1) Write a program to store the element inn 1-D array and perform operations like searching sorting and reversing the array

Reverse Array

```
#include<iostream>
#include<conio.h>
using namespace std;
int main()
{
    int arr[50], size, i, j, temp;
    cout<<"Enter Array Size: ";
    cin>>size;
    cout<<"Enter Array elements: ";
    for(i=0;i<size;i++)
    {
        cin>>arr[i];
    }
    j=i-1;
    i=0;
    while(i<j)
    {
            temp=arr[i];
            arr[i]=arr[j];
            arr[j]=temp;
            i++;
            j--;
    }
```

```
    cout<<"Now the Reverse of the Array is: \n";
    for(i=0; i<size;i++)
    {
      cout<<arr[i]<<" ";
    }
    getch();
}
```

2) Linear search in array

```
#include<iostream>
#include<conio.h>
using namespace std;
int main()
{
        int arr[10], i, num, n, c=0, pos;
        cout<<"Enter the array size: ";
        cin>>n;
        cout<<"Enter Array ELements: ";
        for(i=0;i<n;i++)
        {
                cin>>arr[i];
        }
        cout<<"Enter the number to be search: ";
        cin>>num;
        for(i=0;i<n;i++)
        {
                if(arr[i]==num)
                {
```

```
                    c=1;
                    pos=i+1;
                    break;
                }
        }
        if(c==0)
        {
                cout<<"Number not found...!!";
        }
        else
        {
                cout<<num<<" found at position "<<pos;
        }
getch();
}
```

3) Sort elements of Array in Ascending Order

```
#include<iostream>
#include<conio.h>
using namespace std;

int main()
{
   int i, a[10], temp, j;
   cout << "Enter any 10 numbers in an array: \n";
```

```
// You should loop from 0 to 9 to input 10 elements into the array.
for (i = 0; i < 10; i++)
{
   cin >> a[i];
}
cout << "\n Data before sorting:  ";
for (j = 0; j < 10; j++)
{
   cout << a[j] << " "; // Add a space to separate the numbers.
}
// You should loop only up to 9 in both loops to avoid going out of bounds.
for (i = 0; i < 9; i++)
{
   for (j = 0; j < 9 - i; j++) // Reduce the inner loop by 'i' iterations since the largest elements are already sorted.
   {
      if (a[j] > a[j + 1])
      {
         temp = a[j];
         a[j] = a[j + 1];
         a[j + 1] = temp;
      }
   }
}
cout << "\n Data after sorting: ";
for (j = 0; j < 10; j++)
{
```

```
    cout << a[j] << " "; // Add a space to separate the numbers.
  }
  getch();
  return 0; // Add a return statement to indicate successful program completion.
}
```

4) Read two arrays from the user and merge them and display the element sorted order

```
#include<iostream>
#include<conio.h>
using namespace std;

int main()
{
	int arr1[50], arr2[250], size1, size2, size, i, j,k, merge[100];
	cout<<"Enter Array 1 size";
	cin>>size1;
	cout<<"Enter Array 1 Elements: ";
	for(i=0;i<size1;i++)
	{
		cin>>arr1[i];
	}
	cout<<"Enter Array 2 Size";
	cin>>size2;
	cout<<"Enter Array 2 Elements: ";
	for(i=0;i<size2;i++)
```

```
	{
		cin>>arr2[i];
	}
	for(i=0;i<size1;i++)
	{
		merge[i]=arr1[i];
	}
	size=size1+size2;
	for(i=0, k=size1; k<size && i<size2; i++, k++)
	{
		merge[k]=arr2[i];
	}
	cout<<"Now the new array after merging is: \n";
	for(i=0;i<size;i++)
	{
		cout<<merge[i]<<" ";
	}
	getch();
}
```

