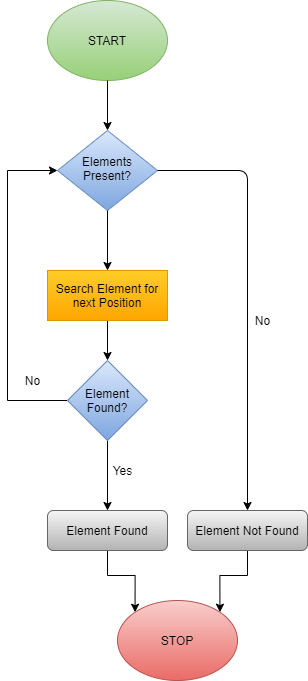
DSU Practicals

1. Draw a flowchart and algorithm and to develop a program for searching an given key elements using the linear search from list of n numbers.

#include <stdio.h>

int main()

{ int n, key, i, found = 0;

printf("Enter the number of elements in the list: "); 

scanf("%d", &n);

int arr[n];

printf("Enter %d elements:\n", n);

for(i = 0; i < n; i++) {

scanf("%d", &arr[i]);

}

printf("Enter the key element to search: ");

scanf("%d", &key);

for(i = 0; i < n; i++) {

if(arr[i] == key) {

found = 1;

printf("Element %d found at position %d.\n", key, i + 1);

break;

}

}

if(!found) {

printf("Element %d not found in the list.\n", key)

}

return 0;

}

**Linear Search Algorithm**

1. **Input**: A list of n elements and a key to search for.
2. **Initialize**:
3. Set found = 0 (meaning key not found initially).
4. **For** each element arr[i] from 0 to n-1:
5. If arr[i] == key:
   1. Print "Element found at position i + 1".
   2. Set found = 1.
   3. Exit the loop.
6. **If** found == 0:
7. Print "Element not found".
8. **End**.
9. Draw a flowchart and algorithm and to develop a program for searching an given key elements using the binary search from list of n numbers.

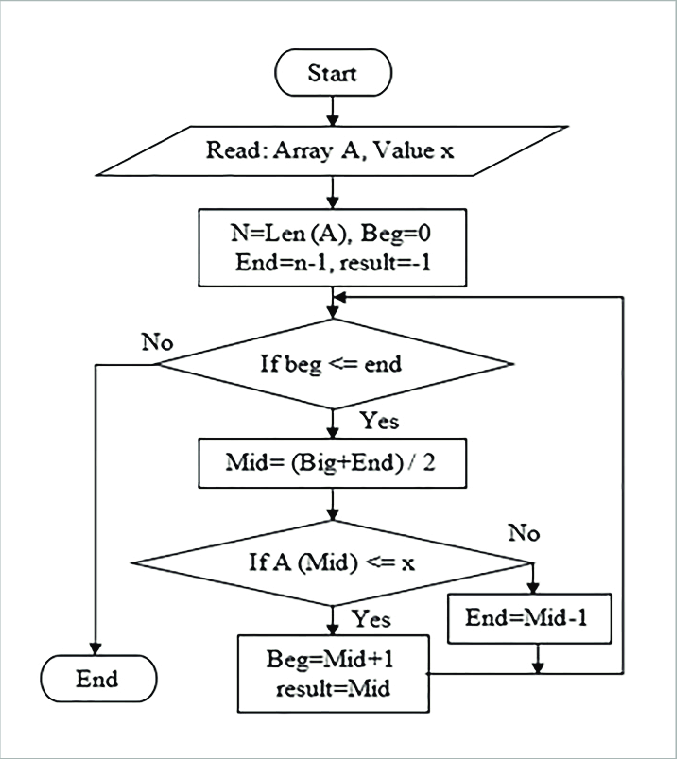
#include <stdio.h>

int binarySearch(int arr[], int n, int key) {

int low = 0, high = n - 1, mid;

while (low <= high) {

mid = (low + high) / 2; // Calculate middle index

 if (arr[mid] == key) {

return mid;

}

else if (arr[mid] < key) {

low = mid + 1;

}

else {

high = mid - 1;

}

}

return -1;

}

int main() {

int n, key, result;

printf("Enter the number of elements in the list: ");

scanf("%d", &n);

int arr[n];

printf("Enter %d sorted elements:\n", n);

for (int i = 0; i < n; i++) {

scanf("%d", &arr[i]);

}

printf("Enter the key element to search: ");

scanf("%d", &key);

result = binarySearch(arr, n, key);

if (result != -1) {

printf("Element %d found at position %d (1-based index).\n", key, result + 1);

} else {

printf("Element %d not found in the list.\n", key);

}

return 0;

}

**1.Input**: A sorted array of n numbers and a key to search for.

* + 1. Set low to the first index (0) and high to the last index (n-1).

1. **Repeat** the following steps while low <= high:
2. Find the middle index: mid = (low + high) / 2.
3. Compare the key with arr[mid]:
   1. If arr[mid] == key, print the position and exit (key found).
   2. If arr[mid] < key, update low = mid + 1 (search in the right half).
   3. If arr[mid] > key, update high = mid - 1 (search in the left half).
4. If low > high, print that the key is not found (key not in the array).

6. **End** the program.

1. Draw a flowchart and algorithm and to develop a program to sort given list of n numbers in ascending orders using bubble sort.

#include <stdio.h>

void bubbleSort(int arr[], int n)

{

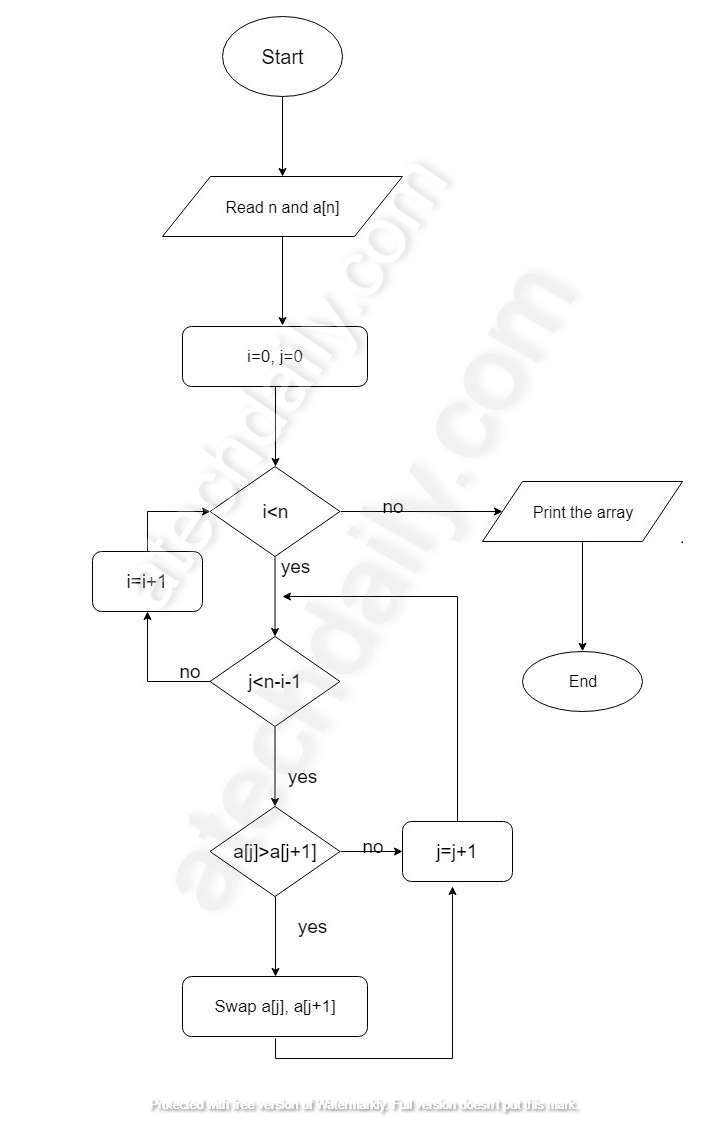
int i, j, temp;

for (i = 0; i < n - 1; i++)

