1. Write a program to implement Bubble sort.

```
#include<iostream>
using namespace std;
int main()
{
	int i, j, temp;
	int a[5] = {2,5,1,10,9};
	for(i=0; i<5; i++)
	{
		cout<<a[i]<<" ";
	}
	for(i=0;i<5;i++){
		for(j=0;j<5;j++){
			if( a[j] < a[i])
			{
				temp = a[i];
				a[i] = a[j];
				a[j] = temp;
			}
		}
	}
	cout<<endl;
	cout<<"Descending Ordered list - ";
	for(i=0; i<5; i++)
	{
		cout<<a[i]<<" ";
	}
	for(i=0;i<5;i++){
		for(j=0;j<5;j++){
			if( a[j] > a[i])
			{
				temp = a[i];
				a[i] = a[j];
				a[j] = temp;
			}
		}
	}
	cout<<endl;
	cout<<"Ascending Ordered list - ";
	for(i=0; i<5; i++)
	{
		cout<<a[i]<<" ";
	}
	return 0;
}
```

Output :

Descending Ordered list - 10 9 5 2 1

Ascending Ordered list - 1 2 5 9 10

2. Write a program to implement Quick sort

```
#include <iostream>
using namespace std;

int partition(int arr[], int low, int high){
   int pivot = arr[high];
   int i = low -1;

   for (int j = low; j < high; j++){
      if (arr[j] <= pivot){
         i++;
         swap(arr[i], arr[j]);
      }
   }

   swap(arr[i + 1], arr[high]);
   return i + 1;
}

void quickSort(int arr[], int low, int high){
   if(low < high){
      int pivotIndex = partition(arr, low, high);

      quickSort(arr, low, pivotIndex -1);
      quickSort(arr, pivotIndex + 1, high);
   }
}

void printArray(int arr[], int size){
   for(int i = 0; i< size; i++){
```

```
        cout << arr[i] << " ";
    }
    cout << endl;
}

int main(){
    int arr[] = {64, 25, 91, 54, 69};
    int n = sizeof(arr) / sizeof(arr[0]);

    cout << "Unsorted array: ";
    printArray(arr, n);

    quickSort(arr, 0, n - 1);

    cout << "Sorted array: ";
    printArray(arr, n);

    return 0;
}
```

Output :

Unsorted array: 64 25 91 54 69

Sorted array: 25 54 64 69 91

3. Write a program to implement Selection sort.

```
#include <iostream>
using namespace std;

void selectionSort(int arr[], int n){
   for(int i = 0; i < n - 1; i++){

      int minIndex = i;

      for(int j = i + 1; j < n; j++){
         if (arr[j] < arr[minIndex]){
            minIndex = j;
         }
      }

      swap(arr[i], arr[minIndex]);
   }
}

void printArray(int arr[], int n){
   for(int i = 0; i < n; i++){
      cout << arr[i] << " ";
   }
   cout << endl;
}

int main(){
   int arr[] = {25, 66, 47, 98, 63};

   int size = sizeof(arr) / sizeof(arr[0]);
```


....

```
    cout << "Unsorted array: ";
    printArray(arr, size);

    selectionSort(arr, size);

    cout << "Sorted array: ";
    printArray(arr, size);

    return 0;
}
```

Output :

Unsorted array: 25 66 47 98 63

Sorted array: 25 47 63 66 98

4. Write a program to implement Insertion sort

```
#include <iostream>
using namespace std;

void insertionSort(int arr[], int n){
   for(int i = 1; i < n; i++){
      int key = arr[i];
      int j = i - 1;

      while(j >= 0 && arr[j] > key){
         arr[j + 1] = arr[j];
         j--;
      }

      arr[j + 1] = key;
   }
}

void printArray(int arr[], int n){
   for(int i = 0; i < n; i++){
      cout << arr[i] << " ";
   }
   cout << endl;
}

int main(){
   int arr[] = {56, 98, 59, 87, 64};
   int n = sizeof(arr) / sizeof(arr[0]);

   cout << "Unsorted array: ";
   printArray(arr, n);
```


....

```
    insertionSort(arr, n);



    cout << "Sorted array: ";
    printArray(arr, n);

    return 0;
}
```

