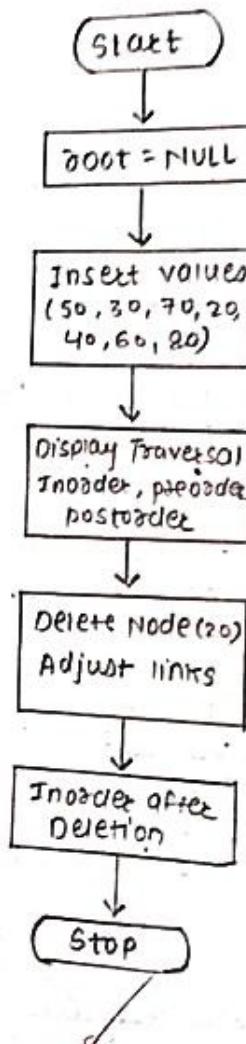


- Insert fixed values - 50, 30, 70, 20, 40, 60, 20
- Display Inorder, preorder and postorder
- Delete node 20.
- Display Inorder traversal again
- g. Stop

conclusion: Q P  
 The Binary Search tree was successfully implemented with insertion, deletion, and traversal operations.  
 It maintains elements in sorted order and allows faster searching with an average time complexity of  $O(\log n)$

#### • Flowchart



Binary search Tree with  
Insertion, deletion and traversal

The stack was successfully implemented using both array and linked list.

Both implementations perform push and

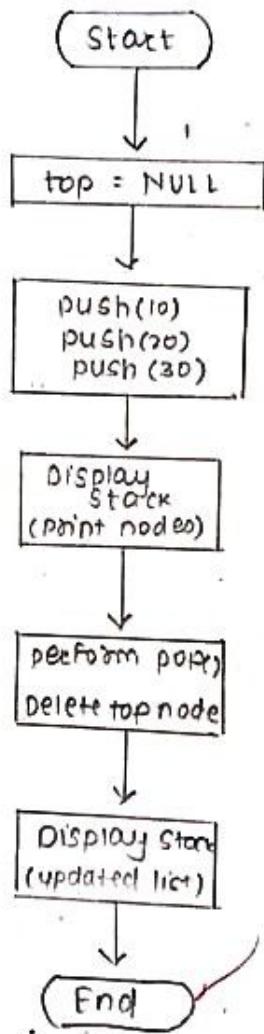
pop operations in constant time  $O(1)$ .

While the array is simple and faster,

the linked list is more flexible because

it does not have a fixed size limit.

### Algorithm



15 / 50

### Stack Using Linked List

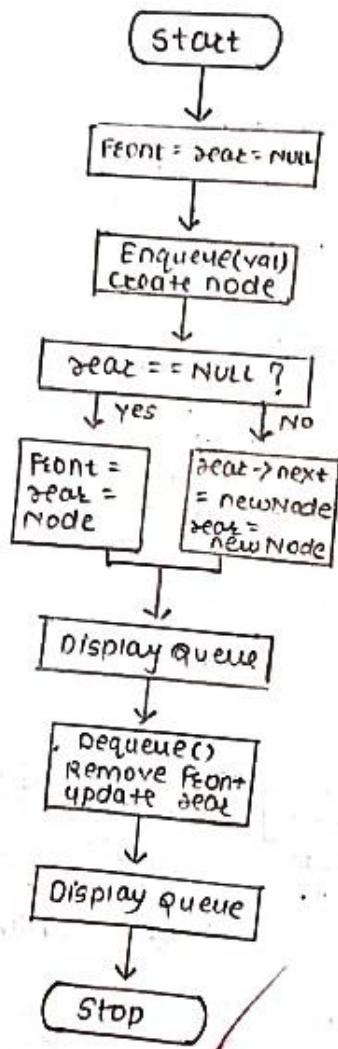
#### b) Stack using Linked List

Program :

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

```
struct Node {
    int data;
    Node* next;
};
```