{

for (j = 0; j < n - i - 1; j++)

{

 if (arr[j] > arr[j + 1])

{

temp = arr[j];

arr[j] = arr[j + 1];

arr[j + 1] = temp;

}

}

}

}

int main()

{

int n;

printf("Enter the number of elements: ");

scanf("%d", &n);

int arr[n];

printf("Enter the elements:\n");

for (int i = 0; i < n; i++) {

scanf("%d", &arr[i]);

}

bubbleSort(arr, n);

printf("Sorted elements in ascending order:\n");

for (int i = 0; i < n; i++) {

printf("%d ", arr[i]);

}

printf("\n");

return 0;

}

**1.Input**: A list arr[] of size n.

1. **For** i = 0 to n-2 (Outer loop for each pass):
2. **For** j = 0 to n-i-2 (Inner loop for each adjacent pair):
   1. **If** arr[j] > arr[j + 1], swap arr[j] and arr[j + 1].
3. Repeat steps 2 until the list is sorted.
4. **Output**: Sorted list arr[].
5. Draw a flowchart and algorithm and to develop a program to implements stack using array(PUSH/POP/DISPLAY)

#include <stdio.h>

#define SIZE 5 // Define stack size

int stack[SIZE], top = -1;

void push() {

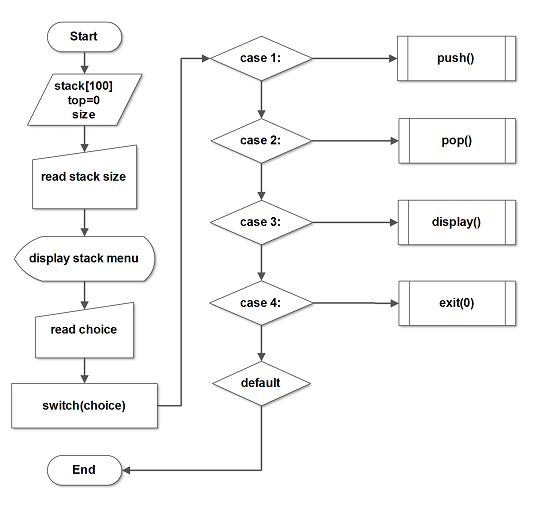
int value;

if (top == SIZE - 1) {

printf("Stack Overflow! Cannot push more elements.\n");

} else {

printf("Enter the value to push: ");

 scanf("%d", &value);

top++;

stack[top] = value;

printf("Value pushed successfully.\n");

}

}

void pop() {

if (top == -1) {

printf("Stack Underflow! No elements to pop.\n");

} else {

printf("Popped element: %d\n", stack[top]);

top--;

}

}

void display() {

if (top == -1) {

printf("Stack is Empty!\n");

} else {

printf("Stack elements are: ");

for (int i = top; i >= 0; i--) {

printf("%d ", stack[i]);

}

printf("\n");

}

}

int main() {

int choice;

while (1) {

printf("\nStack Operations:\n");

printf("1. Push\n2. Pop\n3. Display\n4. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

push();

break;

case 2:

pop();

break;

case 3:

display();

break;

case 4:

printf("Exiting program.\n");

return 0;

default:

printf("Invalid choice! Please try again.\n");

}

}

}

Step 1: Initialize `top = -1` and define stack array of size N.

Step 2: Repeat until the user chooses to exit:

a. Display menu for operations (PUSH, POP, DISPLAY, EXIT).

b. Take user input for choice.

c. If choice == PUSH:

i. If `top == size - 1`, display "Stack Overflow".

ii. Else, increment `top` and add element at `stack[top]`.

d. If choice == POP:

i. If `top == -1`, display "Stack Underflow".

ii. Else, display `stack[top]`, then decrement `top`.

e. If choice == DISPLAY:

i. If `top == -1`, display "Stack is Empty".

ii. Else, iterate and display elements from `stack[top]` to `stack[0]`.

f. If choice == EXIT, terminate the program.

Step 3: End.

1. Draw a flowchart and algorithm and to develop a program to implements queue using array(Insertion /Deletion/Display)

#include <stdio.h>

#define MAX 5

int queue[MAX];

int front = -1, rear = -1;

void enqueue(int item) {

if (rear == MAX - 1) {

printf("Queue is full\n");

return;

}

if (front == -1) {

front = 0;

}

queue[++rear] = item;

printf("%d inserted into the queue\n", item);

}

void dequeue() {

if (front == -1 || front > rear) {

printf("Queue is empty\n");

return;

}

printf("%d removed from the queue\n", queue[front++]);

if (front > rear) {

front = rear = -1;

}

}

void display() {

if (front == -1) {

printf("Queue is empty\n");

return;

}

printf("Queue elements are: ");

for (int i = front; i <= rear; i++) {

printf("%d ", queue[i]);

}

printf("\n");

}

int main() {

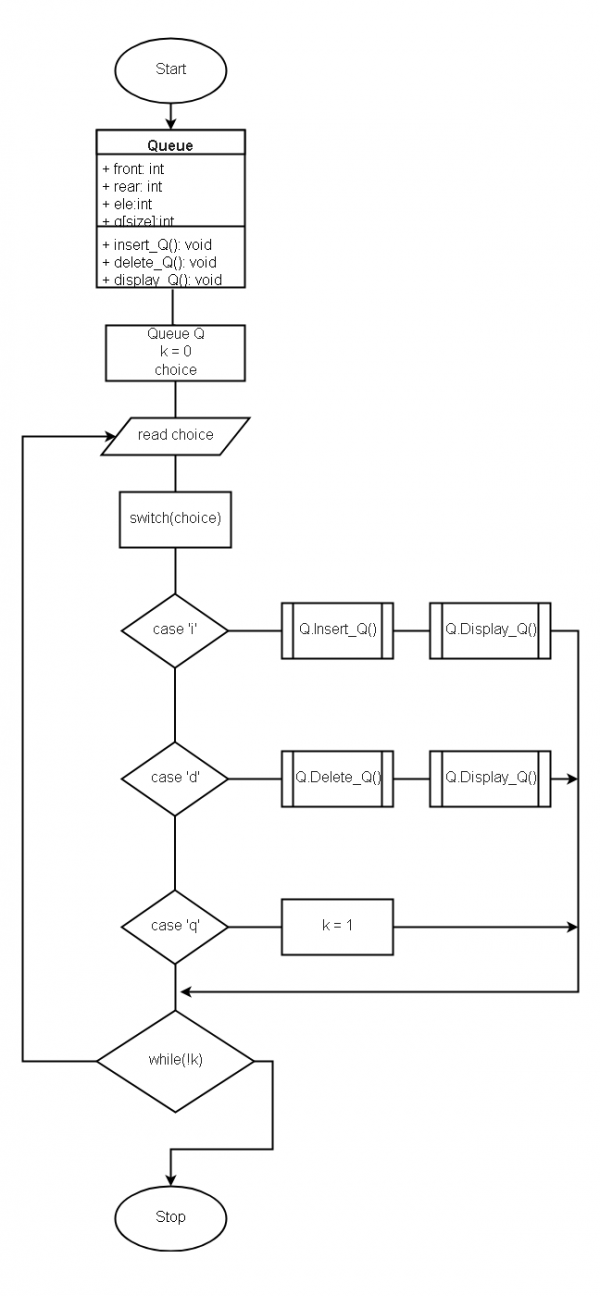
int choice, item;

while (1) {

printf("\n1. Enqueue\n2. Dequeue\n3. Display\n4. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);



switch (choice) {

case 1:

printf("Enter the element to enqueue: ");

scanf("%d", &item);

enqueue(item);

break;

case 2:

dequeue();

break;

case 3:

display();

break;

case 4:

return 0;

default:

printf("Invalid choice, please try again\n");