5) Matrix Addition (add two matrices)

```
#include<iostream>
#include<conio.h>
using namespace std;
int main()
{
```

```
int mat1[3][3], mat2[3][3], i, j, mat3[3][3];
cout<<"Enter matrix 1 elements :";
for(i=0; i<3; i++)
{
        for(j=0; j<3; j++)
        {
           cin>>mat1[i][j];
        }
}
cout<<"Enter matrix 2 elements :";
for(i=0; i<3; i++)
{
        for(j=0;j<3;j++)
        {
                cin>>mat2[i][j];
        }
}
cout<<"Adding the two matrix to form the third matrix.....\n";
for(i=0;i<3;i++)
{
        for(j=0;j<3;j++)
        {
         mat3[i][j]=mat1[i][j] + mat2[i][j];
        }
}
cout<<"The two matrix addede successfully....!!";
```

```
        cout<<"The new matrix will be....\n";
        for(i=0;i<3;i++)
        {
                for(j=0;j<3;j++)
                {
                        cout<<mat3[i][j]<<" ";
                }
                cout<<"\n";
        }
        getch();
}
```

6) Matrix Subtraction

```
#include<iostream>
#include<conio.h>
using namespace std;
int main()
{
        int mat1[3][3], mat2[3][3], i, j, mat3[3][3];
        cout<<"Enter matrix 1 elements :";
        for(i=0; i<3; i++)
        {
                for(j=0; j<3; j++)
                {
                   cin>>mat1[i][j];
                }
```

```
}
cout<<"Enter matrix 2 elements :";
for(i=0; i<3; i++)
{
        for(j=0;j<3;j++)
        {
                cin>>mat2[i][j];
        }
}
cout<<"Subtracting the two matrix to form the third matrix.....\n";
for(i=0;i<3;i++)
{
        for(j=0;j<3;j++)
        {
                mat3[i][j]=mat1[i][j] - mat2[i][j];
        }
}
cout<<"\nThe two matrix subtracted successfully....!!";
cout<<"\nThe new matrix will be....\n";
for(i=0;i<3;i++)
{
        for(j=0;j<3;j++)
        {
                cout<<mat3[i][j]<<" ";
        }
        cout<<"\n";
```

```
        }
        getch();
}
```

7) Create a single linked list and display the node elements in reverse order

```
#include<iostream>
#include<conio.h>
using namespace std;
struct node
{
        int info;
        node *next;
}
*start, *newptr, *save, *ptr;

node *create_new_node(int);
void insert_at_beg(node *);
void display(node *);

int main()
{
        start = NULL;
        int inf;
        char ch='y';
        while(ch=='y'||ch=='Y')
```

```
        {
                cout<<"Enter Information for the new node: ";
                cin>>inf;
                cout<<"\n Creating new node!!Press any key to continue.";
                getch();
                newptr = create_new_node(inf);
                if(newptr != NULL)
                {
                        cout<<"\n\n New node created successfully...!!\n";
                        cout<<"Press any key to continue.";
                        getch();
                }
                else
                {
                        cout<<"\n Sorry cannot create new node!!!Aborting!!!";
                        cout<<"Press any key to exit";
                        getch();
                        exit(1);
                }
                cout<<"\n\n Now inserting this node at the beginning of the
list...\n";
                cout<<"\n Press any key to continue..\n";
                getch();
                insert_at_beg(newptr);
                cout<<"\n Node successfully inserted at the beginning of the list.
\n";
                cout<<"Now the list is: \n";
```

```
                display(start);
                cout<<"\n Want to enter more nodes?(y/n)...";
                cin>>ch;
        }
        getch();
}
node *create_new_node(int n)
{
        ptr = new node;
        ptr->info = n;
        ptr->next = NULL;
        return ptr;
}
void insert_at_beg(node *np)
{
        if(start==NULL)
        {
                start = np;
        }
        else
        {
                save = start;
                start = np;
                np->next = save;
        }
}
```

```
void display(node *np)
{
        while(np != NULL)
        {
                cout<<np->info<<" ->";
                np = np->next;
        }
        cout<<"!!\n";
}
```

