20-11-25

NAME/MATRIC: Famuwagun Olajide . I — 230502025
DEPT.: Computer Science
Course: CSC 301 → Data Structures

ASSIGNMENT

Questions → Exercise 5 -

4. Write short answers to:
(i) What is the difference between arrays and Linked List?
(ii) What is the time complexity of insertion in a linked list?

5. ~~Laboratory Work (To be done on the laptop)~~

6. Discussion Questions;
(a.) What are the key differences between primitive data types and ADTs?
(b.) Why are arrays considered static and linked list dynamic?
(c.) In what situations would you prefer a linked list over an array?
(d.) Give real-world examples where each of the following would be useful:
* Stack
* Queue
* Linked list

Answers →

4 (i)

FEATURE	Array	Linked List
Memory storage	Contiguous	Non-Contiguous
Size	Fixed (Static)	Dynamic
Insertion/Deletion	O(n) (shifting required)	O(1) if pointer is known
Access by index	O(1) - Random access	O(n) - Sequential access only
Memory Overhead	LOW	High (stores pointer(s))

(ii) — Insert at beginning: O(1)
- Insert at end: O(1) with tail pointer or O(n) (without tail pointer)
- Insert at a specific position: O(n) (to reach the position) + O(1) actual insertion

6 (a.)

Primitive Data Types	Abstract Data Types
* Built into the language (int, float, char, etc.)	User/programmer-defined

* Only store raw values	Store data + define operations/behaviour
* Fixed operation by language	Operations defined by the designer
Implementation is visible	Implementation is hidden (abstraction)
Examples: int, double, boolean	Examples: Stack, Queue, List, Map

(b.) * Arrays are static because their size must be known and fixed at creation (in most languages). Once Created, you cannot efficiently resize them.
* Linked lists are dynamic because nodes can be added or removed at runtime without re-allocating the entire structure. Memory is allocated individually for each node.

(c.) * Frequent insertions/deletions at the beginning or middle
* Size of the Collection is ~~known~~ unknown or changes drastically.
* Memory is fragmented and contiguous block is hard to allocate
* You need efficient splitting/merging of lists
* Implementing structures like stacks/queues where only ends are accessed.

(d.)

Data structure	Real-World Examples
i. Stack	→ Undo/Redo feature in editors → Browser back button → Function call stalks
ii. Queue	→ Printer job scheduling → CPU task scheduling (round-robin) → Customer service lines
iii. Linked List	→ Music playlist (insert/delete songs easily) → Image viewer (next/previous) → Representing polynomials.

Exercise 4:-

1) Analyzing time complexity;
Time Complexity: O(n)
→ We visit each element exactly once.
Space Complexity: O(1)
→ Only one extra variable total is used, regardless of input size.

2) Trace how Linked List insertion at the head works using diagram.

Initial state (Before insertion):

Head → [50] → [30] → [20] → [15] → null

Step-by-step Insertion at Head:

(i) Create the new node:

~~head~~

[10] → None

(ii) Set the new node's next pointer to current head:

~~head~~

[10] → [50] → [30] → [20] → [15] → None

(iii) Update the head to point to the new node:

Head → [10] → [50] → [30] → [20] → [15] → None

Final State (After Insertion):

Head → [10] → [50] → [30] → [20] → [15] → None

Visual Diagram (Before Insertion):

Head

[50] → [30] → [20] → [15] → NULL

After Insertion 10 at head:

Head

[10] → [50] → [30] → [20] → [15] → null

Time complexity of insert at head: O(1)

Space Complexity: O(1)