}

}

}

**1.Start**

* + 1. **Initialize front and rear to -1**

1. Display menu:
2. 1: Enqueue
3. 2: Dequeue
4. 3: Display
5. 4: Exit
6. Check user input:
7. **Enqueue**:
   1. Check if rear == MAX - 1 → Display "Queue is full"
   2. Else increment rear, insert element, and update front = 0 if front == -1.
8. **Dequeue**:
   1. Check if front == -1 or front > rear → Display "Queue is empty"
   2. Else remove element, increment front. If front > rear, reset front and rear.
9. **Display**:
   1. If front == -1 → Display "Queue is empty".
   2. Else print elements from queue[front] to queue[rear].
10. **Exit**:
    1. End program.
11. Draw a flowchart and algorithm and to develop a program to implements Create and insert at start operation in Link list.

#include <stdio.h>

#include <stdlib.h>

struct Node {

int data;

struct Node\* next;

};

void insertAtStart(struct Node\*\* head, int value) {

struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node));

if (!newNode) {

printf("Memory allocation failed\n");

return;

}

newNode->data = value;

newNode->next = \*head;

\*head = newNode;

printf("Inserted %d at the start of the list\n", value);

}

void displayList(struct Node\* head) {

if (head == NULL) {

printf("The list is empty\n");

return;

}

printf("Linked List: ");

while (head != NULL) {

printf("%d -> ", head->data);

head = head->next;

}

printf("NULL\n");

}

int main() {

struct Node\* head = NULL;

int choice, value;

while (1) {

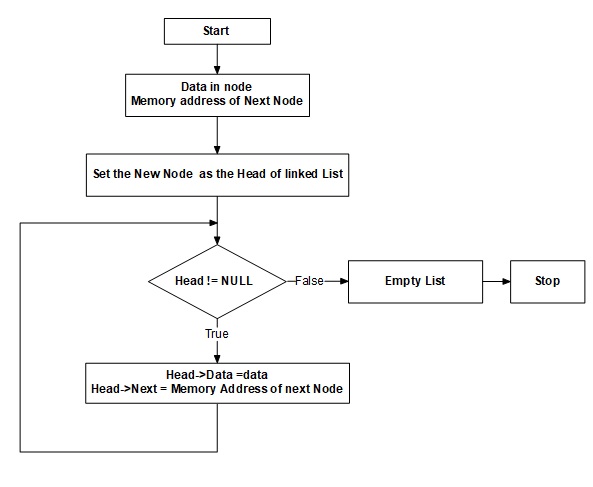
printf("\n1. Insert at start\n2. Display list\n3. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

 printf("Enter the value to insert: ");

scanf("%d", &value);

insertAtStart(&head, value);

break;

case 2:

displayList(head);

break;

case 3:

return 0;

default:

printf("Invalid choice. Please try again.\n");

}

}

}

#### Step 1: Define Node Structure

* Create a struct for the node with two fields:
  + data: To store the value.
  + next: A pointer to the next node.

#### Step 2: Initialize the Linked List

* Start with an empty list (head = NULL).

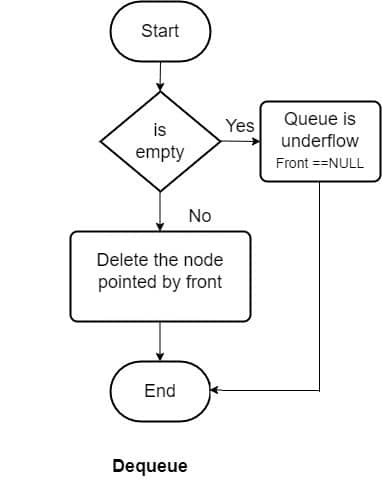
#### Step 3: Insert at Start

1. Allocate memory for a new node.
2. Assign the data to the new node.
3. Set the next pointer of the new node to the current head.
4. Update the head to point to the new node.

#### Step 4: Display the List

1. Start from the head node.
2. Traverse the list until the end (current != NULL).
3. Print each node's data and move to the next node.
4. Draw a flowchart and algorithm and to develop a program to implements queue using link list(deletion/Display) .

#include <stdio.h>

#include <stdlib.h>

struct Node {

int data;

struct Node\* next;

};

struct Node\* front = NULL;

struct Node\* rear = NULL;

void dequeue() {

if (front == NULL) {

printf("Queue is empty\n");

return;

}

struct Node\* temp = front;

printf("Deleted %d from the queue\n", temp->data);

front = front->next;

if (front == NULL) {

rear = NULL;

}

free(temp);

}

void displayQueue() {

if (front == NULL) {

printf("Queue is empty\n");

return;

}

struct Node\* current = front;

printf("Queue elements: ");

while (current != NULL) {

printf("%d -> ", current->data);

current = current->next;

}

printf("NULL\n");

}

int main() {

int choice;

while (1) {

printf("\n1. Dequeue\n2. Display\n3. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

dequeue();

break;

case 2:

displayQueue();

break;

case 3:

return 0;

default:

printf("Invalid choice, please try again\n");

}

}

}

#### Step 1: Define Node Structure

* Create a struct for the node with two fields:
  + data: To store the value.
  + next: A pointer to the next node.

#### Step 2: Initialize the Queue

* Start with an empty queue (front = NULL and rear = NULL).

#### Step 3: Deletion (Dequeue)

1. Check if the queue is empty:
   * If front == NULL, display "Queue is empty" and return.
2. Else:
   * Store the current front node in a temporary variable.
   * Move front to the next node (front = front->next).
   * If front == NULL, set rear = NULL.
   * Free the memory of the temporary node.

#### Step 4: Display

1. Check if the queue is empty:
   * If front == NULL, display "Queue is empty".
2. Else:
   * Start from front and traverse to the end (current != NULL).
   * Print the data of each node.
3. Draw a flowchart and algorithm and to develop a program to sort given list of n numbers in ascending orders using insertion sort.