8) Create double linked list and sort the elements in the linked list

```
#include<iostream>
#include<conio.h>
using namespace std;
int c = 0;
struct node {
   node* next, * prev;
   int data;
} * head = NULL, * tail = NULL, * p = NULL, * r = NULL, * np = NULL;
void create(int x) {
   np = new node;
   np->data = x;
   np->next = NULL;
   np->prev = NULL;
  if (c == 0) {
      tail = np;
```

```
    head = np;
    p = head;
    p->next = NULL;
    p->prev = NULL;
    c++;
}
else {
    p = head;
    r = p;
    if (np->data < p->data) {
        np->next = p;
        p->prev = np;
        np->prev = NULL;
        head = np;
        p = head;
        do {
            p = p->next;
        } while (p->next != NULL);
        tail = p;
    }
    else if (np->data > p->data) {
        while (p != NULL && np->data > p->data) {
            r = p;
            p = p->next;
            if (p == NULL) {
                r->next = np;
```

```
                np->prev = r;
                np->next = NULL;
                tail = np;
                break;
            }
            else if (np->data < p->data) {
                r->next = np;
                np->prev = r;
                np->next = p;
                p->prev = np;
                if (p->next != NULL) {
                    do {
                        p = p->next;
                    } while (p->next != NULL);
                    tail = p;
                    break;
                }
            }
        }
    }
}
void traverse_tail() {
    node* t = tail;
    while (t != NULL) {
        cout << t->data << "\t";
```

```
        t = t->prev;
    }
    cout << endl;
}

void traverse_head() {
    node* t = head;
    while (t != NULL) {
        cout << t->data << "\t";
        t = t->next;
    }
    cout << endl;
}
int main() {
    int i = 0, n, x, ch;
    cout << "Enter the no. of nodes \n";
    cin >> n;
    while (i < n) {
        cout << "Enter the data for node " << i + 1 << ": ";
        cin >> x;
        create(x);
        i++;
    }
    cout << "\nTraversing Doubly Linked List Head first \n";
    traverse_head();
    cout << "\nTraversing doubly Linked List tail first \n";
```

```
    traverse_tail();
    getch();
}
```

9) Infix to postfix notation

```
#include <iostream>
#include <stack>
#include <string>
#include <cctype>
using namespace std;
int getPrecedence(char op) {
    if (op == '+' || op == '-')
        return 1;
    if (op == '*' || op == '/')
        return 2;
    return 0;
}
string infixToPostfix(const string& infix) {
    stack<char> operators;
    string postfix = "";

    for (int i = 0; i < infix.length(); ++i) {
        char ch = infix[i];
        if (isalnum(ch)) {
            postfix += ch;
        } else if (ch == '(') {
```

```
            operators.push(ch);
        } else if (ch == ')') {
            while (!operators.empty() && operators.top() != '(') {
                postfix += operators.top();
                operators.pop();
            }
            if (!operators.empty() && operators.top() == '(') {
                operators.pop();
            }
        } else {
            while (!operators.empty() && getPrecedence(ch) <=
getPrecedence(operators.top())) {
                postfix += operators.top();
                operators.pop();
            }
            operators.push(ch);
        }
    }
    while (!operators.empty()) {
        postfix += operators.top();
        operators.pop();
    }
    return postfix;
}
string postfixToInfix(const string& postfix) {
    stack<string> operands;
    for (int i = 0; i < postfix.length(); ++i) {
```

```
        char ch = postfix[i];
        if (isalnum(ch)) {
            string operand(1, ch);
            operands.push(operand);
        } else {
            string operand2 = operands.top();
            operands.pop();
            string operand1 = operands.top();
            operands.pop();
            string result = "(" + operand1 + ch + operand2 + ")";
            operands.push(result);
        }
    }
    return operands.top();
}
int main() {
    string infixExpression = "A*(B+C)/D";
    string postfixExpression = infixToPostfix(infixExpression);
    string infixExpressionFromPostfix = postfixToInfix(postfixExpression);

    cout << "Infix to Postfix Conversion:" << endl;
    cout << "Infix Expression: " << infixExpression << endl;
    cout << "Postfix Expression: " << postfixExpression << endl;

    cout << "\nPostfix to Infix Conversion:" << endl;
    cout << "Postfix Expression: " << postfixExpression << endl;
```

```
    cout << "Infix Expression: " << infixExpressionFromPostfix << endl;
    return 0;
}
```

