1.Write a recursive function to print numbers from 1 to N.

```
#include<bits/stdc++.h>
using namespace std;
void print1toN(int N) {
    if(N == 0) return;
    print1toN(N -1);
    cout << N << " ";
    }
int main() {
    int N = 5;
    print(N);
    return 0;
}
```

2.Write a recursive function to print number from N to 1.

```
#include<bits/stdc++.h>
using namespace std;
void printNto1(int N) {
    if(N == 0) return;
    printNto1(N -1);
    cout << N << " ";
    }
int main() {
    int N = 5;
    print(N);
    return 0;
}
```

3.Write a recursive function to find the sum of the first N natural numbers.

```
#include<bits/stdc++.h>
using namespace std;
int sumN( int N) {
    if(N == 0) return 0;
    return N + sumN(N -1);
}
```

```
int main() {
    int N = 5;
    cout << sumN(N) << endl;
    return 0;
}
```

4.Write a recursive function to compute the factorial of a given number.

```
#include<bits/stdc++.h>
using namespace std;
int factorial(int N) {
    if( N <= 1) return 1;
    return N * factorial(N - 1);
}
int main() {
    int N;
    cin >> N;
    cout << factorial(N) << endl;
    return 0;
}
```

5.Write a recursive function to compute a ^ b(a raised to the power b).

```
       #include<bits/stdc++.h>
using namespace std;
int power(int a, int b) {
    if(b == 0) return 1;
    return a * power(a, b - 1);
}
int main() {
    int a, b;
    cin >> a >> b;
    cout << power(a, b) << endl;
```

```
    return 0;
}
```

6.Write a recursive function to find the N th Fibonacci number.

```
#include<bits/stdc++.h>
using namespace std;
int fib(int N) {
    if(N <= 1) return N;
    return fib(N -1 ) + fib(N - 2);
}
int main() {
    int N;
    cin >> N;
    cout << fib(N) << endl;
    return 0;
}
```

7.Write a recursive function to find the sum of the digits of a number.

```
#include<bits/stdc++.h>
using namespace std;
int sum(int N) {
    if(N == 0) return 0;
    return N % 10 + sum(N / 10);
}
int main() {
    int N;
    cin >> N;
    cout << sum(N) << endl;
    return 0;
}
```

8.Write a recursive function to check weather a string is a palindrome.

```
#include<bits/stdc++.h>
using namespace std;
palindrome(string & str, int start, int end) {
    if(start >= end) return true;
    if(str[start] != str[end]) return false;
    return palindrome(str, start + 1, end - 1);
}
int main() {
    string str;
    cin >> str;
    if(palindrome(str, 0, str.length() - 1)) {
        cout << "Palindrome" << endl;
    } else {
        cout << "Not Palindrome" << endl;
    }
    return 0;
}
```

9.Write a recursive fuction to reverse a string.

```
#include<bits/stdc++.h>
using namespace std;
void reversestring(string & str, int start, int end) {
    if(start >= end) return;
    swap(str[start], str[end]);
    reversestring(str, start + 1, end - 1);
}
int main() {
    string str;
    cin >> str;
    reversestring(str, 0, str.length() - 1);
    cout << str << endl;
```

```
    return 0;
}
```

10.Write a recursive function to count the number of digits in a given integer.

```
#include<bits/stdc++.h>
using namespace std;
int count1(int N) {
    if(N == 0) return 0;
    return 1 + count1(N / 10);
}
int main() {
    int N;
    cin >> N;
    cout << count(N) << endl;
    return 0;
}
```

11.Write a recursive function to count the number of digits in a given integer.

```
#include<bits/stdc++.h>
using namespace std;
int count2(int N) {
    if(N == 0) return 0;
    return 1 + count2(N / 10);
}
int main() {
    int N;
    cin >> N;
    cout << count(N) << endl;
    return 0;
}
```

12.Write a recursive function to check if a number is a prime.

```
#include<bits/stdc++.h>
using namespace std;
prime(int N, int i = 2) {
    if (N <= 2) return (N == 2);
    if (N % i == 0) return false;
    if (i * i > N ) return true;
    return prime(N, i + 1);
}
int main() {
    int N;
    cin >> N;
    cout << (prime(N) ? "Prime" : "Not Prime") << endl;
    return 0;
}
```