#include <stdio.h>

void insertionSort(int arr[], int n) {

for (int i = 1; i < n; i++) {

int key = arr[i];

int j = i - 1;

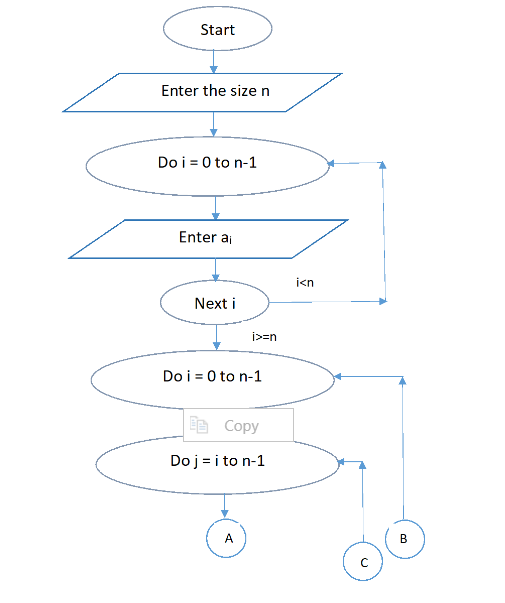
while (j >= 0 && arr[j] > key) {

arr[j + 1] = arr[j];

j--;

}

arr[j + 1] = key; // Insert the key at its correct position

 }

}

void printArray(int arr[], int n) {

for (int i = 0; i < n; i++) {

printf("%d ", arr[i]);

}

printf("\n");

}

int main() {

int n;

printf("Enter the number of elements: ");

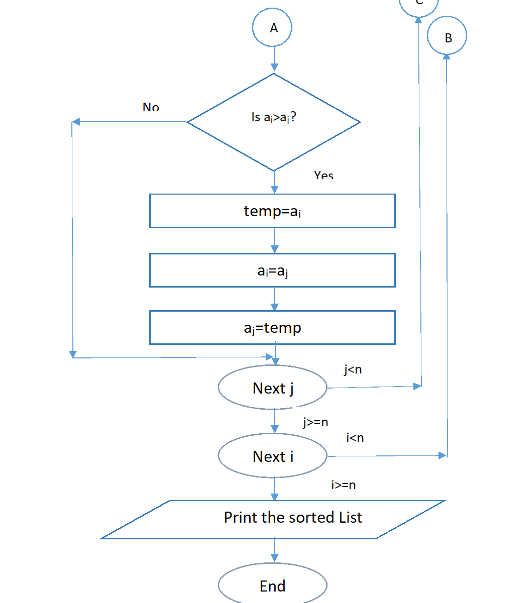
scanf("%d", &n);

int arr[n];

printf("Enter %d elements: ", n);

for (int i = 0; i < n; i++) {

scanf("%d", &arr[i]);

 }

printf("Original array: ");

printArray(arr, n);

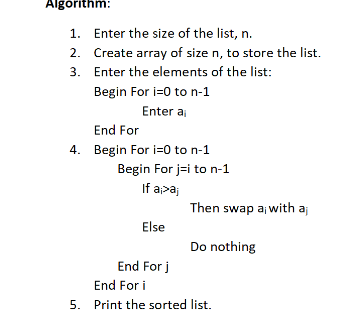
insertionSort(arr, n);

printf("Sorted array in ascending order: ");

printArray(arr, n);

return 0;

}



13.Draw a flowchart and algorithm and to develop a program to implements queue using link list(Insertion/Display)

#include <stdio.h>

#include <stdlib.h>

struct Node {

int data;

struct Node\* next;

};

struct Node\* front = NULL;

struct Node\* rear = NULL;

void enqueue(int value) {

struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node));

if (!newNode) {

printf("Memory allocation failed\n");

return;

}

newNode->data = value;

newNode->next = NULL;

if (rear == NULL) {

front = rear = newNode;

printf("Inserted %d into the queue\n", value);

return;

}

rear->next = newNode;

rear = newNode;

printf("Inserted %d into the queue\n", value);

}

void displayQueue() {

if (front == NULL) {

printf("Queue is empty\n");

return;

}

struct Node\* current = front;

printf("Queue elements: ");

while (current != NULL) {

printf("%d -> ", current->data);

current = current->next;

}

printf("NULL\n");

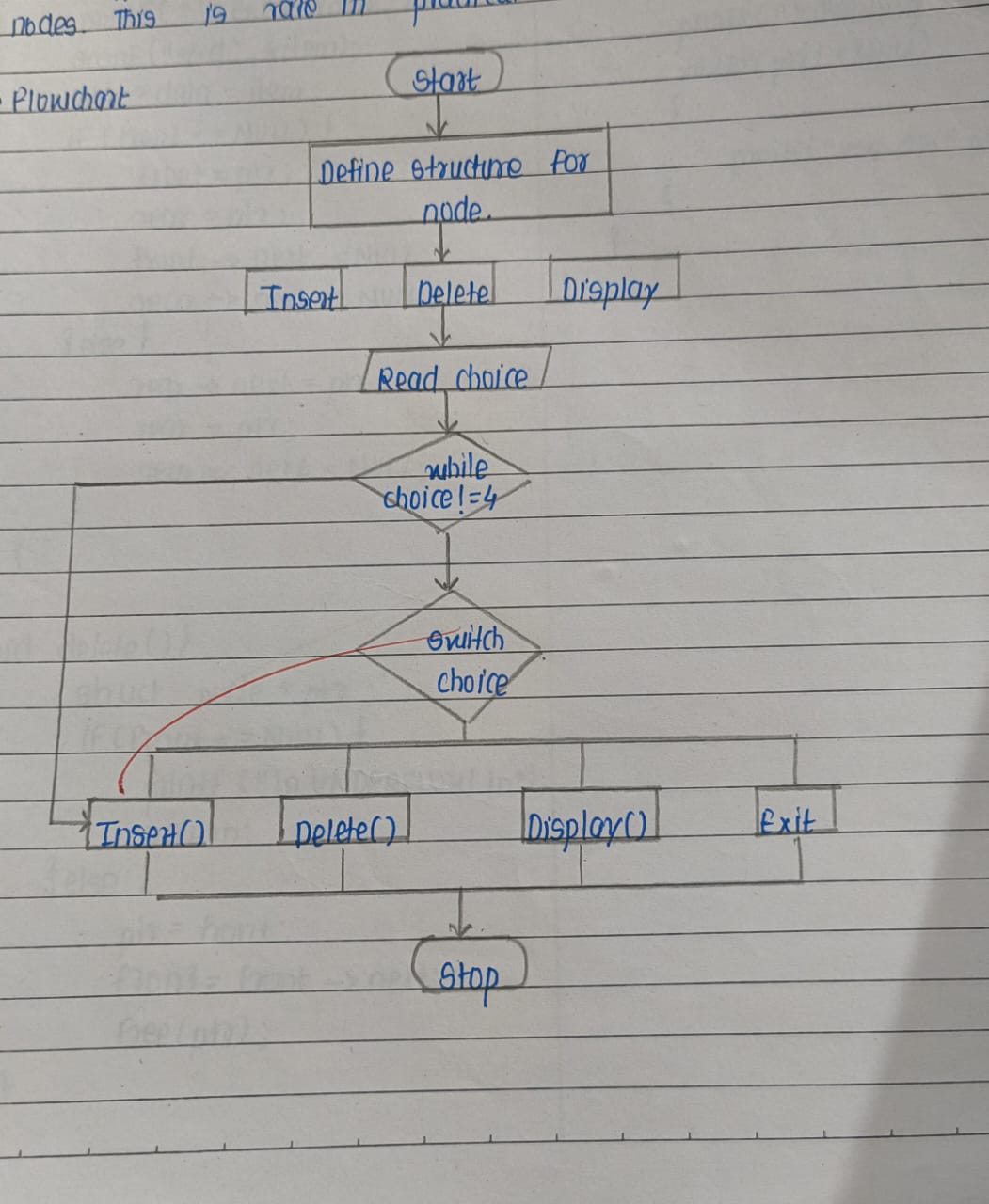
}

int main() {

int choice, value;

while (1) {

printf("\n1. Enqueue\n2. Display\n3. Exit\n");

 printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter the value to insert: ");

scanf("%d", &value);

enqueue(value);

break;

case 2:

displayQueue();

break;

case 3:

return 0;

default:

printf("Invalid choice, please try again\n");

}

}

}

#### Step 1: Define Node Structure

* Create a struct for the node with two fields:
  + data: To store the value.
  + next: A pointer to the next node.

