**Array**

Q1. Write a program to: Create a 1D array of integers. , Take input from the user. ,Display the elements of the array. Hint: Use a loop to input and print array elements.

```
#include <iostream>
using namespace std;
int main() {
 int n;
 cout << "Enter the number of elements: ";
 cin >> n;
 int arr[n];
 cout << "Enter " << n << " elements:\n";
 for (int i = 0; i < n; i++) {
 cin >> arr[i];
 }
 cout << "Array elements are: ";
 for (int i = 0; i < n; i++) {
 cout << arr[i] << " ";
 }
 return 0;
}
```

Q2. Write a program to find the sum and average of all elements in a 1D array.

Hint: Use a variable to accumulate the sum and then divide by the length.

```
#include <iostream>
using namespace std;
int main() {
 int n;
 cout << "Enter the number of elements: ";
 cin >> n;
 int arr[n];
 cout << "Enter " << n << " elements:" << endl;
 for(int i = 0; i < n; i++) {
 cin >> arr[i];
 }
 int sum = 0;
 for(int i = 0; i < n; i++) {
 sum += arr[i];
 }
 double average = static_cast<double>(sum) / n;
 cout << "Sum = " << sum << endl;
 cout << "Average = " << average << endl;
 return 0;
}
```

Q3.Write a program to search for a specific element in a 1D array using linear search.

Hint: Use a loop and a flag variable to check if the element is found.

```
#include <iostream>
using namespace std;
int main() {
 int n, key, flag = 0;
 cout << "Enter the number of elements: ";
 cin >> n;
 int arr[n];
 cout << "Enter " << n << " elements:" << endl;
 for(int i = 0; i < n; i++) {
 cin >> arr[i];
 }
 cout << "Enter the element to search: ";
 cin >> key;
 for(int i = 0; i < n; i++) {
 if(arr[i] == key) {
 flag = 1;
```

```
 cout << "Element found at index " << i << endl;
 break;
 }
 }
 if(flag == 0) {
 cout << "Element not found in the array." << endl;
 }
 return 0;
}
```

Q4. Write a program to find the maximum and minimum elements in a 1D array.
Hint: Initialize max and min with the first element and update during traversal.

```
#include <iostream>
using namespace std;
int main() {
 int n;
 cout << "Enter the number of elements: ";
 cin >> n;
 int arr[n];
 cout << "Enter " << n << " elements:" << endl;
 for(int i = 0; i < n; i++) {
 cin >> arr[i];
 }
 // Initialize max and min with the first element
 int max = arr[0];
 int min = arr[0];
 // Traverse the array to find max and min
 for(int i = 1; i < n; i++) {
 if(arr[i] > max)
 max = arr[i];
 if(arr[i] < min)
 min = arr[i];
 }
 cout << "Maximum element = " << max << endl;
 cout << "Minimum element = " << min << endl;
 return 0;
}
```

Q5.Write a menu-driven program to: Insert an element at a given position. Delete an element from a given position. Display the array after each operation . Hint: Use list slicing or manual shifting of elements.

```
#include <iostream>
using namespace std;
int main() {
 int arr[100];
 int n, choice, pos, element;
 cout << "Enter the number of elements : ";
 cin >> n;
 cout << "Enter " << n << " elements:" << endl;
 for(int i = 0; i < n; i++) {
 cin >> arr[i];
 }
 do {
 cout << "\n----- MENU -----\n";
 cout << "1. Insert element at position\n";
 cout << "2. Delete element from position\n";
 cout << "3. Display array\n";
 cout << "4. Exit\n";
 cout << "Enter your choice: ";
 cin >> choice;
 switch(choice) {
 case 1:
 if(n >= 100) {
```

```
 cout << "Array is full. Cannot insert.\n";
 break;
 }
 cout << "Enter position to insert (0 to " << n << "): ";
 cin >> pos;
 if(pos < 0 || pos > n) {
 cout << "Invalid position.\n";
 break;
 }
 cout << "Enter element to insert: ";
 cin >> element;
 for(int i = n; i > pos; i--) {
 arr[i] = arr[i - 1];
 }
 arr[pos] = element;
 n++;
 cout << "Element inserted successfully.\n";
 break;
 case 2:
 if(n == 0) {
 cout << "Array is empty. Cannot delete.\n";
 break;
 }
 cout << "Enter position to delete (0 to " << n - 1 << "): ";
 cin >> pos;
 if(pos < 0 || pos >= n) {
 cout << "Invalid position.\n";
 break;
 }
 for(int i = pos; i < n - 1; i++) {
 arr[i] = arr[i + 1];
 }
 n--;
 cout << "Element deleted successfully.\n";
 break;
 case 3:
 cout << "Current Array: ";
 for(int i = 0; i < n; i++) {
 cout << arr[i] << " ";
}
 cout << endl;
 break;
 case 4:
 cout << "Exiting program.\n";
 break;
 default:
 cout << "Invalid choice. Try again.\n";
 }
 } while(choice != 4);
 return 0;
}
```

Q6.Write a program to sort the array in ascending order.
Hint: Use Bubble Sort or Python’s built-in sort() method.

```
#include <iostream>
using namespace std;
int main() {
 int n;
 cout << "Enter the number of elements: ";
 cin >> n;
 int arr[n];
```

```
 cout << "Enter " << n << " elements:" << endl;
 for(int i = 0; i < n; i++) {
 cin >> arr[i];
 }
 // Bubble Sort algorithm
 for(int i = 0; i < n - 1; i++) {
 for(int j = 0; j < n - i - 1; j++) {
 if(arr[j] > arr[j + 1]) {
 // Swap
 int temp = arr[j];
 arr[j] = arr[j + 1];
 arr[j + 1] = temp;
 }
 }
 }
 // Display sorted array
 cout << "Sorted array in ascending order: ";
 for(int i = 0; i < n; i++) {
 cout << arr[i] << " ";
 }
 cout << endl;
 return 0;
}
```

