr.erase(arr.begin());
return frontElement;
}
int front() {
if (isEmpty()) {
cout << "Queue is empty!" << endl;
return -1;
}
return arr[0];
}
bool isEmpty() {
return arr.empty();
}
};
```

Output

```
Enqueued: 10
Dequeued: 10
Queue is empty!
```

STUDENT NAME : Jagjeet Singh **ADMISSION NO. : 22scse1120009**

DATE: **EXPERIMENT NO. : 18**

AIM : Implement a Queue Using Linked List

Sourcecode:

```
#include <iostream>
using namespace std;
struct Node {
int data;
Node* next;
};
class Queue {
private:
Node* frontNode;
Node* rearNode;
public:
Queue() {
frontNode = rearNode = nullptr;
}
void enqueue(int x) {
Node* newNode = new Node();
newNode->data = x;
newNode->next = nullptr;
if (rearNode) {
rearNode->next = newNode;
}
rearNode = newNode;
if (!frontNode) {
frontNode = rearNode;
}
}
int dequeue() {
if (isEmpty()) {
cout << "Queue Underflow!" << endl;
return -1;
}
int frontValue = frontNode->data;
Node* temp = frontNode;
frontNode = frontNode->next;
delete temp;
if (!frontNode) {
rearNode = nullptr;
}
return frontValue;
}
int front() {
if (isEmpty()) {
cout << "Queue is empty!" << endl;
return -1;
}
return frontNode->data;
```

```
}
bool isEmpty() {
return frontNode == nullptr;
}
};
```

Output

```
Queue Underflow!
Front Element: 10
Dequeued: 10
```

STUDENT NAME : Jagjeet Singh **ADMISSION NO. : 22scse1120009**

DATE: **EXPERIMENT NO. : 19**

AIM : Implement a Circular Queue

Sourcecode:

```
#include <iostream>
using namespace std;
class CircularQueue {
private:
int* arr;
int size, front, rear, count;
public:
CircularQueue(int n) {
size = n;
arr = new int[n];
front = rear = count = 0;
}
void enqueue(int x) {
if (count == size) {
cout << "Queue Overflow!" << endl;
return;
}
arr[rear] = x;
rear = (rear + 1) % size;
count++;
}
int dequeue() {
if (isEmpty()) {
cout << "Queue Underflow!" << endl;
return -1;
}
int frontValue = arr[front];
front = (front + 1) % size;
count--;
return frontValue;
}
int frontElement() {
if (isEmpty()) {
cout << "Queue is empty!" << endl;
return -1;
}
return arr[front];
}
bool isEmpty() {
return count == 0;
}
};
```

Output

```
Queue Overflow!
Dequeued: 10
```

STUDENT NAME : Jagjeet Singh ADMISSION NO. : 22scse1120009

DATE: EXPERIMENT NO. : 20

AIM : Generate Binary Numbers from 1 to N

Sourcecode:

```
#include <iostream>
#include <queue>
using namespace std;
void generateBinary(int n) {
queue<string> q;
q.push("1");
while (n--) {
string curr = q.front();
q.pop();
cout << curr << " ";
q.push(curr + "0");
q.push(curr + "1");
}
}
```

Output

```
1 10 11 100 101
```

STUDENT NAME : Jagjeet Singh **ADMISSION NO. : 22scse1120009**

DATE: **EXPERIMENT NO. : 21**

AIM : Implement a Queue Using Stacks

Sourcecode:

```
#include <iostream>
#include <stack>
using namespace std;
class QueueUsingStacks {
private:
stack<int> s1, s2;
public:
void enqueue(int x) {
s1.push(x);
}
int dequeue() {
if (isEmpty()) {
cout << "Queue Underflow!" << endl;
return -1;
}
if (s2.empty()) {
while (!s1.empty()) {
s2.push(s1.top());
s1.pop();
}
}
int frontValue = s2.top();
s2.pop();
return frontValue;
}
bool isEmpty() {
return s1.empty() && s2.empty();
}
};
```

Output

```
Queue Underflow!
Dequeued: 10
```

STUDENT NAME : Jagjeet Singh ADMISSION NO. : 22scse1120009

DATE: EXPERIMENT NO. : 22

AIM : Implement a Binary Tree

Sourcecode:

```
#include <iostream>
using namespace std;
class Node {
public:
int data;
Node* left;
Node* right;
Node(int val) : data(val), left(nullptr), right(nullptr) {}
};
class BinaryTree {
private:
Node* root;
Node* insert(Node* node, int val) {
if (node == nullptr) {
return new Node(val);
}
if (val < node->data) {
node->left = insert(node->left, val);
} else {
node->right = insert(node->right, val);
}
return node;
}
Node* deleteNode(Node* root, int key) {
if (root == nullptr) return root;
if (key < root->data) {
root->left = deleteNode(root->left, key);
} else if (key > root->data) {
root->right = deleteNode(root->right, key);
} else {
if (root->left == nullptr) {
Node* temp = root->right;
delete root;
return temp;
} else if (root->right == nullptr) {
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 != nullptr) {
current = current->left;
}
return current;
}
void inorder(Node* node) {
if (node == nullptr) return;
inorder(node->left);
cout << node->data << " ";
inorder(node->right);
}
void preorder(Node* node) {
if (node == nullptr) return;
cout << node->data << " ";
preorder(node->left);
preorder(node->right);
}
void postorder(Node* node) {
if (node == nullptr) return;
postorder(node->left);
postorder(node->right);
cout << node->data << " ";
}
public:
BinaryTree() : root(nullptr) {}
void insert(int val) {
root = insert(root, val);
}
void deleteNode(int key) {
root = deleteNode(root, key);
}
void inorderTraversal() {
inorder(root);
cout << endl;
}
void preorderTraversal() {
preorder(root);
cout << endl;
}
void postorderTraversal() {
postorder(root);
cout << endl;
}
};
int main() {
BinaryTree tree;
tree.insert(50);
tree.insert(30);
tree.insert(20);
tree.insert(40);
tree.insert(70);
tree.insert(60);
tree.insert(80);
cout << "Inorder: ";
tree.inorderTraversal();
cout << "Preorder: ";
tree.preorderTraversal();
cout << "Postorder: ";
tree.postorderTraversal();
tree.deleteNode(20);
cout << "Inorder after deletion: ";
tree.inorderTraversal();
```

```
return 0;
}
};
```

Output

```
Inorder: 20 30 40 50 60 70 80
Preorder: 50 30 20 40 70 60 80
Postorder: 20 40 30 60 80 70 50
Inorder after deletion: 30 40 50 60 70 80
```

STUDENT NAME : Jagjeet Singh **ADMISSION NO. : 22scse1120009**

DATE: **EXPERIMENT NO. : 23**

AIM : Inorder Traversal

Sourcecode:

```
void inorder(Node* node) {
if (node == nullptr) return;
inorder(node->left);
cout << node->data << " ";
inorder(node->right);
}
```

Output

```
20 30 40 50 60 70 80
```

STUDENT NAME : Jagjeet Singh **ADMISSION NO. : 22scse1120009**

DATE: **EXPERIMENT NO. : 24**

AIM : Preorder Traversal

Sourcecode:

```
void preorder(Node* node) {
if (node == nullptr) return;
cout << node->data << " ";
preorder(node->left);
preorder(node->right);
}
```

Output

```
50 30 20 40 70 60 80
```

STUDENT NAME : Jagjeet Singh **ADMISSION NO. : 22scse1120009**

DATE: **EXPERIMENT NO. : 25**

AIM : **Postorder Traversal**

Sourcecode:

```
void postorder(Node* node) {
if (node == nullptr) return;
postorder(node->left);
postorder(node->right);
cout << node->data << " ";
}
```

Output

```
20 40 30 60 80 70 50
```

STUDENT NAME : Jagjeet Singh **ADMISSION NO. : 22scse1120009**

DATE: **EXPERIMENT NO. : 26**

AIM : Level Order Traversal

Sourcecode:

```
#include <queue>
void levelOrder(Node* root) {
if (root == nullptr) return;
queue<Node*> q;
q.push(root);
while (!q.empty()) {
Node* node = q.front();
cout << node->data << " ";
q.pop();
if (node->left) q.push(node->left);
if (node->right) q.push(node->right);
}
}
```

Output

```
50 30 70 20 40 60 80
```

STUDENT NAME : Jagjeet Singh **ADMISSION NO. : 22scse1120009**

DATE: **EXPERIMENT NO. : 27**

AIM : Height of a Binary Tree

Sourcecode:

```
int height(Node* node) {
if (node == nullptr) return 0;
int leftHeight = height(node->left);
int rightHeight = height(node->right);
return max(leftHeight, rightHeight) + 1;
}
```

Output

```
Height: 3
```

STUDENT NAME : Jagjeet Singh ADMISSION NO. : 22scse1120009

DATE: EXPERIMENT NO. : 28

AIM : Diameter of a Binary Tree

Sourcecode:

```
int diameter(Node* root, int &height) {
if (root == nullptr) {
height = 0;
return 0;
}
int leftHeight = 0, rightHeight = 0;
int leftDiameter = diameter(root->left, leftHeight);
int rightDiameter = diameter(root->right, rightHeight);
height = max(leftHeight, rightHeight) + 1;
return max(leftHeight + rightHeight + 1, max(leftDiameter, rightDiameter));
}
```

Output

```
Diameter: 5
```

STUDENT NAME : Jagjeet Singh ADMISSION NO. : 22scse1120009

DATE: EXPERIMENT NO. : 29

AIM : Check if a Binary Tree is Balanced

Sourcecode:

```
bool isBalanced(Node* root) {
int height = 0;
return isBalancedUtil(root, height);
}
bool isBalancedUtil(Node* node, int &height) {
if (node == nullptr) {
height = 0;
return true;
}
int leftHeight = 0, rightHeight = 0;
bool leftBalanced = isBalancedUtil(node->left, leftHeight);
bool rightBalanced = isBalancedUtil(node->right, rightHeight);
height = max(leftHeight, rightHeight) + 1;
return leftBalanced && rightBalanced && abs(leftHeight - rightHeight) <= 1;
}
```

Output

```
Balanced: Yes
```

STUDENT NAME : Jagjeet Singh ADMISSION NO. : 22scse1120009

DATE: EXPERIMENT NO. : 30

AIM : Lowest Common Ancestor

Sourcecode:

```
Node* LCA(Node* root, int n1, int n2) {
if (root == nullptr) return nullptr;
if (root->data == n1 || root->data == n2) return root;
Node* leftLCA = LCA(root->left, n1, n2);
Node* rightLCA = LCA(root->right, n1, n2);
if (leftLCA && rightLCA) return root;
return (leftLCA != nullptr) ? leftLCA : rightLCA;
}
```

Output

```
LCA of 20 and 40: 30
```

STUDENT NAME : Jagjeet Singh ADMISSION NO. : 22scse1120009

DATE: EXPERIMENT NO. : 31

AIM : Implement Graph Using Adjacency List

Sourcecode:

```
#include <iostream>
#include <vector>
using namespace std;
class Graph {
private:
int vertices;
vector<vector<int>> adjList;
public:
Graph(int v) : vertices(v) {
adjList.resize(vertices);
}
void addEdge(int u, int v) {
adjList[u].push_back(v);
adjList[v].push_back(u);
}
void display() {
for (int i = 0; i < vertices; i++) {
cout << i << ": ";
for (int j : adjList[i]) {
cout << j << " ";
}
cout << endl;
}
}
};
```

Output

```
0: 1 2
1: 0 2
2: 0 1
```

STUDENT NAME : Jagjeet Singh **ADMISSION NO. : 22scse1120009**

DATE: **EXPERIMENT NO. : 32**

AIM : **Breadth-First Search (BFS)**

Sourcecode:

```
#include <queue>
void BFS(Graph &graph, int start) {
vector<bool> visited(graph.vertices, false);
queue<int> q;
visited[start] = true;
q.push(start);
while (!q.empty()) {
int node = q.front();
cout << node << " ";
q.pop();
for (int neighbor : graph.adjList[node]) {
if (!visited[neighbor]) {
visited[neighbor] = true;
q.push(neighbor);
}
}
}
}
};
```

Output

```
BFS Traversal: 0 1 2
```

STUDENT NAME : Jagjeet Singh ADMISSION NO. : 22scse1120009

DATE: EXPERIMENT NO. : 33

AIM : Detect Cycle in an Undirected Graph

Sourcecode:

```
#include <iostream>
#include <vector>
using namespace std;
class Graph {
public:
int vertices;
vector<vector<int>> adjList;
Graph(int v) {
vertices = v;
adjList.resize(v);
}
void addEdge(int u, int v) {
adjList[u].push_back(v);
adjList[v].push_back(u);
}
bool dfs(int node, vector<bool>& visited, int parent) {
visited[node] = true;
for (int neighbor : adjList[node]) {
if (!visited[neighbor]) {
if (dfs(neighbor, visited, node)) {
return true;
}
} else if (neighbor != parent) {
return true;
}
}
return false;
}
bool detectCycle() {
vector<bool> visited(vertices, false);
for (int i = 0; i < vertices; i++) {
if (!visited[i]) {
if (dfs(i, visited, -1)) {
return true;
}
}
}
return false;
}
};
```

Output

```
Cycle Detected: Yes
```

STUDENT NAME : Jagjeet Singh ADMISSION NO. : 22scse1120009

DATE: EXPERIMENT NO. : 34

AIM : Connected Components in an Undirected Graph

Sourcecode:

```
#include <iostream>
#include <vector>
using namespace std;
class Graph {
public:
int vertices;
vector<vector<int>> adjList;
Graph(int v) {
vertices = v;
adjList.resize(v);
}
void addEdge(int u, int v) {
adjList[u].push_back(v);
adjList[v].push_back(u);
}
void dfs(int node, vector<bool>& visited) {
visited[node] = true;
for (int neighbor : adjList[node]) {
if (!visited[neighbor]) {
dfs(neighbor, visited);
}
}
}
int countComponents() {
vector<bool> visited(vertices, false);
int count = 0;
for (int i = 0; i < vertices; i++) {
if (!visited[i]) {
dfs(i, visited);
count++;
}
}
return count;
}
```

Output

```
Number of Connected Components: 2
```

STUDENT NAME : Jagjeet Singh ADMISSION NO. : 22scse1120009

DATE: EXPERIMENT NO. : 35

AIM : Find MST Using Kruskal's Algorithm

Sourcecode:

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
class DisjointSet {
public:
vector<int> parent, rank;
DisjointSet(int n) {
parent.resize(n);
rank.resize(n, 0);
for (int i = 0; i < n; i++) parent[i] = i;
}
int find(int u) {
if (u != parent[u]) parent[u] = find(parent[u]);
return parent[u];
}
void unite(int u, int v) {
int root_u = find(u), root_v = find(v);
if (root_u != root_v) {
if (rank[root_u] > rank[root_v]) {
parent[root_v] = root_u;
} else if (rank[root_u] < rank[root_v]) {
parent[root_u] = root_v;
} else {
parent[root_v] = root_u;
rank[root_u]++;
}
}
}
};
class Edge {
public:
int u, v, weight;
Edge(int u, int v, int weight) : u(u), v(v), weight(weight) {}
};
bool compare(Edge& e1, Edge& e2) {
return e1.weight < e2.weight;
}
class Graph {
public:
int vertices;
vector<Edge> edges;
Graph(int v) : vertices(v) {}
```

```
void addEdge(int u, int v, int weight) {
edges.push_back(Edge(u, v, weight));
}
void kruskalMST() {
sort(edges.begin(), edges.end(), compare);
DisjointSet ds(vertices);
vector<Edge> mst;
for (Edge& edge : edges) {
int u = edge.u, v = edge.v;
if (ds.find(u) != ds.find(v)) {
ds.unite(u, v);
mst.push_back(edge);
}
}
cout << "Edges in the MST:" << endl;
for (Edge& edge : mst) {
cout << edge.u << " - " << edge.v << " : " << edge.weight << endl;
}
}
};
```