Output :

Unsorted array: 56 98 59 87 64

Sorted array: 56 59 64 87 98

5. Write a program to implement Linear search.

```
#include<iostream>
using namespace std;
int main()
{
	int a[5] = {10,20,30,40,50};
	int i, item, flag;

	cout<<"enter the number to search - ";
	cin>>item;
	for(i=0;i<5;i++)
	{
		if(a[i]==item){
			flag=i+1;
			break;
		}
		else{
			flag = 0;
		}
	}
	if(flag!=0)
		cout<<"item found";
	else
		cout<<"item not found";
	return 0;
}
```

Output :

enter the number to search - 5

item not found

enter the number to search - 10

item found

……

6. Write a program to implement Binary search.

```
#include <iostream>
using namespace std;

int binarySearch(int arr[], int low, int high, int target){
   if(low <= high){
      int mid = low + (high - low) / 2;

      if(arr[mid] == target){
         return mid;
      }

      if(arr[mid] < target){
         return binarySearch(arr, mid + 1, high, target);
      }

      return binarySearch(arr, low, mid - 1, target);
   }


   return -1;
}

int main(){
   int arr[] = {11, 12, 22, 54, 78};
   int target = 22;
   int n = sizeof(arr) / sizeof(arr[0]);

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

   if(result != -1){
```

```
        cout << "Element " << target << " found at index " << result << endl;
    } else {
        cout << "Element " << target << " not found in the array" << endl;
    }

    return 0;

}
```

Output :

Element 22 found at index 2

7. Write a program to implement Stack operations: push, pop, display.

```
#include <iostream>
#include <stack>
using namespace std;

const int MAX_SIZE = 10;

class Stack {
  private:
    stack<int> st;
  public:
    bool isEmpty(){
      return st.empty();
    }

    bool isFull(){
      return st.size() == MAX_SIZE;
    }

    void push(int value){
      if(isFull()){
        cout << "Stack Overflow. Cannot push " << value << ". Stack is full." << endl;
      } else {
        st.push(value);
        cout << value << " pushed to the stack." << endl;
      }
    }

    void pop(){
      if(isEmpty()){
        cout << "Stack Underflow. Cannot pop from an empty stack." << endl;
```


....

```
        } else {
            cout << st.top() << " popped from the stack." << endl;
            st.pop();
        }
    }

    void display(){
        if(isEmpty()){
            cout << "Stack is empty." << endl;
        } else {
            cout << "Stack elements: ";
            stack<int> temp = st;
            while(!temp.empty()){
                cout << temp.top() << " ";
                temp.pop();
            }
            cout << endl;
        }
    }
};

int main(){
    Stack stack;

    stack.push(5);
    stack.push(10);
    stack.push(15);
    stack.push(20);

    cout << endl;
```

```
    stack.display();

    cout << endl;

    stack.pop();
    stack.display();

    cout << endl;

    stack.pop();
    stack.pop();
    stack.pop();
    stack.pop();

    return 0;
}
```

Output:

5 pushed to the stack.

10 pushed to the stack.

15 pushed to the stack.

20 pushed to the stack.

Stack elements: 20 15 10 5

20 popped from the stack.

Stack elements: 15 10 5

15 popped from the stack.

10 popped from the stack.

5 popped from the stack.

Stack Underflow. Cannot pop from an empty stack.

....

8. Write a program to implement Linear Queue operations: Insert, Delete, Display.

```
#include <iostream>
using namespace std;

const int MAX_SIZE = 5;

class Queue {
  private:
    int front, rear, arr[MAX_SIZE];
  public:
    Queue(){
      front = rear = -1;
    }

    bool isEmpty(){
      return front == -1 && rear == -1;
    }

    bool isFull(){
      return rear == MAX_SIZE -1;
    }

    void enqueue(int value){
      if(isFull()){
        cout << "Queue Overflow. Cannot enqueue " << value << ". Queue is full." << endl;
      } else {
        if(isEmpty()){
          front = 0;
        }
        arr[++rear] = value;
        cout << value << " enqueued to the queue." << endl;
```