#### Step 2: Initialize the Queue

* Start with an empty queue (front = NULL and rear = NULL).

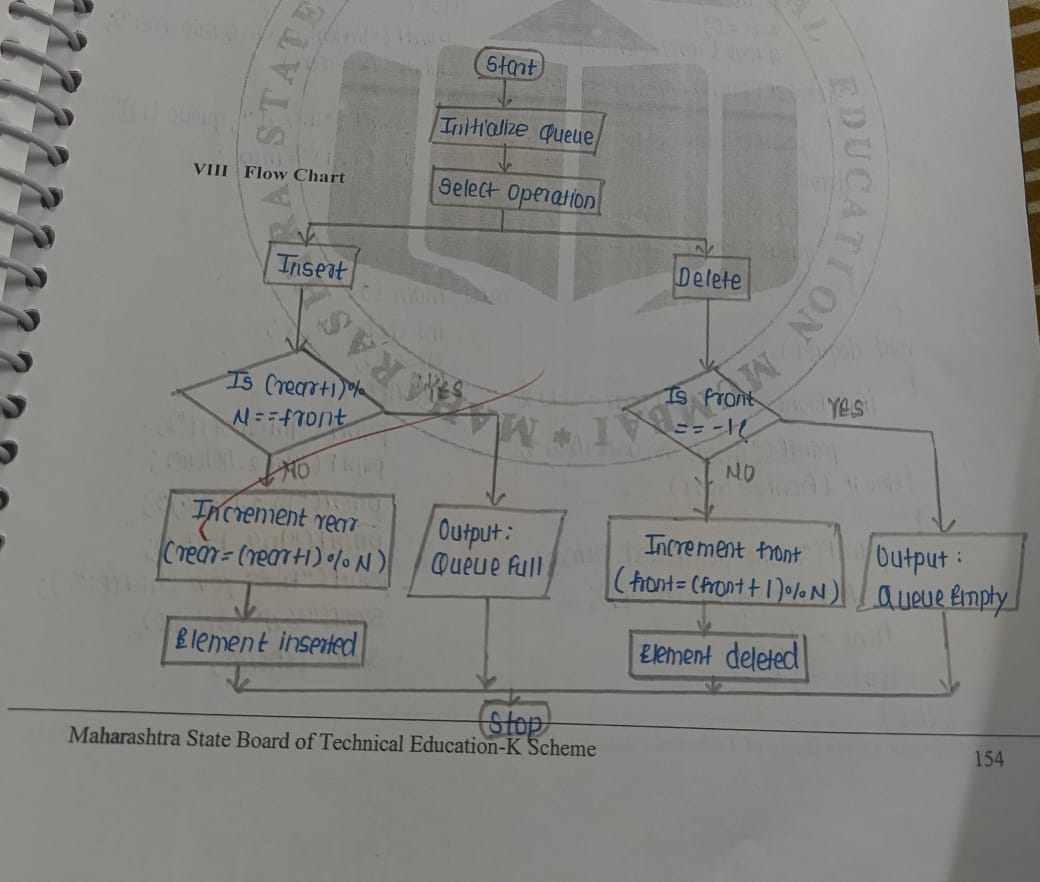
#### Step 3: Insertion (Enqueue)

1. Create a new node.
2. Assign the value to the new node.
3. Set the next pointer of the new node to NULL.
4. If the queue is empty (front == NULL), set both front and rear to the new node.
5. Else, update rear->next to point to the new node and set rear to the new node.

#### Step 4: Display

1. Check if the queue is empty:
   * If front == NULL, display "Queue is empty".
2. Else:
   * Start from front and traverse to the end (current != NULL).
   * Print the data of each node.

14.Draw a flowchart and algorithm and to develop a program to implements Circular queue using array(Insertion /Deletion/Display)



#include <stdio.h>

#define MAX 5

int queue[MAX];

int front = -1, rear = -1;

void enqueue(int value) {

if ((rear + 1) % MAX == front) {

printf("Queue is full\n");

return;

}

if (front == -1) {

front = rear = 0;

} else {

rear = (rear + 1) % MAX;

}

queue[rear] = value;

printf("Inserted %d\n", value);

}

void dequeue() {

if (front == -1) {

printf("Queue is empty\n");

return;

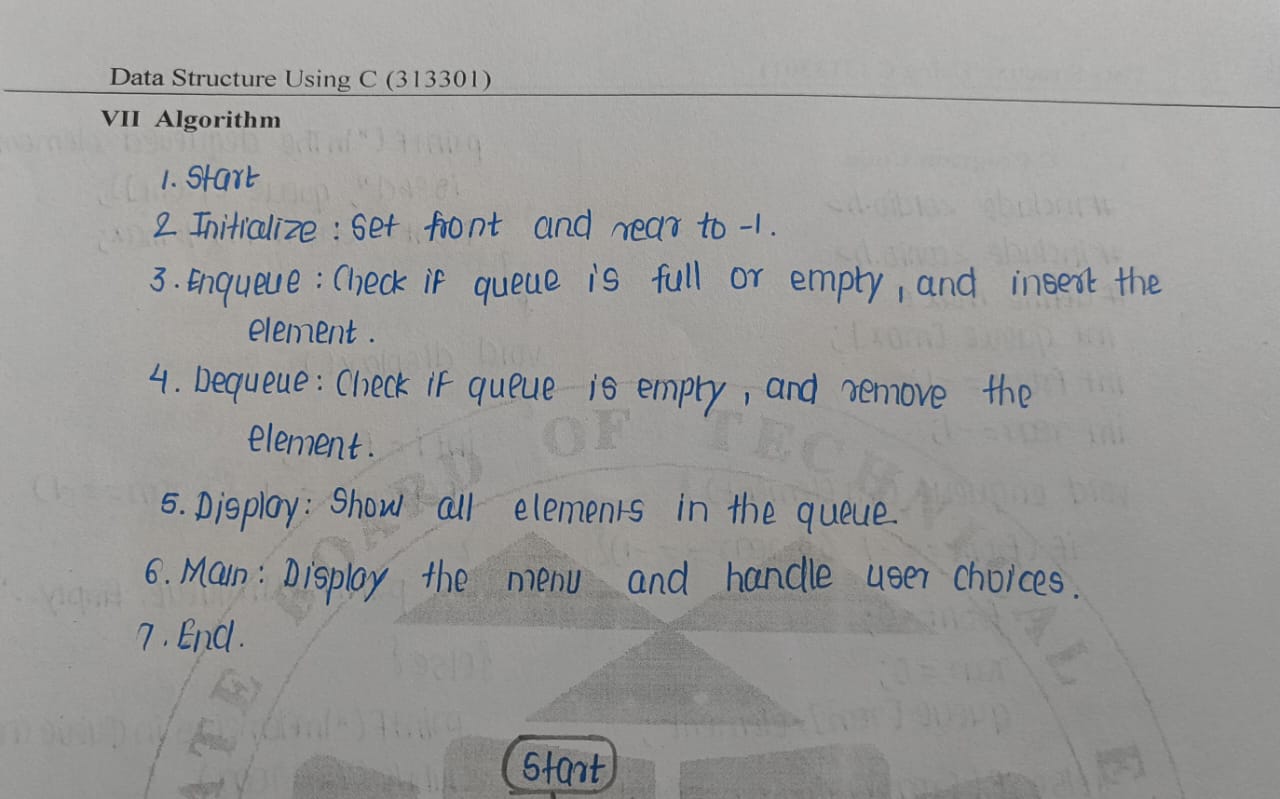
}

printf("Deleted %d\n", queue[front]);

if (front == rear) {

front = rear = -1; // Queue is now empty

} else {

 front = (front + 1) % MAX;