• flowchart



Queue Using Linked List

b) Queue using Linked List

Program :

```

#include <iostream>
using namespace std;

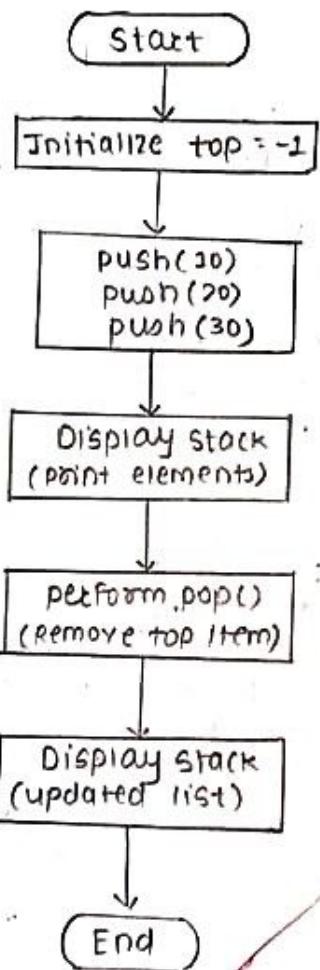
struct Node {
    int data;
    Node* next;
};

class Queue {
    Node* front;
    Node* rear;
public:
    Queue() { front = rear = NULL; }

    void enqueue(int val) {
        Node* newNode = new Node();
        newNode->data = val;
        if (front == NULL) {
            front = newNode;
            rear = newNode;
        } else {
            rear->next = newNode;
            rear = newNode;
        }
    }
}

```

## • Flowchart



Stack Using Array

### Experiment No. 6

Title: Implement Stack using a) Arrays and b) Linked List.

#### a) Stack using Array

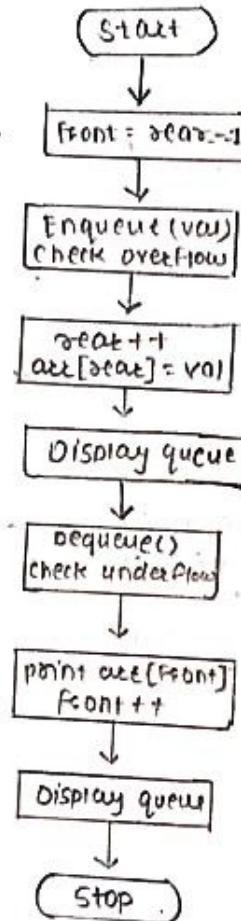
Program :

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

#define MAX 5

class Stack {
    int arr[MAX];
    int top;
public:
    Stack() { top = -1; }
```

• flowchart



✓  
Queue Using Arrays

**Experiment No. 7**

**Title:** Implement Queue using a) Arrays and b) Linked List.

(a) Queue using Array

**Program :**

```

#include <iostream>
using namespace std;

#define MAX 5

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

    void enqueue(int val) {
        if(rear == MAX - 1)
            cout << "Queue is full" << endl;
        else
            arr[++rear] = val;
    }

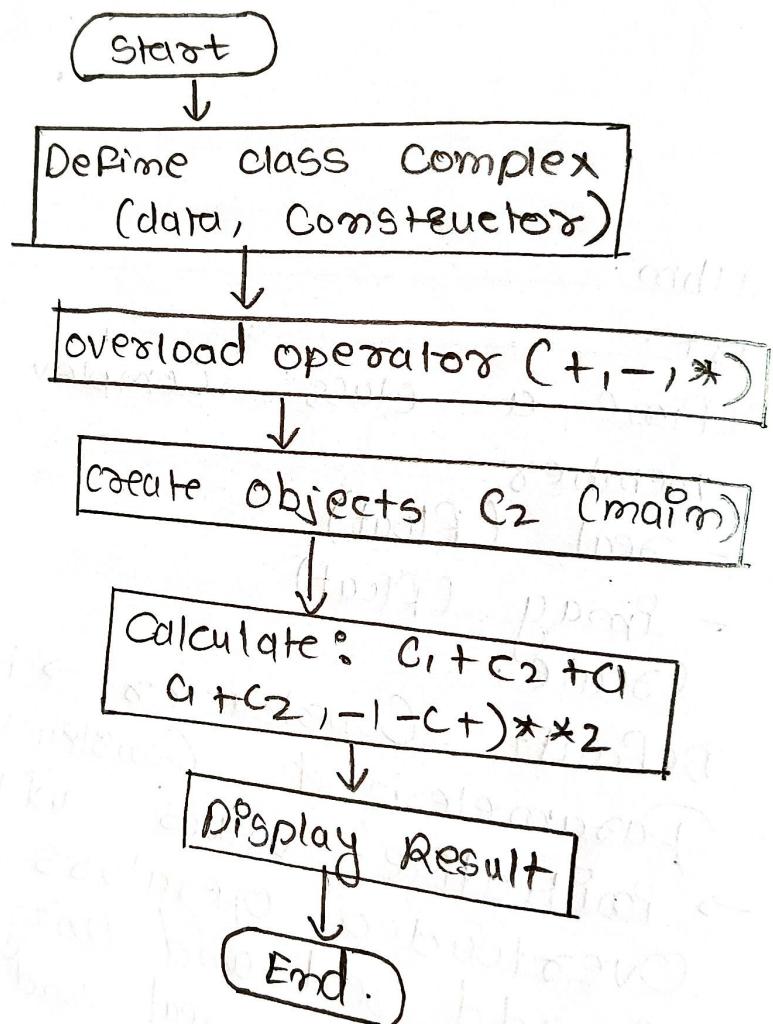
    int dequeue() {
        if(front == rear)
            cout << "Queue is empty" << endl;
        else
            return arr[++front];
    }

