**1. Array:**

- Definition: A collection of elements stored in contiguous memory locations.

- Features: Random access, fixed size.

- Operations: Access, insertion, deletion (in some cases).

- Complexity:

- Access: O(1)

- Insertion/Deletion: O(n) (if elements need to be shifted)

- Implementation:

#define SIZE 10

int arr[SIZE]; // Declaration of an array

**2. Linked List:**

- Definition: A collection of nodes where each node contains a data field and a reference to the next node in the sequence.

- Features: Dynamic size, efficient insertion and deletion.

- Operations: Insertion, deletion, traversal.

- Complexity:

- Access: O(n)

- Insertion/Deletion: O(1)

- Implementation:

struct Node {

int data;

struct Node\* next;

};

struct Node\* head = NULL; // Declaration of an empty linked list

**3. Stack:**

- Definition: A Last In First Out (LIFO) data structure where elements are inserted and removed from the same end, called the top.

- Features: Push (insert), pop (remove), peek (get top element).

- Operations: Push, pop, peek.

- Complexity:

- Push, pop, peek: O(1)

- Implementation:

#define MAX\_SIZE 100

int stack[MAX\_SIZE];

int top = -1; // Empty stack

void push(int value) {

if (top == MAX\_SIZE - 1) {

printf("Stack overflow\n");

return;

}

stack[++top] = value;

}

int pop() {

if (top == -1) {

printf("Stack underflow\n");

return -1; // or any appropriate error value

}

return stack[top--];

}

**4. Queue:**

- Definition: A First In First Out (FIFO) data structure where elements are inserted at the rear end and removed from the front end.

- Features: Enqueue (insert), dequeue (remove), front (get front element).

- Operations: Enqueue, dequeue, front.

- Complexity:

- Enqueue, dequeue: O(1)

- Implementation:

#define MAX\_SIZE 100

int queue[MAX\_SIZE];

int front = -1, rear = -1; // Empty queue

void enqueue(int value) {

if (rear == MAX\_SIZE - 1) {

printf("Queue overflow\n");

return;

}

if (front == -1) {

front = 0;

}

queue[++rear] = value;

}

int dequeue() {

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

printf("Queue underflow\n");

return -1; // or any appropriate error value

}

return queue[front++];

}

**5. Binary Tree:**

- Definition: A tree data structure where each node has at most two children.

- Features: Recursive structure, efficient searching, insertion, and deletion (in some cases).

- Operations: Insertion, deletion, traversal (inorder, preorder, postorder).

- Complexity:

- Search, insertion, deletion (in some cases): O(log n) on average for balanced trees, O(n) worst-case for unbalanced trees.

- Implementation:

struct Node {

int data;

struct Node\* left;

struct Node\* right;

};

struct Node\* createNode(int data) {

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

newNode->data = data;

newNode->left = NULL;

newNode->right = NULL;

return newNode;

}