....

```
        }
    }

    void dequeue(){
        if(isEmpty()){
            cout << "Queue Underflow. Cannot dequeue from an empty queue." << endl;
        } else {
            int dequeuedValue = arr[front++];
            cout << dequeuedValue << " dequeued from the queue." << endl;

            if(front > rear){
                front = rear = -1;
            }
        }
    }

    void display(){
        if(isEmpty()){
            cout << "Queue is empty." << endl;
        } else {
            cout << "Queue elements: ";
            for (int i = front; i <= rear; i++){
                cout << arr[i] << " ";
            }
            cout << endl;
        }
    }
};

int main(){
    Queue queue;
```

```
    queue.enqueue(5);
    queue.enqueue(10);
    queue.enqueue(15);
    queue.enqueue(20);

    cout << endl;

    queue.display();

    cout << endl;

    queue.dequeue();
    queue.display();

    cout << endl;

    queue.dequeue();
    queue.dequeue();
    queue.dequeue();
    queue.dequeue();
}
```

Output :

```
5 enqueued to the queue.
10 enqueued to the queue.
15 enqueued to the queue.
20 enqueued to the queue.

Queue elements: 5 10 15 20

5 dequeued from the queue.
```

Queue elements: 10 15 20

10 dequeued from the queue.

15 dequeued from the queue.

20 dequeued from the queue.

Queue Underflow. Cannot dequeue from an empty queue.

9. Write a program to implement singly linked list with operations. i)create ii) insert iii) delete

```
#include <iostream>
using namespace std;

// Node class represents a single node in the linked list
class Node {
public:
   int data;
   Node* next;

   // Constructor to initialize data and next pointer
   Node(int value) {
      data = value;
      next = nullptr;
   }
};

// LinkedList class represents the linked list and its operations
class LinkedList {
private:
   Node* head; // Pointer to the first node in the linked list

public:
   // Constructor to initialize head to nullptr
   LinkedList() {
      head = nullptr;
   }

   // Function to create a new linked list with a single node
   void create(int value) {
      head = new Node(value);
```

```
}

// Function to insert a new node at the end of the linked list
void insert(int value) {
    // If the list is empty, create a new node and set it as the head
    if (head == nullptr) {
        create(value);
    } else {
        // Traverse the list to find the last node
        Node* temp = head;
        while (temp->next != nullptr) {
            temp = temp->next;
        }

        // Create a new node and attach it to the last node's next pointer
        temp->next = new Node(value);
    }
}

// Function to delete a node with a given value from the linked list
void remove(int value) {
    // If the list is empty, nothing to delete
    if (head == nullptr) {
        cout << "Linked list is empty. Cannot delete from an empty list." << endl;
        return;
    }

    // If the node to delete is the head node
    if (head->data == value) {
        Node* temp = head;
        head = head->next;
```

```
        delete temp;
        cout << "Node with value " << value << " deleted from the list." << endl;
        return;
    }

    // Traverse the list to find the node before the node to delete
    Node* prev = head;
    while (prev->next != nullptr && prev->next->data != value) {
        prev = prev->next;
    }

    // If the node to delete is not found
    if (prev->next == nullptr) {
        cout << "Node with value " << value << " not found in the list." << endl;
        return;
    }

    // Delete the node and adjust the pointers
    Node* temp = prev->next;
    prev->next = temp->next;
    delete temp;
    cout << "Node with value " << value << " deleted from the list." << endl;
}

// Function to display the elements of the linked list
void display() {
    if (head == nullptr) {
        cout << "Linked list is empty." << endl;
    } else {
        Node* temp = head;
        cout << "Linked list elements: ";
```

```
        while (temp != nullptr) {
          cout << temp->data << " ";
          temp = temp->next;
        }
        cout << endl;
      }
    }
};

// Main function to test the LinkedList class
int main() {
  LinkedList linkedList;

  linkedList.create(5);
  linkedList.insert(10);
  linkedList.insert(15);
  linkedList.insert(20);

  cout << endl;

  linkedList.display();

  cout << endl;

  linkedList.remove(10);
  linkedList.display();

  cout << endl;

  linkedList.remove(22); // This will result in a message indicating that the node is not found
```

```
    return 0;
}
```