}

}

void displayQueue() {

if (front == -1) {

printf("Queue is empty\n");

return;

}

printf("Queue elements: ");

int i = front;

do {

printf("%d ", queue[i]);

i = (i + 1) % MAX;

} while (i != (rear + 1) % MAX);

printf("\n");

}

int main() {

int choice, value;

while (1) {

printf("\n1. Enqueue\n2. Dequeue\n3. Display\n4. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter the value to insert: ");

scanf("%d", &value);

enqueue(value);

break;

case 2:

dequeue();

break;

case 3:

displayQueue();

break;

case 4:

return 0;

default:

printf("Invalid choice, please try again\n");

}

}

}

15.Draw a flowchart and algorithm and to develop a program to implements Create and insert at end operation in Link list.

#include <stdio.h>

#include <stdlib.h>

struct Node {

int data;

struct Node\* next;

};

struct Node\* createNode(int value) {

struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node));

newNode->data = value;

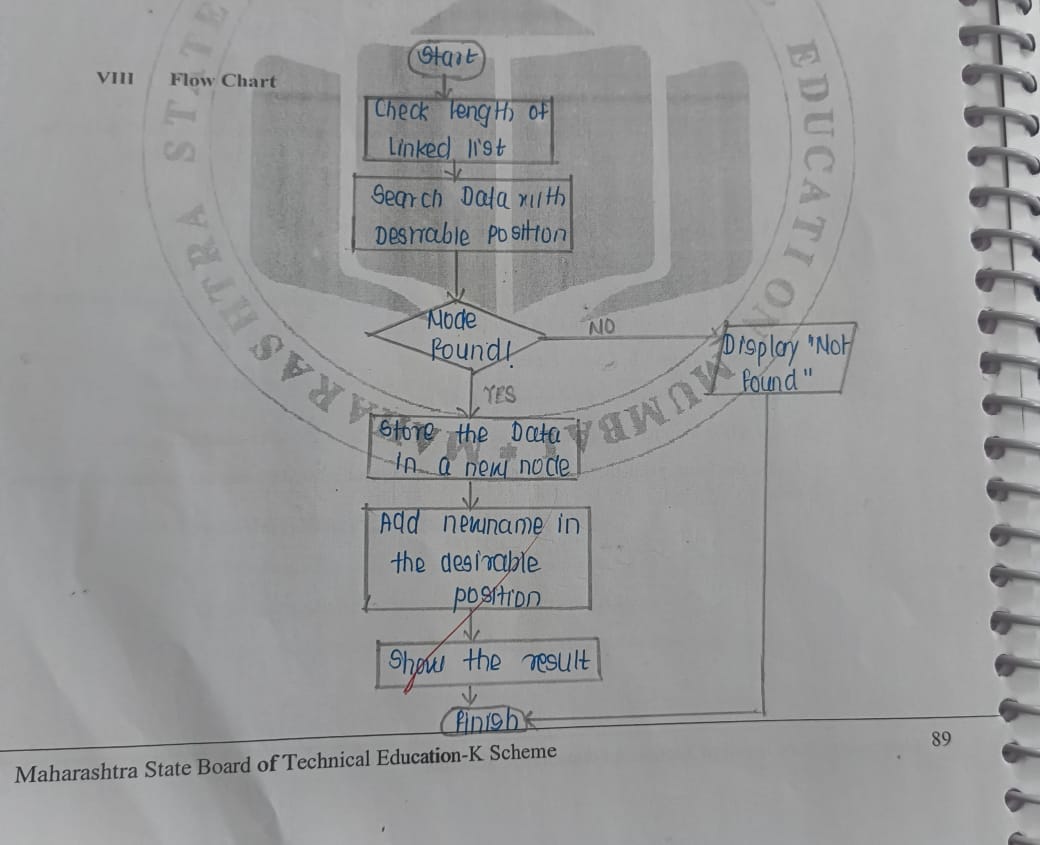
newNode->next = NULL;

return newNode;

}

void insertAtEnd(struct Node\*\* head, int value) {

struct Node\* newNode = createNode(value);

 if (\*head == NULL) {

\*head = newNode;

} else {

struct Node\* temp = \*head;

// Traverse to the last node

while (temp->next != NULL) {

temp = temp->next;

}

temp->next = newNode;

}

}

void displayList(struct Node\* head) {

struct Node\* temp = head;

while (temp != NULL) {

printf("%d -> ", temp->data);

temp = temp->next;

}

printf("NULL\n");

}

int main() {

struct Node\* head = NULL; // Initialize the head pointer to NULL

// Insert nodes at the end of the linked list

insertAtEnd(&head, 10);

insertAtEnd(&head, 20);

insertAtEnd(&head, 30);

// Display the linked list

printf("Linked List: ");

displayList(head);

return 0;

}

 **Start**

 Define a structure for the linked list node containing:

* An integer data for the node's value.
* A pointer next to the next node.

 Create a function createNode to allocate memory for a new node and set its value.

 Create a function insertAtEnd to:

* Check if the linked list is empty.
  + If empty, set the new node as the head.
  + Otherwise, traverse the linked list to find the last node and link the new node to it.

 Create a function displayList to print all node values in the linked list.

 In the main function:

* Initialize the head pointer as NULL.
* Use insertAtEnd to add new nodes to the list.
* Call displayList to show the current linked list.

 **End**

16.Draw a flowchart and algorithm and to develop a program to implements stack using Link list (PUSH/DISPLAY)

#include <stdio.h>

#include <stdlib.h>

// Define the structure for a stack node

struct Node {

int data;

struct Node\* next;

};