Output

```
Edges in the MST:
0 - 1 : 1
1 - 2 : 2
```

STUDENT NAME : Jagjeet Singh ADMISSION NO. : 22scse1120009

DATE: EXPERIMENT NO. : 36

AIM : Find MST Using Prim's Algorithm

Sourcecode:

```
#include <iostream>
#include <vector>
#include <queue>
using namespace std;
class Graph {
public:
int vertices;
vector<vector<pair<int, int>>> adjList;
Graph(int v) : vertices(v) {
adjList.resize(v);
}
void addEdge(int u, int v, int weight) {
adjList[u].push_back({v, weight});
adjList[v].push_back({u, weight});
}
void primMST() {
vector<int> key(vertices, INT_MAX);
vector<int> parent(vertices, -1);
vector<bool> inMST(vertices, false);
key[0] = 0;
priority_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>>>
pq;
pq.push({0, 0});
while (!pq.empty()) {
int u = pq.top().second;
pq.pop();
inMST[u] = true;
for (auto& neighbor : adjList[u]) {
int v = neighbor.first, weight = neighbor.second;
if (!inMST[v] && weight < key[v]) {
key[v] = weight;
parent[v] = u;
pq.push({key[v], v});
}
}
}
cout << "Edges in the MST:" << endl;
for (int i = 1; i < vertices; i++) {
cout << parent[i] << " - " << i << endl;
}
}
};
```

Output

```
Edges in the MST:
0 - 1
1 - 2
```

STUDENT NAME : Jagjeet Singh **ADMISSION NO. : 22scse1120009**

DATE: **EXPERIMENT NO. : 37**

AIM : Fibonacci Sequence (Dynamic Programming)

Sourcecode:

```
#include <iostream>
#include <vector>
using namespace std;
int fib(int n) {
vector<int> dp(n + 1);
dp[0] = 0;
dp[1] = 1;
for (int i = 2; i <= n; i++) {
dp[i] = dp[i - 1] + dp[i - 2];
}
return dp[n];
}
```

Output

```
Fibonacci(5): 5
```

STUDENT NAME : Jagjeet Singh **ADMISSION NO. : 22scse1120009**

DATE: **EXPERIMENT NO. : 38**

AIM : Climbing Stairs

Sourcecode:

```
#include <iostream>
#include <vector>
using namespace std;
int climbStairs(int n) {
vector<int> dp(n + 1);
dp[0] = 1;
dp[1] = 1;
for (int i = 2; i <= n; i++) {
dp[i] = dp[i - 1] + dp[i - 2];
}
return dp[n];
}
```

Output

```
Ways to climb 5 stairs: 8
```

STUDENT NAME : Jagjeet Singh **ADMISSION NO. : 22scse1120009**

DATE: **EXPERIMENT NO. : 39**

AIM : Min Cost Climbing Stairs

Sourcecode:

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
int minCostClimbingStairs(vector<int>& cost) {
int n = cost.size();
vector<int> dp(n + 1);
dp[0] = 0;
dp[1] = 0;
for (int i = 2; i <= n; i++) {
dp[i] = min(dp[i - 1] + cost[i - 1], dp[i - 2] + cost[i - 2]);
}
return dp[n];
}
int main() {
vector<int> cost = {10, 15, 20};
cout << "Minimum cost to reach the top: " << minCostClimbingStairs(cost) << endl;
return 0;
```

Output

```
Minimum cost to reach the top: 15
```

STUDENT NAME : Jagjeet Singh **ADMISSION NO. : 22scse1120009**

DATE: **EXPERIMENT NO. : 40**

AIM : **House Robber**

Sourcecode:

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
int houseRobber(vector<int>& nums) {
int n = nums.size();
vector<int> dp(n + 1);
dp[0] = 0;
dp[1] = nums[0];
for (int i = 2; i <= n; i++) {
dp[i] = max(dp[i - 1], dp[i - 2] + nums[i - 1]);
}
return dp[n];
}
};
```

Output

```
Maximum amount robbed: 10
```

STUDENT NAME : Jagjeet Singh ADMISSION NO. : 22scse1120009

DATE: EXPERIMENT NO. : 41

AIM : Maximum Subarray Sum (Kadane's Algorithm)

Sourcecode:

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
int maxSubArray(vector<int>& nums) {
int maxSum = nums[0], currentSum = nums[0];
for (int i = 1; i < nums.size(); i++) {
currentSum = max(nums[i], currentSum + nums[i]);
maxSum = max(maxSum, currentSum);
}
return maxSum;
}
```

Output

```
Maximum Subarray Sum: 6
```

STUDENT NAME : Jagjeet Singh **ADMISSION NO. : 22scse1120009**

DATE: **EXPERIMENT NO. : 42**

AIM : Activity Selection

PROGRAM : Given a set of activities with start and end times, select the maximum number of activities that do not overlap.

Sourcecode:

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
struct Activity {
int start, end;
};
bool compare(Activity a, Activity b) {
return a.end < b.end;
}
int activitySelection(vector<Activity>& activities) {
sort(activities.begin(), activities.end(), compare);
int count = 1;
int lastSelected = 0;
for (int i = 1; i < activities.size(); i++) {
if (activities[i].start >= activities[lastSelected].end) {
count++;
lastSelected = i;
}
}
return count;
}
```