    void display() {
        for(int i=front+1; i<=rear; i++)
            cout << arr[i] << " ";
    }
};
  
```

1e)

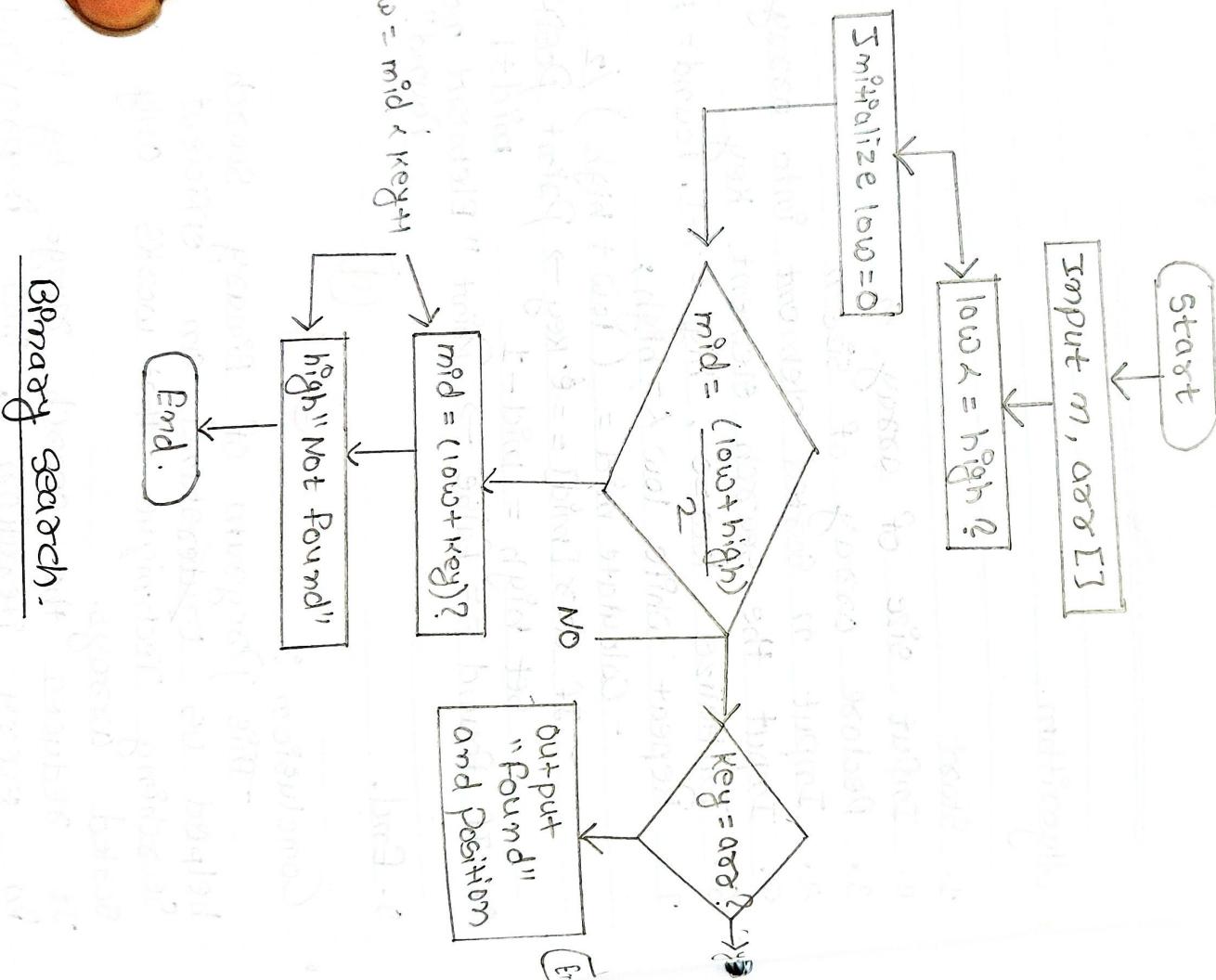
## Flowchart

## operator overloading



Conclusion:

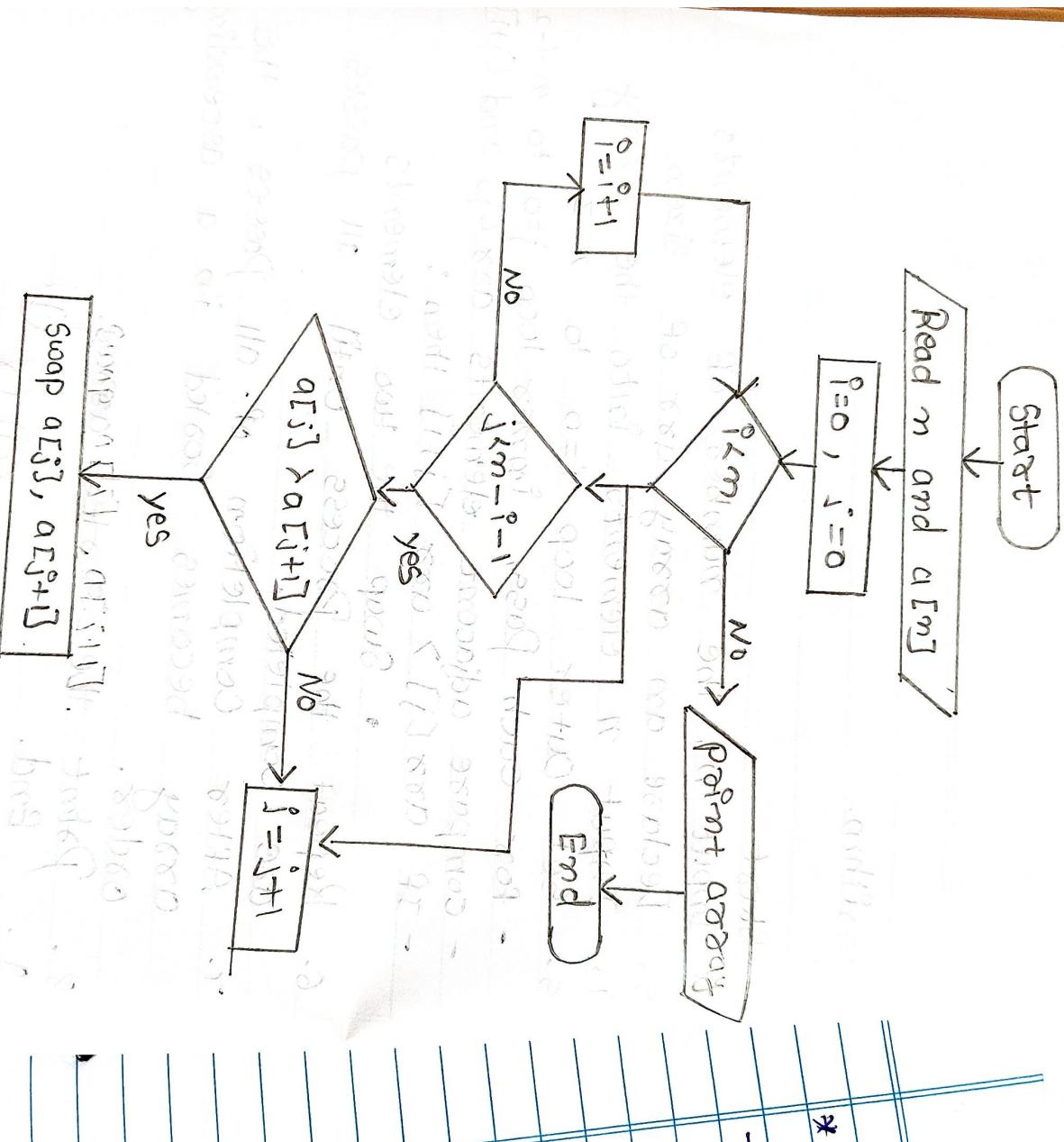
- Flowchart.



