Name: Deepali Kumari

Enrolment ID:2301010034

**Assignment 1 (Sem2) Data Structures**

**Comparison of Linked Lists and Dynamic Arrays**

**Time Complexity**

**Singly Linked List Methods:**

size: O(1)

isEmpty: O(1)

append: O(n)

\_\_str\_\_: O(n)

addFirst: O(1)

addAt: O(n)

removeFirst: O(1)

removeLast: O(n)

getHead: O(1)

removeAt: O(n)

mid: O(n)

search: O(n)

reverse: O(n)

mergeTwoSorted: O(n + m)

rotateByK: O(n)

splitByIndex: O(n)

interleave: O(n + m)

**Dynamic Array Methods:**

insertAt: O(n)

deleteAt: O(n)

size: O(1)

isEmpty: O(1)

rotateByK: O(n)

reverse: O(n)

append: O(1)

prepend: O(n)

merge: O(n + m)

interleave: O(n + m)

mid: O(1)

search: O(n)

splitAt: O(n)

\_\_str\_\_: O(n)

**Space Complexity**

**Singly Linked List Methods:**

\_\_init\_\_: O(1)

size: O(1)

isEmpty: O(1)

append: O(1) for the node, O(n) for the whole list

\_\_str\_\_: O(n)

addFirst: O(1) for the node, O(n) for the whole list

addAt: O(1) for the node, O(n) for the whole list

removeFirst: O(1) for the node, O(n) for the whole list

removeLast: O(1) for the node, O(n) for the whole list

getHead: O(1)

removeAt: O(1) for the node, O(n) for the whole list

mid: O(1) for the pointer, O(n) for the whole list

search: O(1) for the pointer, O(n) for the whole list

reverse: O(1) for the pointer, O(n) for the whole list

mergeTwoSorted: O(n + m)

rotateByK: O(1) for the pointer, O(n) for the whole list

splitByIndex: O(n)

interleave: O(n + m)

**Dynamic Array Methods:**

insertAt: O(1) for the element, O(n) for the whole array

deleteAt: O(1) for the element, O(n) for the whole array

size: O(1)

isEmpty: O(1)

rotateByK: O(1) for the pointer, O(n) for the whole array

reverse: O(1) for the pointer, O(n) for the whole array

append: O(1) for the element, O(n) for the whole array

prepend: O(1) for the element, O(n) for the whole array

merge: O(n + m)

interleave: O(n + m)

mid: O(1)

search O(1) for the pointer, O(n) for the whole array

splitAt: O(n)

\_\_str\_\_: O(n)

**Advantages and Disadvantages of Each Data Structure**

**Singly Linked List:**

Advantages:

1. Dynamic Size: Linked lists can easily grow and shrink in size by allocating and deallocating memory at runtime.

2. Ease of Insertion/Deletion: Adding or removing elements is straightforward, especially at the beginning of the list, with a time complexity of O(1).

3. No Pre-allocation: Memory is allocated as needed, so there's no need to reserve a large block of memory beforehand.

Disadvantages:

1. Access Time: Accessing an element by index requires O(n) time, as you have to traverse the list from the beginning.

2. Memory Overhead: Each node requires extra memory to store pointers (references) to the next node, leading to higher memory usage compared to arrays.

3. Cache Performance: Linked lists have poor cache locality, making them slower compared to arrays for sequential access due to non-contiguous memory allocation.

**Dynamic Array:**

Advantages:

1. Access Time: Direct access to elements by index is O(1), making it fast for random access.

2. Cache Performance: Dynamic arrays have good cache locality because elements are stored in contiguous memory locations, which improves performance for sequential access.

3. Amortized Insertion: While individual insertions at the end can be O(n) when resizing, on average, insertions at the end are O(1) due to amortized resizing.

Disadvantages:

1. Resizing Overhead: When the array grows beyond its capacity, it must be resized, which involves allocating a new array and copying elements, resulting in O(n) time complexity for that operation.

2. Insertion/Deletion Costs\*\*: Inserting or deleting elements at arbitrary positions requires shifting elements, leading to O(n) time complexity.

3. Fixed Capacity: Although dynamic arrays can resize, they require anticipating the resizing factor, which can lead to either wasted space or frequent resizing operations.

Conclusion

Both singly linked lists and dynamic arrays have their own strengths and weaknesses, making them suitable for different types of applications. Linked lists are better suited for applications where frequent insertions and deletions are required, especially at the beginning or middle of the list. Dynamic arrays excel in scenarios where random access to elements is frequent and memory reallocation is manageable. Understanding the specific needs of an application will guide the choice between these two data structures.