Output

```
Maximum number of activities: 4
```

STUDENT NAME : Jagjeet Singh ADMISSION NO. : 22scse1120009

DATE: EXPERIMENT NO. : 43

AIM : Fractional Knapsack Problem

PROGRAM : Given weights and values of items and the maximum capacity of a knapsack, determine the maximum value that can be obtained by including fractions of items

Sourcecode:

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
struct Item {
int value, weight;
};
bool compare(Item a, Item b) {
double ratioA = (double)a.value / a.weight;
double ratioB = (double)b.value / b.weight;
return ratioA > ratioB;
}
double fractionalKnapsack(int W, vector<Item>& items) {
sort(items.begin(), items.end(), compare);
double totalValue = 0;
for (Item& item : items) {
if (W == 0) break;
if (item.weight <= W) {
totalValue += item.value;
W -= item.weight;
} else {
totalValue += item.value * ((double)W / item.weight);
break;
}
}
return totalValue;
}
```

Output

```
Maximum value in knapsack: 240.0
```

STUDENT NAME : Jagjeet Singh ADMISSION NO. : 22scse1120009

DATE: EXPERIMENT NO. : 44

AIM : Huffman Coding

PROGRAM : Given a set of characters and their frequencies, construct the HuffmanTree to encode the characters.

Sourcecode:

```
#include <iostream>
#include <queue>
#include <vector>
#include <unordered_map>
using namespace std;
struct Node {
char ch;
int freq;
Node *left, *right;
Node(char c, int f) : ch(c), freq(f), left(nullptr), right(nullptr) {}
};
struct Compare {
bool operator()(Node* l, Node* r) {
return l->freq > r->freq;
}
};void printHuffmanCodes(Node* root, string str) {
if (!root) return;
if (root->ch != '$') cout << root->ch << ": " << str << endl;
printHuffmanCodes(root->left, str + "0");
printHuffmanCodes(root->right, str + "1");
}
void huffmanCoding(vector<char>& chars, vector<int>& freq) {
priority_queue<Node*, vector<Node*>, Compare> minHeap;
for (int i = 0; i < chars.size(); i++) {
minHeap.push(new Node(chars[i], freq[i]));
}
while (minHeap.size() != 1) {
Node *left = minHeap.top(); minHeap.pop();
Node *right = minHeap.top(); minHeap.pop();
Node *top = new Node('$', left->freq + right->freq);
top->left = left;
top->right = right;
minHeap.push(top);
}
printHuffmanCodes(minHeap.top(), "");
}
```

Output

```
a: 0
b: 10
c: 110
d: 111
```

STUDENT NAME : Jagjeet Singh **ADMISSION NO. : 22scse1120009**

DATE: **EXPERIMENT NO. : 45**

AIM : Job Sequencing Problem

PROGRAM : Given a set of jobs, each with a deadline and profit, maximize the total profit by scheduling the jobs to be done before their deadlines

Sourcecode:

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
struct Job {
int id, deadline, profit;
};
bool compare(Job a, Job b) {
return a.profit > b.profit;
}
int jobSequencing(vector<Job>& jobs, int n) {
sort(jobs.begin(), jobs.end(), compare);
vector<bool> slot(n, false);
int totalProfit = 0;
for (int i = 0; i < n; i++) {
for (int j = min(n, jobs[i].deadline) - 1; j >= 0; j--) {
if (!slot[j]) {
slot[j] = true;
totalProfit += jobs[i].profit;
break;
}
}
}
return totalProfit;
}
return 0;
```

Output

```
Total profit: 250
```

STUDENT NAME : Jagjeet Singh **ADMISSION NO. : 22scse1120009**

DATE: **EXPERIMENT NO. : 46**

AIM : Minimum Number of Coins

PROGRAM : Given different denominations of coins and an amount, findthe minimum number of coins needed to make up that amount.