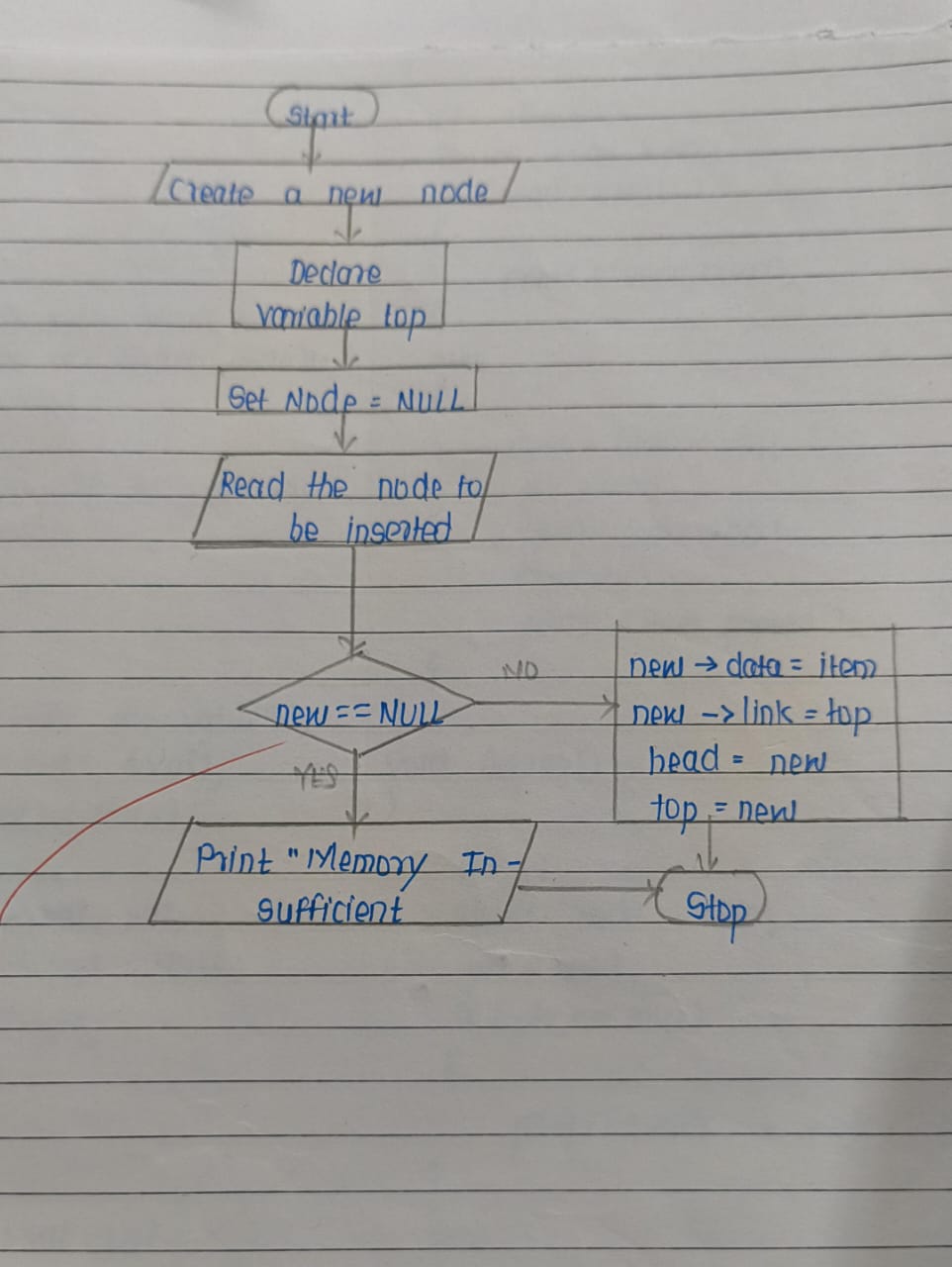
// Function to push an element onto the stack

void push(struct Node\*\* top, int value) {

struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node));

if (newNode == NULL) {

printf("Memory allocation failed. Stack Overflow!\n");

 return;

}

newNode->data = value;

newNode->next = \*top;

\*top = newNode;

printf("%d pushed onto the stack.\n", value);

}

// Function to display the stack

void display(struct Node\* top) {

if (top == NULL) {

printf("Stack is empty.\n");

return;

}

struct Node\* temp = top;

printf("Stack: ");

while (temp != NULL) {

printf("%d -> ", temp->data);

temp = temp->next;

}

printf("NULL\n");

}

// Main function

int main() {

struct Node\* top = NULL; // Initialize the stack as empty

int choice, value;

while (1) {

printf("\n1. PUSH\n2. DISPLAY\n3. EXIT\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter the value to push: ");

scanf("%d", &value);

push(&top, value);

break;

case 2:

display(top);

break;

case 3:

printf("Exiting...\n");

return 0;

default:

printf("Invalid choice! Please try again.\n");

}

}

}

 **Start**

 Define a structure Node with two members:

* An integer data to store the value.
* A pointer next to point to the next node in the stack.

 Initialize a pointer top as NULL to represent the top of the stack.

 Implement the PUSH operation:

* Allocate memory for a new node.
* Assign the input value to the data field of the new node.
* Set the next pointer of the new node to the current top.
* Update top to point to the new node.

 Implement the DISPLAY operation:

* Check if the stack is empty (top == NULL).
* If empty, print "Stack is empty".
* Otherwise, traverse from top to the end of the list, printing each node's data.

 Create a menu-driven program in the main function to allow the user to choose between PUSH, DISPLAY, or exit.

 Repeat until the user chooses to exit.

 **End**

17.Draw a flowchart and algorithm and to develop a program to implements stack using Link list (POP/DISPLAY)

#include <stdio.h>

#include <stdlib.h>

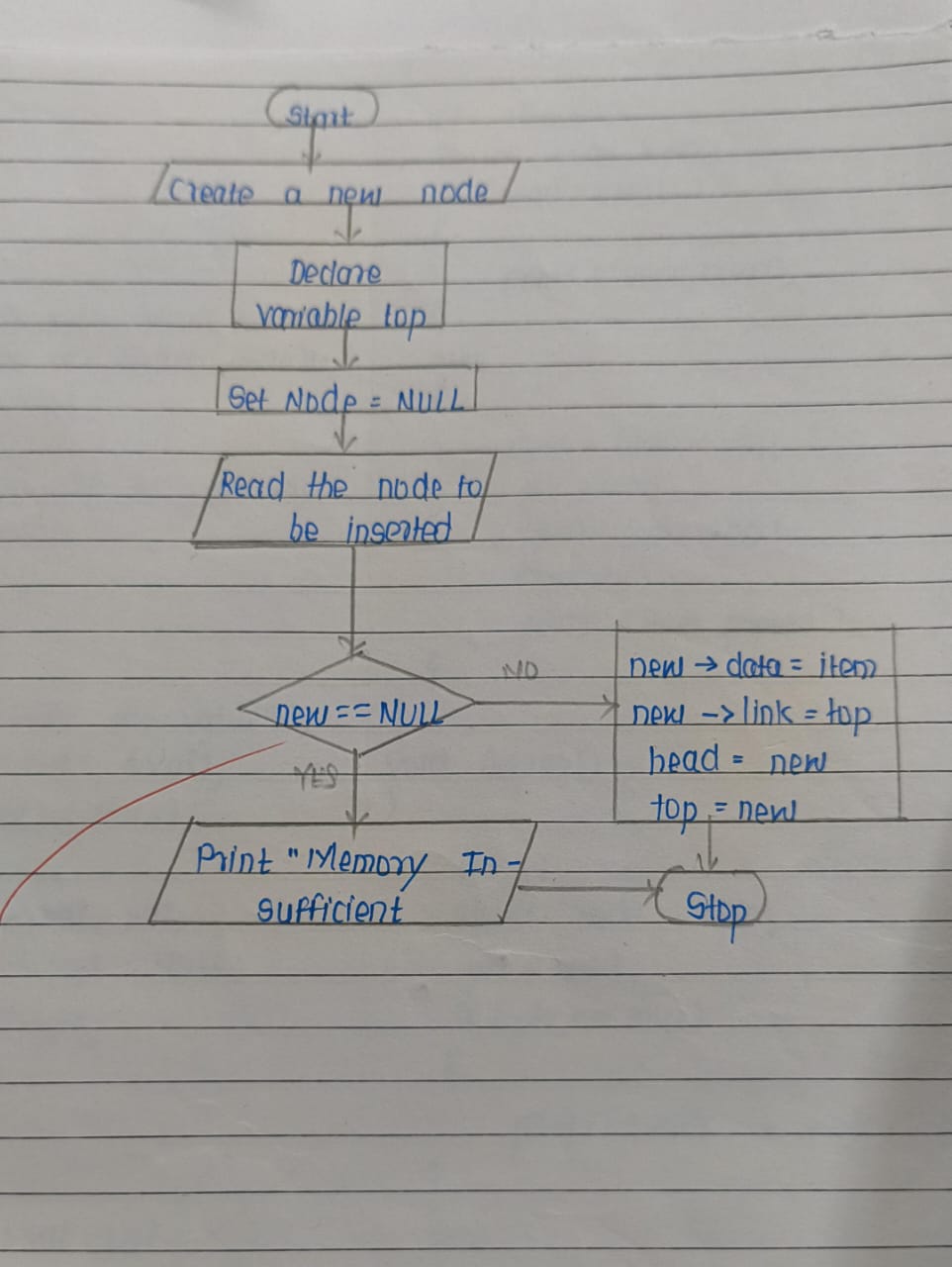
// Define the structure for a stack node