Q7. Write a program to reverse the elements of a 1D array without using the built-in reverse() function.
Hint: Swap elements from both ends using two-pointer technique.

```
#include <iostream>
using namespace std;
int main() {
 int n;
 cout << "Enter the number of elements: ";
 cin >> n;
 int arr[n];
 cout << "Enter " << n << " elements:" << endl;
 for(int i = 0; i < n; i++) {
 cin >> arr[i];
 }
 // Reverse using two-pointer technique
 int start = 0, end = n - 1;
 while(start < end) {
 // Swap elements at start and end
 int temp = arr[start];
 arr[start] = arr[end];
 arr[end] = temp;
 start++;
 end--;
 }
 // Output the reversed array
 cout << "Reversed array: ";
 for(int i = 0; i < n; i++) {
 cout << arr[i] << " ";
 }
 cout << endl;
 return 0;
}
```

Q8.Write a program to: Create a 2D array (matrix). Take input from the user. Display it in matrix format. Hint: Use nested loops for rows and columns.

```
#include <iostream>
using namespace std;
int main() {
 int rows, cols;
```

```
 cout << "Enter number of rows: ";
 cin >> rows;
 cout << "Enter number of columns: ";
 cin >> cols;
 int matrix[rows][cols];
 cout << "Enter elements of the matrix:" << endl;
 for(int i = 0; i < rows; i++) {
 for(int j = 0; j < cols; j++) {
 cout << "Element at [" << i << "][" << j << "]: ";
 cin >> matrix[i][j];
 }
 }
 cout << "\nMatrix is:" << endl;
 for(int i = 0; i < rows; i++) {
 for(int j = 0; j < cols; j++) {
 cout << matrix[i][j] << "\t";
 }
 cout << endl;
 }
 return 0;
}
```

Q9. Write a program to perform addition of two matrices of the same order.
Hint: C[i][j] = A[i][j] + B[i][j]

```
#include <iostream>
using namespace std;
int main() {
 int rows, cols;
 cout << "Enter number of rows: ";
 cin >> rows;
 cout << "Enter number of columns: ";
 cin >> cols;
 int A[rows][cols], B[rows][cols], C[rows][cols];
 cout << "\nEnter elements of Matrix A:" << endl;
 for(int i = 0; i < rows; i++) {
 for(int j = 0; j < cols; j++) {
 cout << "A[" << i << "][" << j << "]: ";
 cin >> A[i][j];
 }
 }
 cout << "\nEnter elements of Matrix B:" << endl;
 for(int i = 0; i < rows; i++) {
 for(int j = 0; j < cols; j++) {
 cout << "B[" << i << "][" << j << "]: ";
 cin >> B[i][j];
 }
 }
 for(int i = 0; i < rows; i++) {
 for(int j = 0; j < cols; j++) {
 C[i][j] = A[i][j] + B[i][j];
 }
 }
 cout << "\nResultant Matrix C (A + B):" << endl;
 for(int i = 0; i < rows; i++) {
 for(int j = 0; j < cols; j++) {
 cout << C[i][j] << "\t";
 }
 cout << endl;
 }
 return 0;
}
```

Q10.Write a program to find the transpose of a given 2D matrix.
Hint: Swap matrix[i][j] with matrix[j][i].

```
#include <iostream>
using namespace std;
int main() {
 int rows, cols;
 cout << "Enter number of rows: ";
 cin >> rows;
 cout << "Enter number of columns: ";
 cin >> cols;
 int matrix[rows][cols];
 int transpose[cols][rows];
 cout << "Enter elements of the matrix:" << endl;
 for(int i = 0; i < rows; i++) {
 for(int j = 0; j < cols; j++) {
 cout << "Element at [" << i << "][" << j << "]: ";
 cin >> matrix[i][j];
 }
 }
 for(int i = 0; i < rows; i++) {
 for(int j = 0; j < cols; j++) {
 transpose[j][i] = matrix[i][j];
 }
 }
 cout << "\nOriginal Matrix:" << endl;
 for(int i = 0; i < rows; i++) {
 for(int j = 0; j < cols; j++) {
 cout << matrix[i][j] << "\t";
 }
 cout << endl;
 }
 cout << "\nTranspose of the Matrix:" << endl;
 for(int i = 0; i < cols; i++) {
 for(int j = 0; j < rows; j++) {
 cout << transpose[i][j] << "\t";
 }
 cout << endl;
 }
 return 0;
}
```