## Binary Search:

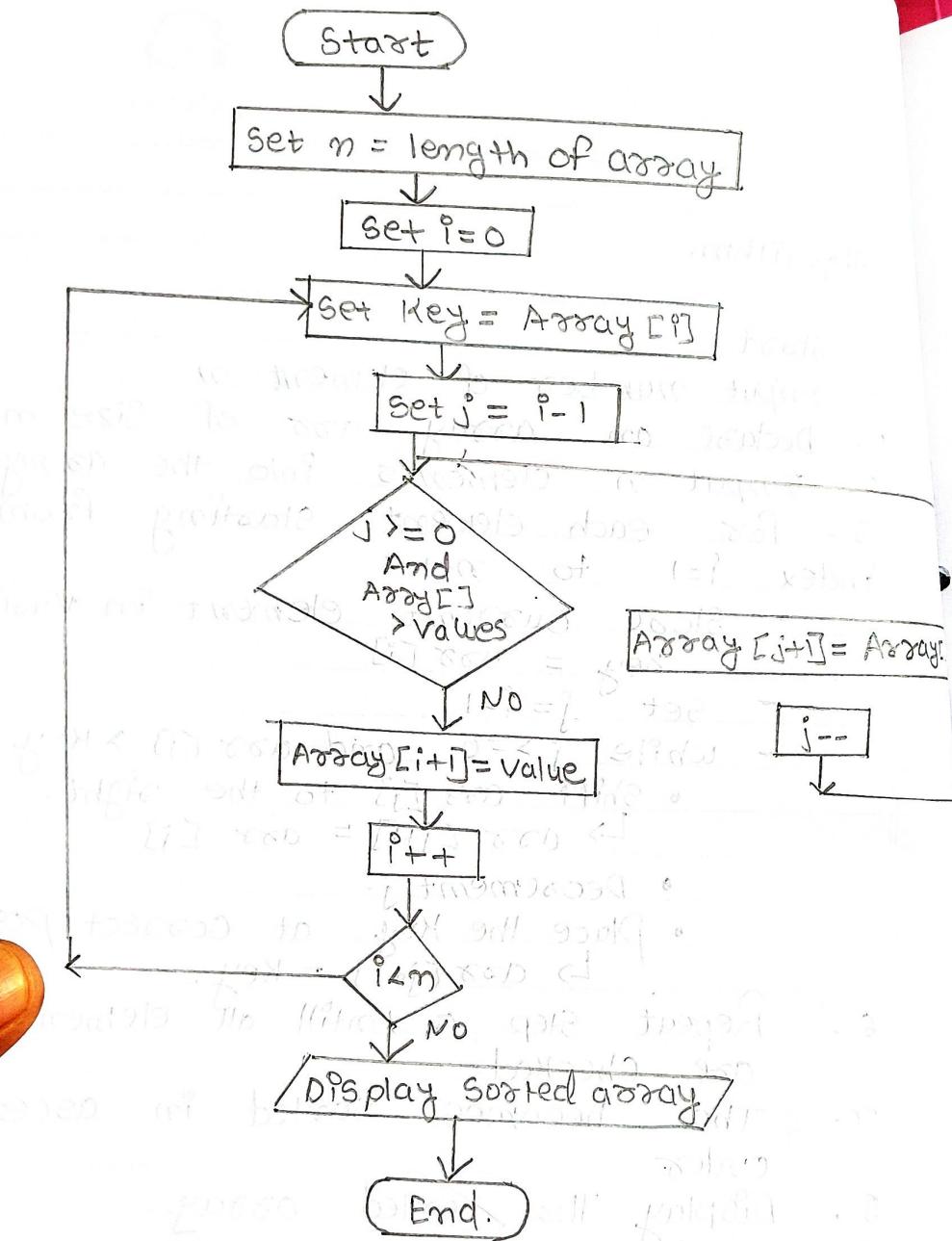
• Flowchart.

Bubble Sort .



After each iteration, the largest element is moved to its position at the end of the array. This continues until all elements are sorted. The final sorted array is printed.