10) Bubble Sort

```
#include<iostream>
using namespace std;
int main()
{
        int a [50],n,i,j,temp;
        cout<<"Enter size of the array:";
        cin>>n;
        cout<<"Enter the array elements:";
        for(i=0;i<n;++i)
        cin>>a[i];
        for(i=1;i<n;++i)
        {
                for(j=0;j<(n-i);++j)
                if(a[j]>a[j+1])
                {
                        temp=a[j];
                        a[j]=a[j+1];
                        a[j+1]=temp;
                }
        }
        cout<<"Array after bubble sort:";
```

```
	for(i=0;i<n;++i)
	cout<<" "<<a[i];
	return 0;
}
```

11) Selection Sort

```
#include<iostream>
#include<conio.h>
using namespace std;
int main()
{
	int size,arr[50],i,j,temp;
	cout<<"Enter Array Size:";
	cin>>size;
	cout<<"Enter Array Elements:";
	for(i=0;i<size;i++)
	{
		cin>>arr[i];
	}
	cout<<"Sorting Array using selection sort...\n";
	for(i=0;i<size;i++)
	{
		for(j=i+1;j<size;j++)
		{
			if(arr[i]>arr[j])
			{
```

```
                        temp=arr[i];
                        arr[i]=arr[j];
                        arr[j]=temp;
                    }
                }
        }
        cout<<"Now the array after sorting is:\n";
        for(i=0;i<size;i++)
        {
                cout<<arr[i]<<" ";
        }
        getch();
}
```

12) Insertion Sort

```
#include<iostream>
using namespace std;
int main()
{
int size,arr[50],i,j,temp;
cout<<"Enter array size: ";
cin>>size;
cout<<"Enter array elements: ";
for(i=0;i<size;i++)
{
cin>>arr[i];
```

```
}
cout<<"Sorting array using insertion sort!\n";
for(i=0;i<size;i++)
{
temp=arr[i];
j=i-1;
while((temp<arr[j])&&(j>=0))
{
arr[j+1]=arr[j];
j=j-1;
}
arr[j+1]=temp;
}
cout<<"Now the array after sorting is: \n";
for(i=0;i<size;i++)
{
cout<<arr[i]<<" ";
}
return 0;
}
```

13) Write a program to implement merge sort

```
#include <iostream>
#include <vector>
// Merge two subarrays of arr[]
// First subarray is arr[l..m]
```

```
// Second subarray is arr[m+1..r]
void merge(std::vector<int>& arr, int l, int m, int r) {
    int n1 = m - l + 1;
    int n2 = r - m;
    // Create temporary arrays
    std::vector<int> L(n1);
    std::vector<int> R(n2);

    // Copy data to temporary arrays L[] and R[]
    for (int i = 0; i < n1; i++) {
        L[i] = arr[l + i];
    }
    for (int j = 0; j < n2; j++) {
        R[j] = arr[m + 1 + j];
    }
    // Merge the two arrays back into arr[l..r]
    int i = 0; // Initial index of first subarray
    int j = 0; // Initial index of second subarray
    int k = l; // Initial index of merged subarray

    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) {
            arr[k] = L[i];
            i++;
```

```
        } else {
            arr[k] = R[j];
            j++;
        }
        k++;
    }
    // Copy the remaining elements of L[], if any
    while (i < n1) {
        arr[k] = L[i];
        i++;
        k++;
    }
    // Copy the remaining elements of R[], if any
    while (j < n2) {
        arr[k] = R[j];
        j++;
        k++;
    }
}
// Main function to perform merge sort on an array arr[l..r]
void mergeSort(std::vector<int>& arr, int l, int r) {
    if (l < r) {
        // Same as (l+r)/2, but avoids overflow for large l and r
        int m = l + (r - l) / 2;
```

```
        // Sort first and second halves
        mergeSort(arr, l, m);
        mergeSort(arr, m + 1, r);
        // Merge the sorted halves
        merge(arr, l, m, r);
    }
}
int main() {
    std::vector<int> arr = {12, 11, 13, 5, 6, 7};
    std::cout << "Original array: ";
    for (int num : arr) {
        std::cout << num << " ";
    }
    std::cout << std::endl;
    int arrSize = arr.size();
    mergeSort(arr, 0, arrSize - 1);

    std::cout << "Sorted array: ";
    for (int num : arr) {
        std::cout << num << " ";
    }
    std::cout << std::endl;
    return 0;
}
```