**Linked List**

Q1.Create a singly linked list from input and support push front, push back, delete value, print. Operations to Practice: Node creation, head/tail management, traversal, delete by value.

```
#include <iostream>
#include <string>
using namespace std;
struct Node
{
int data;
 Node* next;
 Node(intval)
 {
 data = val;
 next = NULL;
 }
};
class SinglyLinkedList{
 Node* head;
 Node* tail;
```

```
public:
SinglyLinkedList() {
 head = NULL;
 tail = NULL;
 }
 void push_front(intval) {
 Node* newNode = new Node(val);
 if (!head)
 {
 head = tail = newNode;
 }
 else {
newNode->next = head;
 head = newNode;
 }
 }
 void push_back(intval) {
 Node* newNode = new Node(val);
 if (!head)
 {
 head = tail = newNode;
 }
 else {
 tail->next = newNode;
 tail = newNode;
 }
}
void delete_value(intval) {
 if (!head) return;
 if (head->data == val)
 {
 Node* temp = head;
 head = head->next;
 if (temp == tail) tail = NULL; 77777
 delete temp;
 return;
 }
 Node* curr = head;
 while (curr->next &&curr->next->data != val)
curr = curr->next;
 if (curr->next)
 {
 Node* temp = curr->next;
curr->next = curr->next->next;
 if (temp == tail) tail = curr;
 delete temp;
 }
 }
 void print() {
 Node* curr = head;
 while (curr){
cout<<curr->data << " ";
curr = curr->next;
 }
cout<<endl;
 }
};
int main(){
int n;
cout<< "Enter number of operations: ";
```

```
cin>> n;
SinglyLinkedList list;
 string op;
int val;
 for (inti = 0; i< n; i++) {
cin>> op;
 if (op == "push_front")
 {
cin>>val;
list.push_front(val);
 }
 else if (op == "push_back") {
 cin>>val;
list.push_back(val);
 }
 else if (op == "delete_value") {
cin>>val;
list.delete_value(val);
 }
 else if (op == "print") {
list.print();
 }
 }
 return 0;
}
```

Q2.Given a list and integer k, reverse nodes in groups of k. Last group ¡ k remains as-is. Operations to Practice: Pointer reversal, group detection, reconnect sublists. Target: O(n) time, O(1) extra.

```
2. #include <bits/stdc++.h>
using namespace std;
struct Node {
int data;
Node* next;
Node(int x): data(x), next(NULL) {}
};
Node* reverseKGroup(Node* head, int k) {
Node* temp = head;
for (int i = 0; i < k; i++) {
if (!temp) return head;
temp = temp->next;
}
Node* prev = NULL;
Node* curr = head;
Node* next = NULL;
int count = 0;
while (curr && count < k) {
next = curr->next;
curr->next = prev;
prev = curr;
curr = next;
count++;
}
if (next) head->next = reverseKGroup(next, k);
return prev;
}
void printList(Node* head) {
while (head) {
cout << head->data << " ";
head = head->next;
}
}
```

```
int main() {
Node* head = new Node(1);
head->next = new Node(2);
head->next->next = new Node(3);
head->next->next->next = new Node(4);
head->next->next->next->next = new Node(5);
head->next->next->next->next->next = new Node(6);
head->next->next->next->next->next->next = new Node(7);
int k = 3;
head = reverseKGroup(head, k);
printList(head);
return 0;
}
```

Q3. Remove duplicate values from an unsorted singly list; keep first occurrence. Operations to Practice: Hashing or nested scan. Target: O(n) with hash; O(n2 )withoutextraspace.

```
. #include <bits/stdc++.h>
using namespace std;
struct Node {
int data;
Node* next;
Node(int x) : data(x), next(NULL) {}
};
void removeDuplicates(Node* head) {
unordered_set<int> seen;
Node* curr = head;
Node* prev = NULL;
while (curr) {
if (seen.count(curr->data)) {
prev->next = curr->next;
delete curr;
} else {
seen.insert(curr->data);
prev = curr;
}
curr = prev->next;
}
}
void printList(Node* head) {
while (head) {
cout << head->data << " ";
head = head->next;
}
}
int main() {
Node* head = new Node(4);
head->next = new Node(2);
head->next->next = new Node(4);
head->next->next->next = new Node(3);
head->next->next->next->next = new Node(2);
head->next->next->next->next->next = new Node(9);
removeDuplicates(head);
printList(head);
return 0;
}
```

Q4.: Detect if the list has a cycle. If yes, remove it and print linear list. Operations to Practice: Floyd’s tortoise-hare, meeting point, cycle length or entry node; pointer fix

```
. #include <bits/stdc++.h>
using namespace std;
struct Node {
```

```
int data;
Node* next;
Node(int x): data(x), next(NULL) {}
};
void removeCycle(Node* head) {
Node *slow = head, *fast = head;
bool hasCycle = false;
while (fast && fast->next) {
slow = slow->next;
fast = fast->next->next;
if (slow == fast) {
hasCycle = true;
break;
}
}
if (!hasCycle) return;
slow = head;
while (slow->next != fast->next) {
slow = slow->next;
fast = fast->next;
}
fast->next = NULL;
}
void printList(Node* head) {
while (head) {
cout << head->data << " ";
head = head->next;
}
}
int main() {
Node* head = new Node(1);
head->next = new Node(2);
head->next->next = new Node(3);
head->next->next->next = new Node(4);
head->next->next->next->next = new Node(5);
head->next->next->next->next->next = head->next->next;
removeCycle(head);
printList(head);
return 0;
}
```

Q5.Check whether a singly linked list is a palindrome. Restore list if modified. Operations to Practice: Middle via slow/fast, reverse second half, compare, optional restore.

```
. #include <bits/stdc++.h>
using namespace std;
struct Node {
int data;
Node* next;
Node(int x): data(x), next(NULL) {}
};
Node* reverseList(Node* head) {
Node* prev = NULL;
Node* curr = head;
while (curr) {
Node* nxt = curr->next;
curr->next = prev;
prev = curr;
curr = nxt;
}
return prev;
}
```

```
bool isPalindrome(Node* head) {
if (!head || !head->next) return true;
Node* slow = head;
Node* fast = head;
while (fast->next && fast->next->next) {
slow = slow->next;
fast = fast->next->next;
}
slow->next = reverseList(slow->next);
Node* first = head;
Node* second = slow->next;
bool result = true;
while (second) {
if (first->data != second->data) {
result = false;
break;
}
first = first->next;
second = second->next;
}
slow->next = reverseList(slow->next);
return result;
}
int main() {
Node* head = new Node(1);
head->next = new Node(2);
head->next->next = new Node(3);
head->next->next->next = new Node(2);
head->next->next->next->next = new Node(1);
if (isPalindrome(head))
cout << "YES";
else
cout << "NO";
return 0;
}
```