## Flowchart :-



## Insertion Sort

Title:  
To Implement th

**Program:**

a) Bubble Sort  
#include <iostream.h>  
using namespace std;

void bubbleSort(int arr[], int n){  
 for (int i = 0; i < n - 1; i++) {  
 for (int j = 0; j < n - i - 1; j++) {  
 if (arr[j] > arr[j + 1]) {  
 swap(arr[j], arr[j + 1]);  
 }  
 }  
 }  
}

int main(){  
 int n;  
 cout << "Enter size of array: ";  
 cin >> n;  
 int arr[n];

cout << "Enter elements: ";  
for (int i = 0; i < n; i++) {  
 cin >> arr[i];  
}

bub

cor

for

}

cc

re

}

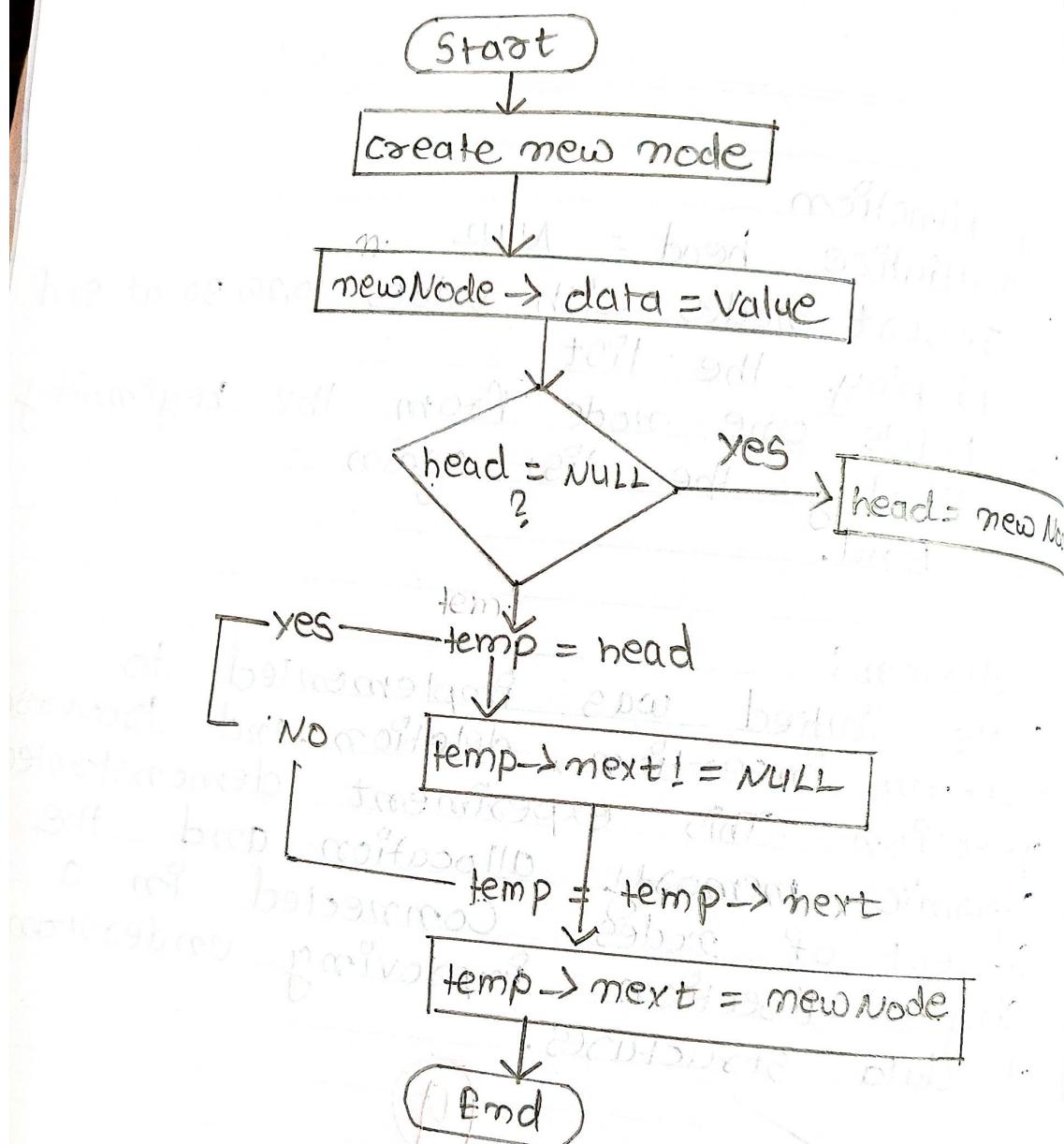
b)

#ir

us

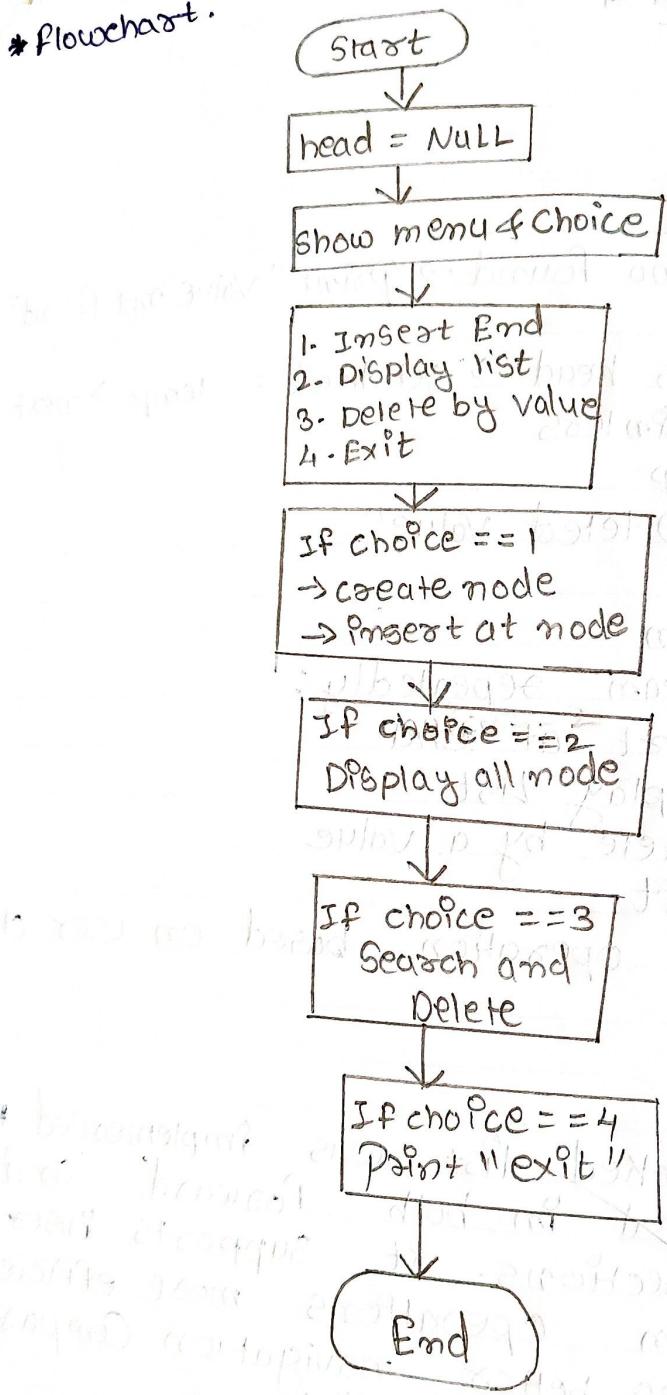
v

• Flowchart.



Single link List

\* Flowchart.



Double linked list

6. Doubly  
linked list

**Title:**  
To implement Doubly linked li

**Program:**

