Data Structures - Unit 1 Notes

This unit provides an introduction to fundamental data structures, algorithm analysis, and basic operations on linear structures like arrays, linked lists, stacks, and queues.

1. Introduction

Data structures are specialized ways of organizing and storing data in a computer so that it can be used efficiently. Different kinds of data structures are suited to different kinds of applications, and some are highly specialized to specific tasks. Understanding data structures is fundamental to writing efficient and optimized programs. This unit covers the basic building blocks, setting the stage for more complex structures later in the course.

2. Key Concepts

* **Abstract Data Type (ADT):** A high-level description of a data structure, focusing on its operations and behavior rather than its specific implementation. Examples include lists, stacks, and queues.
* **Algorithm Analysis:** Determining the efficiency of an algorithm by analyzing its time and space complexity, often expressed using Big O notation (e.g., O(n), O(log n), O(n^2)).
* **Arrays:** Contiguous blocks of memory used to store elements of the same data type. Accessing elements is fast using their index.
* **Pointers:** Variables that store memory addresses. Used extensively in implementing linked lists and other dynamic data structures.
* **Linked Lists:** A sequence of nodes where each node contains data and a pointer to the next node. Variations include singly, doubly, and circular linked lists.
* **Stacks:** A LIFO (Last-In, First-Out) data structure. Primary operations are push (add an

element) and pop (remove an element).

* **Queues:** A FIFO (First-In, First-Out) data structure. Primary operations are enqueue (add an element) and dequeue (remove an element).

* **Strings:** Sequences of characters. Often implemented as arrays of characters.

3. Formulas & Examples

* **Array Access:** `element = array[index]`

    * Example: `int numbers[5] = {1, 2, 3, 4, 5};  int thirdNumber = numbers[2]; // thirdNumber will be 3`

* **Linked List Traversal:**

    ```c++
    Node* current = head;
    while (current != nullptr) {
        // Process current->data
        current = current->next;
    }
    ```

* **Stack Operations (Array Implementation):**

    * `push(x)`: `array[top++] = x;`

    * `pop()`: `return array[--top];`

* **Queue Operations (Array Implementation - Circular Queue):**

    * `enqueue(x)`: `array[rear] = x; rear = (rear + 1) % size;`

    * `dequeue()`: `x = array[front]; front = (front + 1) % size; return x;`

4. Practical Applications

* **Arrays:** Storing and accessing collections of data (e.g., student records, sensor readings).

* **Linked Lists:** Implementing dynamic data structures where the size is not known in advance (e.g., playlists, browser history).

* **Stacks:** Function call management, undo/redo functionality, expression evaluation (postfix notation).

* **Queues:** Managing print jobs, buffering data streams, breadth-first search algorithms.

* **Strings:** Text processing, pattern matching, data manipulation.

5. Summary

Unit 1 introduced the fundamental concepts of data structures and algorithms. We explored basic data structures like arrays, linked lists, stacks, and queues, examined their properties and operations, and discussed their practical applications. Algorithm analysis, using Big O notation, was introduced as a way to measure the efficiency of algorithms. These foundational concepts will be essential for understanding and implementing more complex data structures in subsequent units.