**Stack**

Q1.Given a string of ()[]{}, determine if it is balanced. Operations to Practice: Push on opening, pop on matching closing; check underflow and leftovers.

```
1#include <bits/stdc++.h>
using namespace std;
bool isBalanced(string s) {
stack<char> st;
unordered_map<char, char> match = {{')','('}, {']','['}, {'}','{'}};
for (char c : s) {
if (c == '(' || c == '[' || c == '{')
st.push(c);
else if (c == ')' || c == ']' || c == '}') {
if (st.empty() || st.top() != match[c])
return false;
st.pop();
}
}
return st.empty();
}
int main() {
string s1 = "{[()]}";
string s2 = "([)]";
cout << (isBalanced(s1) ? "YES" : "NO") << endl;
cout << (isBalanced(s2) ? "YES" : "NO") << endl;
```

```
return 0;
}
```

Q2.: For each element, find the next greater element to its right; if none, -1. Operations to Practice: Monotonic stack (indices). Target: O(n).

```
2.#include <iostream>
#include <stack>
#include <vector>
using namespace std;
vector<int>nextGreaterElement(const vector<int>&arr)
{
int n = arr.size();
 vector<int>nge(n, -1);
 stack<int>st;
 for (inti = 0; i< n; i++)
 {
 while (!st.empty() &&arr[i] >arr[st.top()]) {
nge[st.top()] = arr[i];
st.pop();
 }
st.push(i);
 }
 return nge;
}
int main() {
int n, val;
cout<< "Enter number of elements: ";
cin>> n;
 vector<int>arr(n);
cout<< "Enter elements: ";
 for (inti = 0; i< n; i++) {
 cin>>arr[i];
 }
 vector<int> result = nextGreaterElement(arr);
cout<< "Next Greater Elements: ";
 for (int x : result)
cout<< x << " ";
cout<<endl;
 return 0;
}
```

Q3.Evaluate a space-separated postfix expression with +, -, *, / and integers. Operations to Practice: Operand push, operator pop-2, compute, push result.

```
. #include <bits/stdc++.h>
using namespace std;
int evaluatePostfix(string exp) {
stack<int> st;
stringstream ss(exp);
string token;
while (ss >> token) {
if (isdigit(token[0])) {
st.push(stoi(token));
} else {
int val2 = st.top(); st.pop();
int val1 = st.top(); st.pop();
switch (token[0]) {
case '+': st.push(val1 + val2); break;
case '-': st.push(val1 - val2); break;
case '*': st.push(val1 * val2); break;
case '/': st.push(val1 / val2); break;
}
}
```

```
}
return st.top();
}
int main() {
string exp = "100 20 5 / + 3 *";
cout << evaluatePostfix(exp);
return 0;
}
```

Q4.: Convert an infix expression (with parentheses) to postfix. Operations to Practice: Operator stack with precedence/associativity; output queue/string.

```
#include <bits/stdc++.h>
using namespace std;
int precedence(char op) {
if (op == '^') return 3;
if (op == '*' || op == '/') return 2;
if (op == '+' || op == '-') return 1;
return 0;
}
string infixToPostfix(string exp) {
stack<char> st;
string result = "";
for (char c : exp) {
if (isalnum(c)) result += c;
else if (c == '(') st.push(c);
else if (c == ')') {
while (!st.empty() && st.top() != '(') {
result += st.top(); st.pop();
}
st.pop();
} else {
while (!st.empty() && precedence(st.top()) >= precedence(c)) {
result += st.top(); st.pop();
}
st.push(c);
}
}
while (!st.empty()) {
result += st.top();
st.pop();
}
return result;
}
int main() {
string exp = "A*(B+C)/D";
cout << infixToPostfix(exp);
return 0;
}
```

Q5.Implement a stack with two queues (push/pop/top/size). Operations to Practice: Two-queue method: costly push or costly pop variant.

```
5. #include <iostream>
#include <queue>
#include <string>
#include <sstream>
using namespace std;
class Stack
{
 queue<int> q1, q2;
public:
 void push(int x) {
```

```
 q2.push(x);
 while (!q1.empty())
 {
 q2.push(q1.front());
 q1.pop();
 }
 swap(q1, q2);
 }
 void pop() {
 if (q1.empty())
 {
cout<< "Stack is empty!" <<endl;
 return;
 }
 }
 void top() {
 if (q1.empty())
 {
cout<< "Stack is empty!" <<endl;
 return;
 }
cout<< q1.front() <<endl;
 }
int size() {
 return q1.size();
 }
};
int main() {
 Stack st;
 string command;
cout<< "Enter commands (push <num>, pop, top, exit):" <<endl;
 while (true)
 {
getline(cin, command);
 if (command.substr(0, 4) == "push")
 {
intnum;
stringstreamss(command.substr(5));
ss>>num;
st.push(num);
 }
 else if (command == "pop" ){
st.pop();
 }
 else if (command == "top"){
st.top();
 }
 else if (command == "exit") {
 break;
 }
 }
 return 0;
}
```