13) Create the tree and display the elements , construct the binary tree

```
#include<iostream>
using namespace std;
class Node {
   int key;
   Node* left;
   Node* right;
public:
   Node() {
      key = -1;
      left = NULL;
      right = NULL;
   };
   void setKey(int aKey) {
      key = aKey;
   };
   void setLeft(Node* aLeft) {
      left = aLeft;
   };
   void setRight(Node* aRight) {
      right = aRight;
   };
   int Key() {
      return key;
```

```
    };
    Node* Left() {
        return left;
    };
    Node* Right() {
        return right;
    };
};
// Tree class
class Tree {
    Node* root;
public:
    Tree();
    ~Tree();
    Node* Root() {
        return root;
    };
    void addNode(int key);
    void inOrder(Node* n);
    void preOrder(Node* n);
    void postOrder(Node* n);

private:
    void addNode(int key, Node* leaf);
    void freeNode(Node* leaf);
};
```

```
// Constructor
Tree::Tree() {
   root = NULL;
}
// Destructor
Tree::~Tree() {
   freeNode(root);
}
// Free the node
void Tree::freeNode(Node* leaf) {
   if (leaf != NULL) {
      freeNode(leaf->Left());
      freeNode(leaf->Right());
      delete leaf;
   }
}
// Add a node
void Tree::addNode(int key) {
   if (root == NULL) {
      cout << "Add root node... " << key << endl;
      Node* n = new Node();
      n->setKey(key);
      root = n;
   } else {
      cout << "Add other node... " << key << endl;
      addNode(key, root);
```

```
    }
}
// Add a node (private)
void Tree::addNode(int key, Node* leaf) {
    if (key <= leaf->Key()) {
        if (leaf->Left() != NULL)
            addNode(key, leaf->Left());
        else {
            Node* n = new Node();
            n->setKey(key);
            leaf->setLeft(n);
        }
    } else {
        if (leaf->Right() != NULL)
            addNode(key, leaf->Right());
        else {
            Node* n = new Node();
            n->setKey(key);
            leaf->setRight(n);
        }
    }
}
// Print the tree in-order
// Traverse the left sub-tree, root, right sub-tree
void Tree::inOrder(Node* n) {
    if (n) {
```

```
        inOrder(n->Left());
        cout << n->Key() << " "; // Add a space here
        inOrder(n->Right());
    }
}
// Print the tree in-order
// Traverse the left sub-tree, root, right sub-tree
void Tree::preOrder(Node* n) {
    if (n) {
        cout << n->Key() << " "; // Add a space here
        preOrder(n->Left());
        preOrder(n->Right());
    }
}
// Print the tree post-order
// Traverse the left sub-tree, root, right sub-tree, root
void Tree::postOrder(Node* n) {
    if (n) {
        postOrder(n->Left());
        postOrder(n->Right());
        cout << n->Key() << " "; // Add a space here
    }
}
// Test main program
int main() {
    Tree* tree = new Tree();
```

```
    tree->addNode(30);
    tree->addNode(10);
    tree->addNode(20);
    tree->addNode(40);
    tree->addNode(50);
    cout << "In order traversal" << endl;
    tree->inOrder(tree->Root());
    cout << endl;
    cout << "Pre order traversal" << endl;
    tree->preOrder(tree->Root());
    cout << endl;
    cout << "Post order traversal" << endl;
    tree->postOrder(tree->Root());
    cout << endl;
    delete tree;
    return 0;
}
```