```

#include <iostream>
using namespace std;

struct Node {
    int data; // data part
    Node *prev, *next; // pointer
};

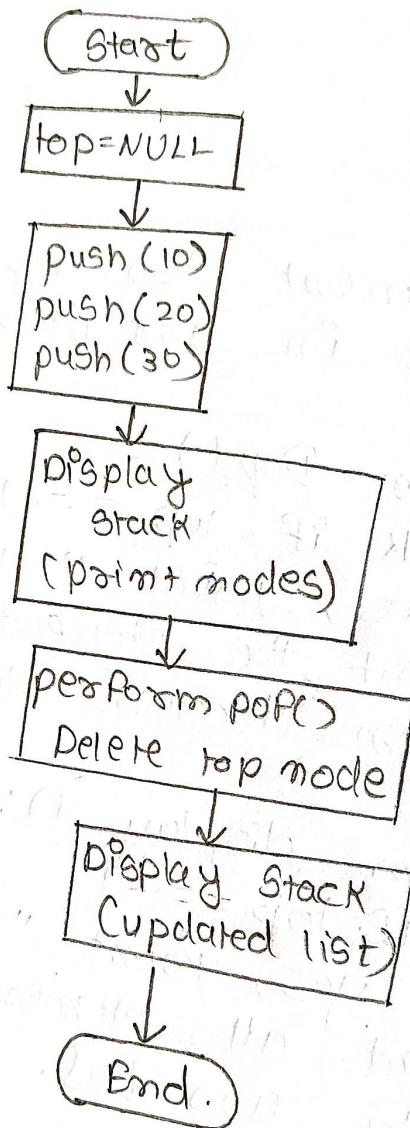
Node* head = NULL; // initialize

// Insert node at the end of list
void insertEnd(int value) {
    Node* newNode = new Node();
    newNode->data = value;
    newNode->next = NULL;
    newNode->prev = NULL;

    if (head == NULL) {
        head = newNode;
    } else {
        Node* temp = head;
        while (temp->next != NULL)
            temp = temp->next;
        temp->next = newNode;
        newNode->prev = temp;
    }
}

// Display all nodes in list
void display() {
    if (head == NULL)
        cout << "List is empty" << endl;
    else {
        Node* temp = head;
        cout << "List: ";
        while (temp != NULL) {
            cout << temp->data << " ";
            temp = temp->next;
        }
    }
}
  
```

Flowchart:



Array /  
Linked list

### Experiment

Title: Implement

a) Stack using

Program :

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

```
#define MAX 100
```

```
class Stack {
    int arr[MAX];
    int top;
}
```

```
public:
    Stack() {
        top = -1;
    }
```

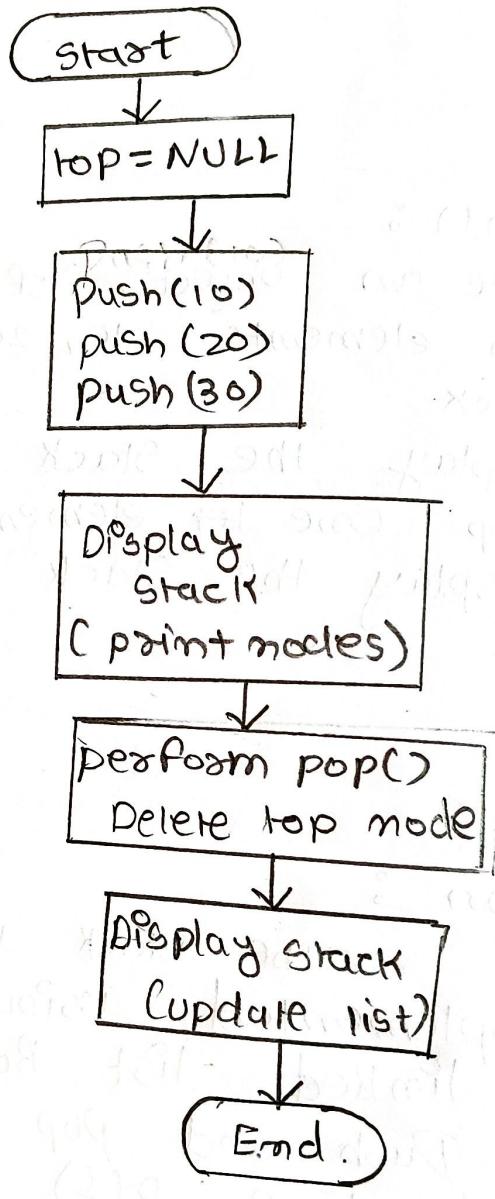
```
void push(int item) {
    if (top == MAX - 1)
        cout << "Stack overflow";
    else
        arr[++top] = item;
}
```

```
void pop() {
    if (top == -1)
        cout << "Stack underflow";
    else
        cout << arr[top--];
}
```

```
void display() {
    if (top == -1)
        cout << "Stack is empty";
    else
        for (int i = top; i >= 0; i--)
            cout << arr[i] << " ";
}
```

Stack  
Using  
Linked list

Algorithm:-



Stack Using linked list.

b) Stack using Lir

Program :

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

struct Node {
    int data;
    Node* next;
};

class Stack {
    Node* top;
public:
    Stack() { top = NULL; }

    void push(int new_data) {
        Node* new_node = new Node();
        new_node->data = new_data;
        new_node->next = top;
        top = new_node;
    }

    void display() {
        Node* temp = top;
        while (temp != NULL) {
            cout << temp->data << endl;
            temp = temp->next;
        }
    }
};
```