**<u>Queue</u>**

Q1.Implement a fixed-size circular queue with enqueue, dequeue, front, isFull, isEmpty. Operations to Practice: Array ring buffer; head/tail modulo arithmetic.

```
1.#include <iostream>
#include <string>
#include <sstream>
using namespace std;
```

```
class CircularQueue
{
int *arr;
int front, rear, size;
public:
CircularQueue(int s) {
 size = s;
arr = new int[size];
 front = rear = -1;
 }
boolisFull() {
 return ((rear + 1) % size == front);
 }
boolisEmpty() {
 return (front == -1);
 }
 void enqueue(int x) {
 if (isFull())
 {
cout<< "Queue is Full!" <<endl;
 return;
 }
 if (isEmpty())
 {
 front = rear = 0;
 }
 else {
 rear = (rear + 1) % size;
 }
arr[rear] = x;
 }
 void dequeue() {
 if (isEmpty())
 {
cout<< "Queue is Empty!" <<endl;
 return;
 }
 if (front == rear)
 {
 front = rear = -1;
 }
 else {
 front = (front + 1) % size;
 }
 }
 void getFront()
 {
 if (isEmpty())
 {
cout<< "Queue is Empty!" <<endl;
 }
 else {
cout<<arr[front] <<endl;
 }
 }
 void print()
 {
 if (isEmpty())
 {
cout<< "Queue is Empty!" <<endl;
```

```
 return;
 }
inti = front;
 while (true)
 {
cout<<arr[i];
 if (i == rear) break;
cout<< " ";
i = (i + 1) % size;
 }
cout<<endl;
 }
};
int main() {
int n;
cout<< "Enter queue size: ";
cin>> n;
cin.ignore();
CircularQueue q(n);
 string command;
cout<< "Enter commands (enqueue<num>, dequeue, front, isFull, isEmpty, print, exit):"
<<endl;
 while (true)
 {
getline(cin, command);
 if (command.substr(0, 7) == "enqueue")
 {
intnum;
stringstreamss(command.substr(8));
ss>>num;
q.enqueue(num);
 }
 else if (command == "dequeue"){
q.dequeue();
 }
 else if (command == "front") {

q.getFront();
 }
 else if (command == "isFull") {
 cout<< (q.isFull() ? "TRUE" : "FALSE") <<endl;
 }
 else if (command == "isEmpty") {
 cout<< (q.isEmpty() ? "TRUE" : "FALSE") <<endl;
 }
 else if (command == "print") {
 q.print();
 }
 else if (command == "exit") {
 break;
 }
 else if (command.empty()) {
 continue;
 }
 else {
 cout<< "Invalid command!" <<endl;
 }
 }
 return 0;
}
```

Q2.Given array and window size k, print max of each window. Operations to Practice: Monotonic deque storing indices. Target: O(n).

```
.#include <iostream>
#include <deque>
#include <vector>
using namespace std;
vector<int>slidingWindowMax(vector<int>&arr, int k)
{
deque<int>dq;
 vector<int> result;
 for (inti = 0; i<arr.size(); i++)
 {
 while (!dq.empty() &&dq.front() <= i - k)
dq.pop_front();
 while (!dq.empty() &&arr[dq.back()] <= arr[i])
dq.pop_back();
dq.push_back(i);
 if (i>= k - 1)
result.push_back(arr[dq.front()]);
 }
 return result;
}
int main()
{
int n, k;
cout<< "Enter number of elements: ";
cin>> n;
 vector<int>arr(n);
cout<< "Enter elements of array: ";
 for (inti = 0; i< n; i++)
cin>>arr[i];
cout<< "Enter window size k: ";
cin>> k;
 vector<int> res = slidingWindowMax(arr, k);
cout<< "Output: ";
 for (int x : res)
cout<< x << " ";
cout<<endl;
 return 0;
}
```

Q3.Implement a FIFO queue using two stacks. Operations to Practice: In-stack for enqueue, out-stack for dequeue; transfer lazily.

```
3.#include <iostream>
#include <stack>
#include <string>
#include <sstream>
using namespace std;
class Queue
{
 stack<int> s1, s2;
public:
 void enqueue(int x) {
 s1.push(x);
 }
 void dequeue() {
 if (s2.empty() && s1.empty())
 {
cout<< "Queue is empty!" <<endl;
 return;
```

```
 }
 if (s2.empty())
 {
 while (!s1.empty())
 {
 s2.push(s1.top());
 s1.pop();
 }
 }
cout<< s2.top() <<endl;
 s2.pop();
 }
 void front() {
 if (s2.empty() && s1.empty())
 {
cout<< "Queue is empty!" <<endl;
 return;
 }
 if (s2.empty())
 {
 while (!s1.empty())
 {
 s2.push(s1.top());
 s1.pop();
 }
 }
cout<< s2.top() <<endl;
 }
int size()
 {
 return s1.size() + s2.size();
 }
};
int main() {
 Queue q;
 string command;
cout<< "Enter commands (enqueue<num>, dequeue, front, size, exit):" <<endl;
 while (true) {
getline(cin, command);
 if (command.substr(0, 7) == "enqueue")

 string nums = command.substr(8);
stringstreamss(nums);
 string token;
 while (getline(ss, token, ','))
 {
int x = stoi(token);
q.enqueue(x);
 }
 }
 else if (command == "dequeue" {
 q.dequeue();
 }
 else if (command == "front") {
 q.front();
 }
 else if (command == "size") {
 cout<<q.size() <<endl;
 }
 else if (command == "exit") {
```

```
 break;
 }
 else if (command.empty()) {
 continue;
 }
 else {
 cout<< "Invalid command!" <<endl;
 }
 }
 return 0;
}
```