13.Write a recursive function to find the sum of all elements in an integer array.

```
#include<bits/stdc++.h>
using namespace std;
sumarray(int arr[], int N) {
    if(N == 0) return 0;
    return arr[N - 1] + sumarray(arr, N - 1);
}
int main() {
   int arr[] = {1, 2, 3, 4, 5};
   int N = sizeof(arr) / sizeof(arr[0]);
   cout << "Sum: " << sumarray(arr, N) << endl;
   return 0;
}
```

14.Write a recursive function to find the maximum element in an array.

```
#include<bits/stdc++.h>
using namespace std;
maxarray(int arr[], int N) {
    if(N == 1) return arr[0];
    return max(arr[N - 1], maxarray(arr, N - 1));
```

```
}
int main() {
   int arr[] = {1, 2, 3, 4, 5};
   int N = sizeof(arr) / sizeof(arr[0]);
   cout << "Max: " << maxarray(arr, N) << endl;
   return 0;
}
```

15.Write a c program implement a singly linked list with operations : Insert at the beginning,insert at a specefic position,insert at the end,and display.

```
#include <bits/stdc++.h>
using namespace std;
struct Node {
    int data;
    Node* next;
};

Node* root = nullptr;

void insertAtBeginning(int value) {
    Node* newNode = new Node();
    newNode->data = value;
    newNode->next = root;
    root = newNode;
}

void insertAtEnd(int value) {
    Node* newNode = new Node();
    newNode->data = value;
    newNode->next = nullptr;

    if (root == nullptr) {
        root = newNode;
    } else {
        Node* temp = root;
        while (temp->next != nullptr)
            temp = temp->next;
        temp->next = newNode;
    }
```

```
}

void insertAtPosition(int value, int pos) {
    if (pos == 1) {
        insertAtBeginning(value);
        return;
    }

    Node* newNode = new Node();
    newNode->data = value;

    Node* temp = root;
    for (int i = 1; i < pos - 1 && temp != nullptr; i++)
        temp = temp->next;

    if (temp == nullptr) {
        cout << "Invalid position\n";
        delete newNode;
    } else {
        newNode->next = temp->next;
        temp->next = newNode;
    }
}

void display() {
    Node* temp = root;
    while (temp != nullptr) {
        cout << temp->data << " -> ";
        temp = temp->next;
    }
    cout << "NULL\n";
}
int main() {
    int choice, value, pos;

    while (true) {
        cout << "1. Insert at Beginning\n";
        cout << "2. Insert at End\n";
        cout << "3. Insert at Position\n";
        cout << "4. Display List\n";
```

```
        cout << "5. Exit\n";
        cin >> choice;

        switch (choice) {
        case 1:
            cout << "Enter value: ";
            cin >> value;
            insertAtBeginning(value);
            break;
        case 2:
            cout << "Enter value: ";
            cin >> value;
            insertAtEnd(value);
            break;
        case 3:
            cout << "Enter value and position: ";
            cin >> value >> pos;
            insertAtPosition(value, pos);
            break;
        case 4:
            display();
            break;
        case 5:
            exit(0);
        default:
            cout << "Invalid choice\n";
        }
    }
      return 0;
}
```

16 .Write functions to delete a node from the beginning,end and a specific position in a singly linked list.

```
#include <bits/stdc++.h>
using namespace std;
void deleteFromBeginning() {
    if (root == nullptr) {
        cout << "List is empty.\n";
```

```
        return;
    }
    Node* temp = root;
    root = root->next;
    delete temp;
}

void deleteFromEnd() {
    if (root == nullptr) {
        cout << "List is empty.\n";
        return;
    }

    if (root->next == nullptr) {
        delete root;
        root = nullptr;
        return;
    }

    Node* temp = root;
    while (temp->next->next != nullptr)
        temp = temp->next;

    delete temp->next;
    temp->next = nullptr;
}

void deleteFromPosition(int pos) {
    if (pos == 1) {
        deleteFromBeginning();
        return;
    }

    Node* temp = root;
    for (int i = 1; i < pos - 1 && temp != nullptr; i++)
        temp = temp->next;

    if (temp == nullptr || temp->next == nullptr) {
        cout << "Invalid position.\n";
        return;
```

```
    }

    Node* toDelete = temp->next;
    temp->next = toDelete->next;
    delete toDelete;
}
int main() {
    addNode(10);
    addNode(20);
    addNode(30);
    display();

    deleteFromBeginning();
    display();

    deleteFromEnd();
    display();

    deleteFromPosition(1);
    display();

    return 0;
}
```