Sourcecode:

```
#include <iostream>
#include <vector>
#include <climits>
using namespace std;
int minCoins(vector<int>& coins, int amount) {
vector<int> dp(amount + 1, INT_MAX);
dp[0] = 0;
for (int i = 1; i <= amount; i++) {
for (int coin : coins) {
if (i - coin >= 0 && dp[i - coin] != INT_MAX) {
dp[i] = min(dp[i], dp[i - coin] + 1);
}
}
}
return dp[amount] == INT_MAX ? -1 : dp[amount];
}
```

Output

```
Minimum coins required: 3
```

STUDENT NAME : Jagjeet Singh ADMISSION NO. : 22scse1120009

DATE: EXPERIMENT NO. : 47

AIM : **N-Queens Problem**

PROGRAM : Place N queens on an N×N chessboard so that no two queens threaten each other.

Sourcecode:

```
#include <iostream>
#include <vector>
using namespace std;
bool isSafe(int row, int col, vector<vector<int>>& board, int n) {
// Check the current column for any queen
for (int i = 0; i < col; i++) {
if (board[row][i] == 1) return false;
}
for (int i = row, j = col; i >= 0 && j >= 0; i--, j--) {
if (board[i][j] == 1) return false;
}
for (int i = row, j = col; i < n && j >= 0; i++, j--) {
if (board[i][j] == 1) return false;
}
return true;
}
bool solveNQueensUtil(vector<vector<int>>& board, int col, int n)
if (col >= n) return true;
for (int row = 0; row < n; row++) {
if (isSafe(row, col, board, n))
board[row][col] = 1;
if (solveNQueensUtil(board, col + 1, n)) return true;
// If placing queen in [row][col] doesn't lead to a solution, backtrack
board[row][col] = 0;
}
}
return false;
}
void printBoard(vector<vector<int>>& board, int n) {
for (int i = 0; i < n; i++) {
for (int j = 0; j < n; j++) {
cout << (board[i][j] == 1 ? "Q " : ". ");
}
```

```
cout << endl;
}
}
bool solveNQueens(int n) {
vector<vector<int>> board(n, vector<int>(n, 0)); // Create a board initialized with 0s (empty
cells)
if (solveNQueensUtil(board, 0, n)) {
printBoard(board, n);
return true;
}
cout << "Solution does not exist" << endl;
return false;
}
int main() {
int n;
cout << "Enter the number of queens: ";
cin >> n;
solveNQueens(n);
return 0;
}
```

Output

```
Q . . .
. . Q .
. Q . .
. . . Q
```

STUDENT NAME : Jagjeet Singh ADMISSION NO. : 22scse1120009

DATE: EXPERIMENT NO. : 48

AIM : Permutations:

PROGRAM : Generate all possible permutations of a given list of numbers or characters

Sourcecode:

```
public class Main {

  public static void generatePermutations(char[] arr, int index) {
    if (index == arr.length) {
      System.out.println(new String(arr));
      return;
    }

    for (int i = index; i < arr.length; i++) {
      swap(arr, index, i);
      generatePermutations(arr, index + 1);
      swap(arr, index, i);
    }
  }

  private static void swap(char[] arr, int i, int j) {
    char temp = arr[i];
    arr[i] = arr[j];
    arr[j] = temp;
  }

  public static void main(String[] args) {
    char[] arr = {'1', '2', '3'};
    generatePermutations(arr, 0);
  }
}
```

Output

```
123
132
213
231
312
321
```

STUDENT NAME : Jagjeet Singh ADMISSION NO. : 22scse1120009

DATE: EXPERIMENT NO. : 49

AIM : Subsets

PROGRAM : Generate all possible subsets of a given set of numbers

Sourcecode:

```
import java.util.*;

public class Main {

   public static void generateSubsets(int[] arr, int index, List<Integer> currentSubset) {
      System.out.println(currentSubset);

      for (int i = index; i < arr.length; i++) {
         currentSubset.add(arr[i]);
         generateSubsets(arr, i + 1, currentSubset);
         currentSubset.remove(currentSubset.size() - 1);
      }
   }
   public static void main(String[] args) {
      int[] arr = {1, 2, 3};
      generateSubsets(arr, 0, new ArrayList<>());
   }
}
```

Output

```
[]
[1]
[1, 2]
[1, 2, 3]
[1, 3]
[2]
[2, 3]
[3]
```