Q4.Simulate round-robin scheduling: each job has burst time; given quantum q, print completion order and turnaround times. Operations to Practice: Queue rotation, decrement remaining time, track finish time.

```
4.#include <iostream>
#include <queue>
#include <vector>
#include <string>
#include <sstream>
using namespace std;
struct Job
{
 string id;
intremainingTime;
intarrivalTime;
 Job(string i, int t) : id(i), remainingTime(t), arrivalTime(0) {}
};
int main() {
int n;
cout<< "Enter number of jobs: ";
cin>> n;
cin.ignore();
 vector<Job> jobs;
 string input;
cout<< "Enter jobs in format J1=5,J2=3,... : ";
getline(cin, input);
stringstreamss(input);
 string token;
 while (getline(ss, token, ','))
 {
size_tpos = token.find('=');
 string id = token.substr(0, pos);
int burst = stoi(token.substr(pos + 1));
jobs.push_back(Job(id, burst));
 }
int quantum;
cout<< "Enter time quantum q: ";
cin>> quantum;
 queue<Job> q;
 vector<string>finishOrder;
 vector<int>completionTime(n, 0);
 for (auto &job : jobs)
 {
q.push(job);
 }
int time = 0;
 while (!q.empty())
 {
 Job curr = q.front();
q.pop();
 if (curr.remainingTime<= quantum {
```

```
 time += curr.remainingTime;
curr.remainingTime = 0;
finishOrder.push_back(curr.id);
 } else {
 time += quantum;
curr.remainingTime -= quantum;
q.push(curr);
 }
 }
cout<< "Finish order: ";
 for (inti = 0; i<finishOrder.size(); i++)
 {
cout<<finishOrder[i];
 if (i != finishOrder.size() - 1) cout<< ", ";
 }
cout<<endl;
 return 0;
}
```

Q5.Given documents with priorities, simulate printing: at each step if any doc has higher priority than front, move front to back; else print it. Report when target index prints. Operations to Practice: Queue of (prio5.#include <iostream>

```
#include <queue>
#include <vector>
#include <set>
using namespace std;
struct Document
{
int priority;
int index;
 Document(int p, inti) : priority(p), index(i) {}
};
int main() {
int n, targetIndex;
cout<< "Enter number of documents: ";
cin>> n;
 vector<int> priorities(n);
cout<< "Enter document priorities: ";
 for (inti = 0; i< n; i++)
cin>> priorities[i];
cout<< "Enter target index: ";
cin>>targetIndex;
 queue<Document> q;
multiset<int>ms;
 for (inti = 0; i< n; i++) {
q.push(Document(priorities[i], i));
ms.insert(priorities[i]);
 }
intprintedCount = 0;
 while (!q.empty()) {
 Document curr = q.front();
q.pop();
intmaxPriority = *ms.rbegin();
 if (curr.priority<maxPriority)
 {
q.push(curr);
 }
 else{
printedCount++;
ms.erase(ms.find(curr.priority));
 if (curr.index == targetIndex)
```

```
 {
cout<<printedCount<<endl;
 break;
 }
 }
 }
 return 0;
}
```

**Trees**

Question 1: Binary Tree Traversals Problem Statement: Create a binary tree and perform the following traversals: 1. Inorder Traversal 2. Preorder Traversal 3. Postorder Traversal

```
#include <iostream>
using namespace std;
struct Node {
 char data;
 Node* left;
 Node* right;
 Node(char val) {
 data = val;
 left = right = nullptr;
 }
};
void inorder(Node* root) {
 if (root != nullptr) {
 inorder(root->left);
 cout << root->data << " ";
 inorder(root->right);
 }
}
void preorder(Node* root) {
 if (root != nullptr) {
 cout << root->data << " ";
 preorder(root->left);
 preorder(root->right);
 }
}
void postorder(Node* root) {
 if (root != nullptr) {
 postorder(root->left);
 postorder(root->right);
 cout << root->data << " ";
 }
}
int main() {
 Node* root = new Node('A');
 root->left = new Node('B');
 root->right = new Node('C');
 root->left->left = new Node('D');
 root->left->right = new Node('E');
 cout << "Inorder Traversal: ";
 inorder(root);
 cout << "\nPreorder Traversal: ";
 preorder(root);
 cout << "\nPostorder Traversal: ";
 postorder(root);
 return 0;
}
```

Question 2: Height of a Binary Tree Problem Statement: Write a program to find the height (or depth) of a binary tree. The height of a tree is the number of edges on the longest path from the root node to a leaf node. Hint: Use recursion to calculate height.

```
height(node) = 1 + max(height(node.left), height(node.right))
#include <iostream>
#include <algorithm>
using namespace std;
struct Node {
 char data;
 Node* left;
 Node* right;
 Node(char val) {
 data = val;
 left = right = nullptr;
 }
};
int height(Node* root) {
 if (root == nullptr)
 return 0;
 return 1 + max(height(root->left), height(root->right));
}
int main() {
 Node* root = new Node('A');
 root->left = new Node('B');
 root->right = new Node('C');
 root->left->left = new Node('D');
 root->left->right = new Node('E');
 cout << "Height of tree = " << height(root);
 return 0;
}
```