17.Write a function to search for an element in a singly linked list.

```
#include<bits/stdc++.h>
using namespace std;
void search(int key) {
    Node* temp = root;
    int pos = 1;
    while (temp != NULL) {
        if (temp->data == key) {
            cout << "Element found at position: " << pos <<
endl;
            return;
        }
        else if (key < temp->data) {
```

```
            temp = temp->left;
        }
        else {
            temp = temp->right;
        }
        pos++;
    }
    cout << "Element not found." << endl;
}
int main() {
    struct Node {
        int data;
        Node* left;
        Node* right;
    };
    int key = 5;
    search(key);
    return 0;
}
```

18.Write a program to count the number of nodes in a singly list.

```
#include<bits/stdc++.h>
using namespace std;
struct Node {
    int data;
    Node* next;
};
Node* root = nullptr;
void addNode(int data) {
    Node* newNode = new Node();
    newNode->data = data;
    newNode->next = nullptr;

    if (root == nullptr) {
        root = newNode;
    } else {
        Node* temp = root;
```

```
        while (temp->next != nullptr) {
            temp = temp->next;
        }
        temp->next = newNode;
    }
}
int countNodes() {
    int count = 0;
    Node* temp = root;
    while (temp != nullptr) {
        count++;
        temp = temp->next;
    }
    return count;
}
void displayList() {
    Node* temp = root;
    if (temp == nullptr) {
        cout << "List is empty.\n";
        return;
    }

    cout << "Linked List: ";
    while (temp != nullptr) {
        cout << temp->data << " -> ";
        temp = temp->next;
    }
    cout << "NULL\n";
}
int main() {
    // Adding some nodes
    addNode(10);
    addNode(20);
    addNode(30);

    // Display list
    displayList();

    // Count and display number of nodes
    cout << "Number of nodes: " << countNodes() << endl;
```

```
    return 0;
}
```

19.Implement a doubly linked list with insert a beginning,end and display operations.

```
#include<bits/stdc++.h>
using namespace std;
struct Node{
    int data;
    Node* next;
    Node* prev;
};
Node* root = nullptr;
void add_node(int data){
    Node* new_node = new Node();
    new_node -> data = data;
    new_node -> next = nullptr;
    new_node -> prev = nullptr;
    if(root == nullptr)
        root = new_node;
    else{
        Node* current_node = root;
        while(current_node -> next != nullptr){
            current_node = current_node -> next;
        }
        current_node -> next = new_node;
        new_node->prev = current_node;
    }
}
void print_forward()
{
    Node* current_node = root;
    while(current_node != nullptr){
        cout << current_node -> data << " ";
        current_node = current_node -> next;
    }
    cout << endl;
```

```
}

void search_node(int data)
{
    Node* current_node =  root;
    int index = 0;
    while(current_node != nullptr)
    {
        if(current_node -> data == data){
            printf("Found at %d\n", index);
            return;
        }
        current_node = current_node->next;
        index++;
    }
    printf("Not Found\n");
}
void insert_node(int index, int data)
{
    Node* new_node = new Node();
    new_node -> data = data;
    new_node -> next = nullptr;
    new_node -> prev = nullptr;
    if(index == 0)
    {
        new_node -> next = root;
        if(root != nullptr)
            root -> prev = new_node;
        root =  new_node;
        return;
    }
    Node* current_node = root;
    for(int i = 0; i < index - 1 && current_node != nullptr;
i++){
        current_node = current_node->next;
    }
    if(current_node == nullptr)
    {
        printf("Index out of bound\n");
        delete new_node;
```

```
        return;
    }
    new_node -> next = current_node->next;
    new_node -> prev = current_node;
    if(current_node->next != nullptr){
        current_node->next->prev = new_node;
    }
    current_node->next = new_node;
}
void delete_node(int index)
{
    if(root == nullptr){
        printf("List is empty\n");
        return;
    }
    if(index == 0)
    {
        Node* temp = root;
        root = root->next;
        if(root != nullptr)
            root->prev = nullptr;
        delete temp;
        return;
    }
    Node* current_node = root;
    for(int i = 0; i < index - 1 && current_node != nullptr;
i++){
        current_node = current_node->next;
    }
    if(current_node == nullptr || current_node->next == nullptr)
    {
        printf("Index out of bound\n");
        return;
    }
    if(current_node -> prev != nullptr)
        current_node -> prev -> next = current_node -> next;
    if(current_node->next != nullptr)
        current_node -> next -> prev = current_node -> prev;

    delete current_node;
```

```
}
void print_backward(Node* current_node)
{
    if(current_node != nullptr)
    {
      cout << current_node -> data << " ";
      print_backward(current_node -> next);

    }
}

int main()
{
    int arr[] = {1, 2, 3, 4, 5};
    int n = 5;
    for(int i = 0; i < n; i++)
        add_node(arr[i]);
    insert_node(2, 10);
    print_forward();
    return 0;
}
```