14) write a program to implement the collision technique

```
#include <iostream>
#include <list>
#include <iterator>
const int tableSize = 10;
class HashTable {
private:
    std::list<std::pair<int, int>> table[tableSize];
```

```
    // Hash function: simple modulo operation
    int hash(int key) {
        return key % tableSize;
    }
public:
    // Insert a key-value pair into the hash table using chaining
    void insert(int key, int value) {
        int index = hash(key);
        table[index].push_back(std::make_pair(key, value));
    }
    // Search for a key in the hash table
    bool search(int key) {
        int index = hash(key);
        for (const auto& pair : table[index]) {
            if (pair.first == key) {
                return true; // Key found
            }
        }
        return false; // Key not found
    }
    // Display the hash table
    void display() {
        std::cout << "Hash Table:" << std::endl;
        for (int i = 0; i < tableSize; i++) {
            std::cout << "[" << i << "] -> ";
```

```
            if (!table[i].empty()) {
                for (const auto& pair : table[i]) {
                    std::cout << "(" << pair.first << ", " << pair.second << ") ";
                }
            } else {
                std::cout << "Empty";
            }
            std::cout << std::endl;
        }
    }
};
int main() {
    HashTable ht;
    // Insert some key-value pairs into the hash table
    ht.insert(12, 120);
    ht.insert(22, 220);
    ht.insert(42, 420);
    ht.insert(7, 70);
    ht.insert(32, 320);
    ht.insert(17, 170);

    // Display the hash table
    ht.display();

    // Search for a key
    int keyToSearch = 42;
```

```
    if (ht.search(keyToSearch)) {
        std::cout << "Key " << keyToSearch << " found in the hash table." << std::endl;
    } else {
        std::cout << "Key " << keyToSearch << " not found in the hash table." << std::endl;
    }
    return 0;
}
```

15) Shortest path diagram

```
#include <iostream>
#include <vector>
#include <queue>
#include <limits>
const int INF = std::numeric_limits<int>::max(); // Infinity value for distances
class Graph {
public:
    int vertices;
    std::vector<std::vector<std::pair<int, int>>> adjList; // Adjacency list with (vertex, weight) pairs

    Graph(int V) : vertices(V), adjList(V) {}

    // Add an edge to the graph
    void addEdge(int u, int v, int weight) {
        adjList[u].push_back(std::make_pair(v, weight));
```

```
        adjList[v].push_back(std::make_pair(u, weight)); // For undirected graph
    }

    // Find the shortest path using Dijkstra's algorithm
    void shortestPath(int startVertex) {
        std::vector<int> distance(vertices, INF); // Initialize distances to infinity
        std::vector<bool> visited(vertices, false);
        distance[startVertex] = 0; // Distance to itself is 0
        // Priority queue to choose the vertex with the shortest distance
        std::priority_queue<std::pair<int, int>, std::vector<std::pair<int, int>>,
std::greater<std::pair<int, int>>> pq;
        pq.push(std::make_pair(0, startVertex));
        while (!pq.empty()) {
            int u = pq.top().second;
            pq.pop();
            if (visited[u]) continue;
            visited[u] = true;
            for (const auto& neighbor : adjList[u]) {
                int v = neighbor.first;
                int weight = neighbor.second;
                if (!visited[v] && distance[u] != INF && distance[u] + weight <
distance[v]) {
                    distance[v] = distance[u] + weight;
                    pq.push(std::make_pair(distance[v], v));
                }
            }
        }
```

```
        // Print the shortest distances from the startVertex to all vertices
        std::cout << "Shortest distances from vertex " << startVertex << ":\n";
        for (int i = 0; i < vertices; ++i) {
            std::cout << "Vertex " << i << ": " << distance[i] << "\n";
        }
    }
};

int main() {
    int V = 6; // Number of vertices
    Graph g(V);

    // Add edges and their weights
    g.addEdge(0, 1, 2);
    g.addEdge(0, 2, 4);
    g.addEdge(1, 2, 1);
    g.addEdge(1, 3, 7);
    g.addEdge(2, 4, 3);
    g.addEdge(3, 4, 1);
    g.addEdge(3, 5, 5);
    g.addEdge(4, 5, 2);

    int startVertex = 0; // Starting vertex for finding shortest paths
    g.shortestPath(startVertex);
    return 0;
}
```