Question 3: Count Leaf and Non-Leaf Nodes Problem Statement: Construct a binary tree and write a program to count the number of leaf nodes and non-leaf nodes.

```
#include <iostream>
using namespace std;
struct Node {
 char data;
 Node* left;
 Node* right;
 Node(char val) {
 data = val;
 left = right = nullptr;
 }
};
// Function to count leaf nodes
int countLeaf(Node* root) {
 if (root == nullptr)
 return 0;
 if (root->left == nullptr && root->right == nullptr)
 return 1;
 return countLeaf(root->left) + countLeaf(root->right);
}
// Function to count non-leaf nodes
int countNonLeaf(Node* root) {
 if (root == nullptr || (root->left == nullptr && root->right == nullptr))
 return 0;
 return 1 + countNonLeaf(root->left) + countNonLeaf(root->right);
}
int main() {
 Node* root = new Node('A');
 root->left = new Node('B');
```

```
 root->right = new Node('C');
 root->left->left = new Node('D');
 root->left->right = new Node('E');
 cout << "Total Leaf Nodes = " << countLeaf(root) << endl;
 cout << "Total Non-Leaf Nodes = " << countNonLeaf(root);
 return 0;
}
```

Question 4: Expression Tree Construction and Traversals Problem Statement: Construct an Expression Tree for the postfix expression: $AB + CD - *$ and display its Inorder, Preorder, and Postorder traversals.

```
#include <iostream>
#include <stack>
using namespace std;
struct Node {
 char data;
 Node* left;
 Node* right;
 Node(char val) {
 data = val;
 left = right = nullptr;
 }
};
// Check if a character is an operator
bool isOperator(char c) {
 return (c == '+' || c == '-' || c == '*' || c == '/');
}
// Construct expression tree from postfix expression
Node* constructTree(string postfix) {
 stack<Node*> st;
 for (char ch : postfix) {
 if (!isOperator(ch)) {
 st.push(new Node(ch));
 } else {
 Node* node = new Node(ch);
 node->right = st.top(); st.pop();
 node->left = st.top(); st.pop();
 st.push(node);
 }
 }
 return st.top();
}
// Traversals
void inorder(Node* root) {
 if (root != nullptr) {
 if (isOperator(root->data)) cout << "(";
 inorder(root->left);
 cout << root->data;
 inorder(root->right);
 if (isOperator(root->data)) cout << ")";
 }

}
void preorder(Node* root) {
 if (root != nullptr) {
 cout << root->data;
 preorder(root->left);
 preorder(root->right);
 }
}
void postorder(Node* root) {
 if (root != nullptr) {
```

```
 postorder(root->left);
 postorder(root->right);
 cout << root->data;
 }
}
int main() {
 string postfix = "AB+CD-*";
 Node* root = constructTree(postfix);
 cout << "Inorder: ";
 inorder(root);
 cout << "\nPreorder: ";
 preorder(root);
 cout << "\nPostorder: ";
 postorder(root);
 return 0;
}
```

Question 5: Level Order Traversal (BreadthFirst Search) Problem Statement: Write a program to perform Level Order Traversal of a binary tree using a queue

```
#include <iostream>
#include <queue>
using namespace std;
// Define the structure of a tree node
struct Node {
 int data;
 Node* left;
 Node* right;
 Node(int val) {
 data = val;
 left = right = nullptr;
 }
};
// Function for Level Order Traversal
void levelOrderTraversal(Node* root) {
 if (root == nullptr)
 return;
 queue<Node*> q; // Create a queue
 q.push(root); // Enqueue root node
 while (!q.empty()) {
 Node* current = q.front();
 q.pop();
 cout << current->data << " ";
 // Enqueue left child
 if (current->left != nullptr)
 q.push(current->left);
 // Enqueue right child
 if (current->right != nullptr)
 q.push(current->right);
 }
}
int main() {
 // Create the example tree
 Node* root = new Node(1);
 root->left = new Node(2);
 root->right = new Node(3);
 root->left->left = new Node(4);
 root->left->right = new Node(5);
 root->right->right = new Node(6);
 cout << "Level Order Traversal: ";
 levelOrderTraversal(root);
 return 0;
```

```
}
```

**Graphs :**

Q1.Representation of Graph using Adjacency Matrix and List :Write a program to represent a graph using: 1. Adjacency Matrix 2. Adjacency List Display both representations for the given graph

```
#include <iostream>
#include <vector>
using namespace std;
class Graph {
 int V;
 vector<vector<int>> adjMatrix;
 vector<vector<int>> adjList;
 char names[4] = {'A','B','C','D'};
public:
 Graph(int v) {
 V = v;
 adjMatrix = vector<vector<int>>(V, vector<int>(V, 0));
 adjList = vector<vector<int>>(V);
 }
 void addEdge(int u, int v) {
 adjMatrix[u][v] = 1;
 adjMatrix[v][u] = 1;
 adjList[u].push_back(v);
 adjList[v].push_back(u);
 }
 void displayMatrix() {
 cout << "Adjacency Matrix:\n ";
 for(int i=0;i<V;i++) cout << names[i] << " ";
 cout << endl;
 for(int i=0;i<V;i++) {
 cout << names[i] << " ";
 for(int j=0;j<V;j++) {
 cout << adjMatrix[i][j] << " ";
 }
 cout << endl;
 }
 }
 void displayList() {
 cout << "\nAdjacency List:" << endl;
 for(int i=0;i<V;i++) {
 cout << names[i] << " -> ";
 for(int j : adjList[i]) {
 cout << names[j] << " ";
 }
 cout << endl;
 }
 }
};
int main() {
 Graph g(4);
 g.addEdge(0,1); // A-B
 g.addEdge(0,2); // A-C
 g.addEdge(1,3); // B-D
 g.addEdge(2,3); // C-D
 g.displayMatrix();
 g.displayList();
 return 0;
}
```