20.Implement a stack using array with push,pop, and display operations.

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

#define MAX 100
int myStack[MAX];
int top = -1;

int isEmpty() {
    if(top == -1) return 1;
    else return 0;
}

int isFull() {
    return top == MAX - 1;
```

```
}

void push(int value) {
    if (isFull()) {
        printf("Stack Overflow\n");
        return;
    }
    myStack[++top] = value;
}

void pop() {
    if (isEmpty()) {
        printf("Stack Underflow\n");
        return;
    }
    top--;
}

int peek() {
    if (isEmpty()) {
        printf("Stack is empty\n");
        return -1;
    }
    return myStack[top];
}

void display() {
    if (isEmpty()) {
        printf("Stack is empty\n");
        return;
    }
    for (int i = top; i >= 0; i--) {
        printf("%d\n", myStack[i]);
    }
}

int main() {
    push(5);
    push(6);
    push(10);
```

```
    pop();
    display();

    return 0;
}
```

21.Write a c program to demonstrate stack overflow and underflow conditions.

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

#define SIZE 100
int stack[SIZE];
int top = -1;

void pushWithCheck(int val) {
    if (top == SIZE - 1) {
        cout << "Overflow\n";
        return;
    }
    stack[++top] = val;
}

void popWithCheck() {
    if (top == -1) {
        cout << "Underflow\n";
        return;
    }
    --top;
}
int main() {
    int n, val;
    cout << "Enter the number of elements to push: ";
    cin >> n;

    for (int i = 0; i < n; ++i) {
        cout << "Enter value to push: ";
        cin >> val;
```

```
        pushWithCheck(val);
    }

    cout << "Popping elements:\n";
    for (int i = 0; i < n; ++i) {
        popWithCheck();
    }
```

22.Implement a queue using array with enqueue,dedqueue and display.

```
#include<bits/stdc++.h>
using namespace std;
#define MAX 100
int myQueue[MAX];
int front = -1, rear = -1;

int isEmpty() {
    return front == -1 || front > rear;
}

int isFull() {
    return rear == MAX - 1;
}

void enqueue(int value) {
    if (isFull()) {
        printf("Queue Overflow\n");
        return;
    }
    if (isEmpty()) {
        front = 0;
    }
    myQueue[++rear] = value;
}

void dequeue() {
    if (isEmpty()) {
        printf("Queue Underflow\n");
        return;
    }
```

```
    front++;
}

int peek() {
    if (isEmpty()) {
        printf("Queue is empty\n");
        return -1;
    }
    return myQueue[front];
}

void display() {
    if (isEmpty()) {
        printf("Queue is empty\n");
        return;
    }
    for (int i = front; i <= rear; i++) {
        printf("%d\n", myQueue[i]);
    }
}

int main() {
    enqueue(5);
    enqueue(6);
    enqueue(10);
    dequeue();
    display();

    return 0;
}
```