struct Node {

int data;

struct Node\* next;

};

// Function to pop an element from the stack

void pop(struct Node\*\* top) {

if (\*top == NULL) {

printf("Stack Underflow! No elements to pop.\n");

return;

}

struct Node\* temp = \*top;

printf("Popped element: %d\n", temp->data);

\*top = (\*top)->next; // Update top to the next node

free(temp); // Free the memory of the popped node

}

// Function to display the stack

void display(struct Node\* top) {

if (top == NULL) {

printf("Stack is empty.\n");

return;

}

struct Node\* temp = top;

printf("Stack: ");

while (temp != NULL) {

printf("%d -> ", temp->data);

temp = temp->next;

}

printf("NULL\n");

}

// Main function

int main() {

struct Node\* top = NULL; // Initialize the stack as empty

int choice, value;

while (1) {

printf("\n1. POP\n2. DISPLAY\n3. EXIT\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

pop(&top);

break;

case 2:

display(top);

break;

case 3:

printf("Exiting...\n");

return 0;

default:

printf("Invalid choice! Please try again.\n");

}

}

}

1. **Start**
2. Define a structure Node with two members:
   * An integer data to store the value.
   * A pointer next to point to the next node in the stack.
3. Initialize a pointer top as NULL to represent the top of the stack.
4. Implement the POP operation:
   * Check if the stack is empty (top == NULL).
     + If empty, print "Stack Underflow" and exit the function.
   * Otherwise, store the top node in a temporary pointer.
   * Update top to point to the next node.
   * Free the temporary node and print the popped value.
5. Implement the DISPLAY operation:
   * Check if the stack is empty (top == NULL).
   * If empty, print "Stack is empty".
   * Otherwise, traverse from top to the end of the list and print each node's data.
6. Create a menu-driven program in the main function to allow the user to choose between POP, DISPLAY, or exit.
7. Repeat until the user chooses to exit.
8. **End**

18. Draw a flowchart and algorithm and to develop a program to sort given list of n numbers using Selection sort.

#include <stdio.h>

void selectionSort(int arr[], int n) {

for (int i = 0; i < n - 1; i++) {

int minIndex = i; // Assume the first unsorted element is the minimum

for (int j = i + 1; j < n; j++) {

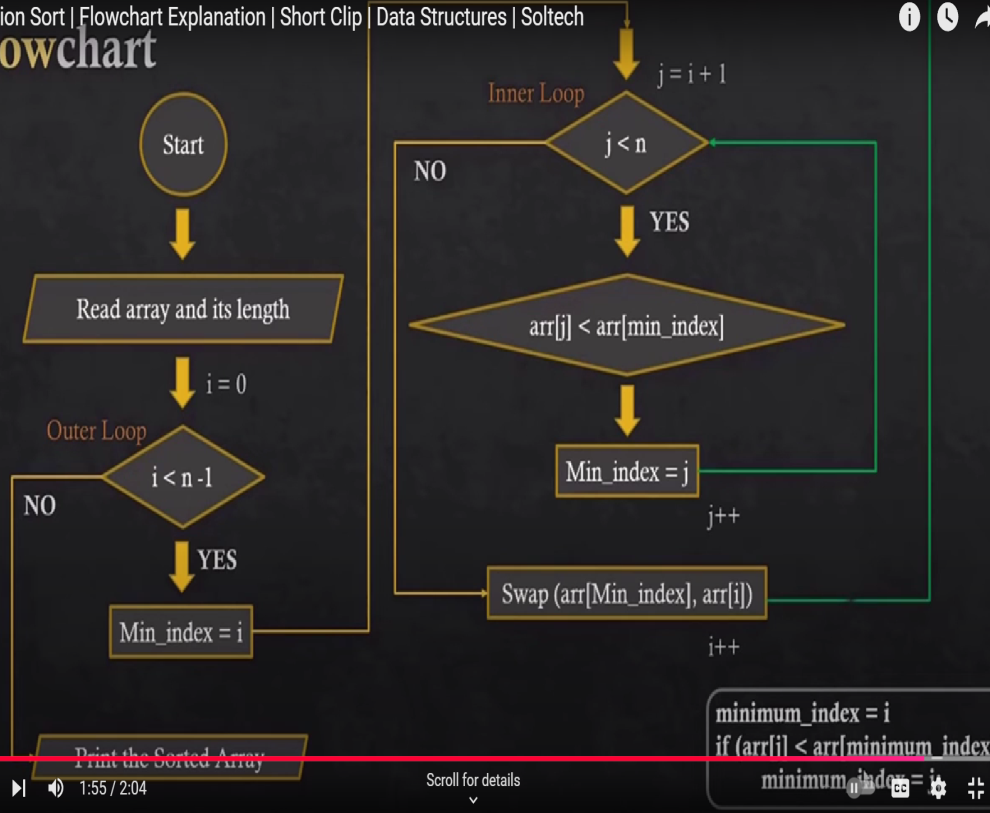
if (arr[j] < arr[minIndex]) {

minIndex = j; // Update minIndex if a smaller element is found

}

}

// Swap the found minimum element with the first unsorted element

 if (minIndex != i) {

int temp = arr[i];

arr[i] = arr[minIndex];

arr[minIndex] = temp;

}

}

}

void printArray(int arr[], int n) {

for (int i = 0; i < n; i++) {

printf("%d ", arr[i]);

}

printf("\n");

}

int main() {

int n;

printf("Enter the number of elements: ");

scanf("%d", &n);

int arr[n];

printf("Enter %d elements:\n", n);

for (int i = 0; i < n; i++) {

scanf("%d", &arr[i]);

}

printf("Original array: ");

printArray(arr, n);

selectionSort(arr, n);

printf("Sorted array: ");

printArray(arr, n);

return 0;

}

 **Start**

 Input the size of the list, n, and the elements of the list.

 Repeat for i from 0 to n-2:

* Set minIndex = i.
* Repeat for j from i+1 to n-1:
  + If list[j] < list[minIndex], set minIndex = j.
* If minIndex is not i, swap list[i] and list[minIndex].

 Print the sorted list.

 **End**