Question 2: Depth First Search (DFS) Traversal Problem Statement: Implement Depth First Search (DFS) traversal of a graph using recursion.

```
#include <iostream>
#include <vector>
using namespace std;
class Graph {
 int V;
 vector<vector<int>> adj;
 vector<bool> visited;
public:
 Graph(int v) {
 V = v;
 adj.resize(V + 1);
 visited.resize(V + 1, false);
 }
 void addEdge(int u, int v) {
 adj[u].push_back(v);
 adj[v].push_back(u); // Undirected graph
 }
 void DFS(int node) {
 visited[node] = true;
 cout << node << " ";
 for(int next : adj[node]) {
 if(!visited[next])
 DFS(next);
 }
 }
};
int main() {
 Graph g(5);
 // Given edges
 g.addEdge(1, 2);
 g.addEdge(1, 3);
 g.addEdge(2, 4);
 g.addEdge(3, 5);
 cout << "DFS Traversal: ";
 g.DFS(1); // Start DFS from vertex 1
 return 0;
}
```

Question 3: Breadth First Search (BFS) Traversal Problem Statement: Implement Breadth First Search (BFS) traversal of a graph using a queue.

```
#include <iostream>
#include <vector>
#include <queue>
using namespace std;
class Graph {
 int V;
 vector<vector<int>> adj;
 vector<bool> visited;
public:
 Graph(int v) {
 V = v;
 adj.resize(V + 1);
 visited.resize(V + 1, false);
 }
 void addEdge(int u, int v) {
 adj[u].push_back(v);
 adj[v].push_back(u); // Undirected Graph
 }
 void BFS(int start) {
 queue<int> q;
 visited[start] = true;
```

```
 q.push(start);
 while(!q.empty()) {
 int node = q.front();
 q.pop();
 cout << node << " ";
 for(int next : adj[node]) {
 if(!visited[next]) {
 visited[next] = true;
 q.push(next);
 }
 }
 }
 }
};
int main() {
 Graph g(5);
 // Given edges
 g.addEdge(1, 2);
 g.addEdge(1, 3);
 g.addEdge(2, 4);
 g.addEdge(3, 5);
 cout << "BFS Traversal: ";
 g.BFS(1); // Start BFS from node 1
 return 0;
}
```

Question 4: Minimum Spanning Tree using Kruskal’s Algorithm Problem Statement: Write a program to find the Minimum Spanning Tree (MST) of a connected weighted graph using Kruskal’s Algorithm.

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
class Edge {
public:
 int u, v, w;
};
int findParent(int node, vector<int> &parent) {
 if (parent[node] == node)
 return node;
 return parent[node] = findParent(parent[node], parent);
}
void unionSet(int a, int b, vector<int> &parent, vector<int> &rank) {
 a = findParent(a, parent);
 b = findParent(b, parent);
 if (rank[a] < rank[b]) parent[a] = b;
 else if (rank[a] > rank[b]) parent[b] = a;
 else {
 parent[b] = a;
 rank[a]++;
 }
}
int main() {
 vector<Edge> edges = {
 {0, 1, 1}, // A - B
 {1, 2, 3}, // B - C
 {0, 2, 2}, // A - C
 {2, 3, 4} // C - D
 };
 int V = 4;
 vector<int> parent(V), rank(V, 0);
 for (int i = 0; i < V; i++) parent[i] = i;
```

```
 sort(edges.begin(), edges.end(),
 [](Edge a, Edge b) { return a.w < b.w; });
 int totalCost = 0;
 cout << "Edges in MST:\n";
 for (auto e : edges) {
 if (findParent(e.u, parent) != findParent(e.v, parent)) {
 unionSet(e.u, e.v, parent, rank);
 totalCost += e.w;
 cout << char(e.u + 'A') << " - " << char(e.v + 'A')
 << " (" << e.w << ")\n";
 }
 }
 cout << "Total Cost: " << totalCost << endl;
 return 0;
}
```

Question 5: Shortest Path using Dijkstra’s Algorithm Problem Statement: Write a program to find the shortest path from a given source vertex to all other vertices using Dijkstra’s Algorithm.

```
#include <iostream>
#include <vector>
#include <climits>
using namespace std;
int main() {
 int V = 5;
 char name[] = {'A','B','C','D','E'};
 // Adjacency matrix with weights
 int graph[5][5] = {
 {0,2,4,0,0}, // A
 {2,0,1,7,0}, // B
 {4,1,0,0,3}, // C
 {0,7,0,0,1}, // D
 {0,0,3,1,0} // E
 };
 vector<int> dist(V, INT_MAX);
 vector<bool> visited(V, false);
 int start = 0; // A as source
 dist[start] = 0;
 for(int i = 0; i < V - 1; i++) {
 int u = -1;
 for(int j = 0; j < V; j++) {
 if(!visited[j] && (u == -1 || dist[j] < dist[u]))
 u = j;
 }
 visited[u] = true;
 for(int v = 0; v < V; v++) {
 if(graph[u][v] && !visited[v]) {
 dist[v] = min(dist[v], dist[u] + graph[u][v]);
 }
 }
 }
 cout << "Shortest distances from A:\n";
 for(int i = 1; i < V; i++) {
 cout << "A to " << name[i] << ": " << dist[i] << endl;
 }
 return 0;
}
```