23.Write a c program to sort an array using insertion sort.

```
#include<bits/stdc++.h>
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;
    }
}

int main() {
    int arr[] = {12, 11, 13, 5, 6};
    int n = sizeof(arr) / sizeof(arr[0]);

    insertionSort(arr, n);

    std::cout << "Sorted array: \n";
    for (int i = 0; i < n; i++)
        std::cout << arr[i] << " ";
    std::cout << std::endl;

    return 0;
}
```

24.Write c program to sort an array using boubble sort.

```
#include<bits/stdc++.h>
using namespace std;
void bubbleSort(int arr[], int n) {
    for (int i = 0; i < n-1; ++i)
    {
        int swapped = 0;
        for(int j = 0; j < n-1 - i; j++)
        {
            if(arr[j+1] < arr[j])
            {
                swap(arr[j+1], arr[j]);
                swapped = 1;
            }
        }
        if(swapped == 0)
```

```
            break;
    }

    for (int i = 0; i < n; ++i)
    {
        printf("%d ", arr[i]);
    }
}

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;
    }
    for (int i = 0; i < n; ++i)
    {
        printf("%d ", arr[i]);
    }
}

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[minIndex] > arr[j])
                minIndex = j;
        }
        int temp = arr[minIndex];
        arr[minIndex] = arr[i];
```

```
        arr[i] =temp;
    }
    for (int i = 0; i < n; ++i)
    {
        printf("%d ", arr[i]);
    }
}
int main()
{
    int arr[] = {5, 3, 8, 4, 2};
    int size  = sizeof(arr)/sizeof(arr[0]);
    selectionSort(arr, size);
    return 0;
}
```

25.Modify the buuble sort algorithm to detect already sotrted arrays .

```
#include <bits/stdc++.h>
using namespace std;
#include <algorithm>
void optimizedBubbleSort(int arr[], int n) {
    bool swapped;
    for (int i = 0; i < n - 1; ++i) {
        swapped = false;
        for (int j = 0; j < n - i - 1; ++j) {
            if (arr[j] > arr[j + 1]) {
                swap(arr[j], arr[j + 1]);
                swapped = true;
            }
        }
        if (!swapped) break;
    }
}

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

```
    cout << endl;
}

int main() {
    int arr[] = {64, 34, 25, 12, 22, 11, 90};
    int n = sizeof(arr) / sizeof(arr[0]);

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

    optimizedBubbleSort(arr, n);

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

    return 0;
}
```

26.Write a program to sort an array using selection sort.

```
#include<bits/stdc++.h>
using namespace std;
#include<algorithm>
void selectionSort(int arr[], int n) {
    for (int i = 0; i < n - 1; ++i) {
        int minIdx = i;
        for (int j = i + 1; j < n; ++j) {
            if (arr[j] < arr[minIdx]) {
                minIdx = j;
            }
        }
        std::swap(arr[i], arr[minIdx]);
    }
}

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

```
    cout << endl;
}

int main() {
    int arr[] = {64, 25, 12, 22, 11, 90};
    int n = sizeof(arr) / sizeof(arr[0]);

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

    selectionSort(arr, n);

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

    return 0;
}
```

27.Write a c program to implement binary search in a sorted array.

```
#include<bits/stdc++.h>
using namespace std;
int binarySearch(int arr[], int target, int n)
{
    int low = 0, high = n;
    while(low <= high)
    {
        int mid = (low + high) / 2;
        if(arr[mid] == target)
            return mid;
        else if (arr[mid] > target)
            high = mid - 1;
        else if (arr[mid] < target)
            low = mid + 1;
    }
    return -1;
}
```

```
int main()
{
    int n, target;
    scanf("%d", &n);
    int arr[n];
    for(int i = 0; i < n; i++)
        cin >> arr[i];
    cin >> target;
    sort(arr, arr + n);
    printf("Sorted array: \n");
    for(int i = 0; i < n; i++)
        printf("%d ", arr[i]);
    cout << endl;
    int idx = binarySearch(arr, target, n);
    if(idx >= 0)
        printf("%d is at index %d\n", target, idx);
    else
        printf("Not Found!\n");

    return 0;
}
```