**Linked List** v/s **Dynamic Array**

Time Complexity

|  |  |  |
| --- | --- | --- |
| Operation | Linked List | Dynamic Array |
| Access (random) | O(n) - need to traverse | **O(1)** - Direct access by index |
| Search | O(n) - Linear search | **O(n)** - Linear search |
| Insertion (beginning) | O(1) - update head | **O(n)** - Shift elements |
| Insertion (end) | O(1) - with tail pointer | **O(n)** - Might reallocate |
| Insertion (at Index) | O(n) - find insertion point | **O(n)** - Shift elements |
| Deletion (beginning) | O(1) - update head | **O(1)** - Shift elements if needed |
| Deletion (end) | O(n) - find last node | **O(1)** - Decrement size |
| Deletion (at Index) | O(n) – find deletion point | **O(n)** - Shift elements |
|  |  |  |

Space Complexity

|  |  |
| --- | --- |
| Data Structure | Space Complexity |
| Linked List | O(n) |
| Dynamic Array | O(1) amortized |

\***amortized** = average cost of an operation

**Linked List:**

**Advantages:**

* **Efficient insertions and deletions:** Especially for insertions/deletions at the beginning or end, linked lists are faster due to constant-time updates to pointers.
* **Dynamic size:** Linked lists don't require pre-defining the size, making them suitable for situations where the data size is unknown beforehand.

**Disadvantages:**

* **Slower random access:** Finding a specific element requires traversing the list, leading to linear search time (O(n)).
* **Memory overhead:** Each node has a pointer, which adds some memory overhead compared to dynamic arrays.

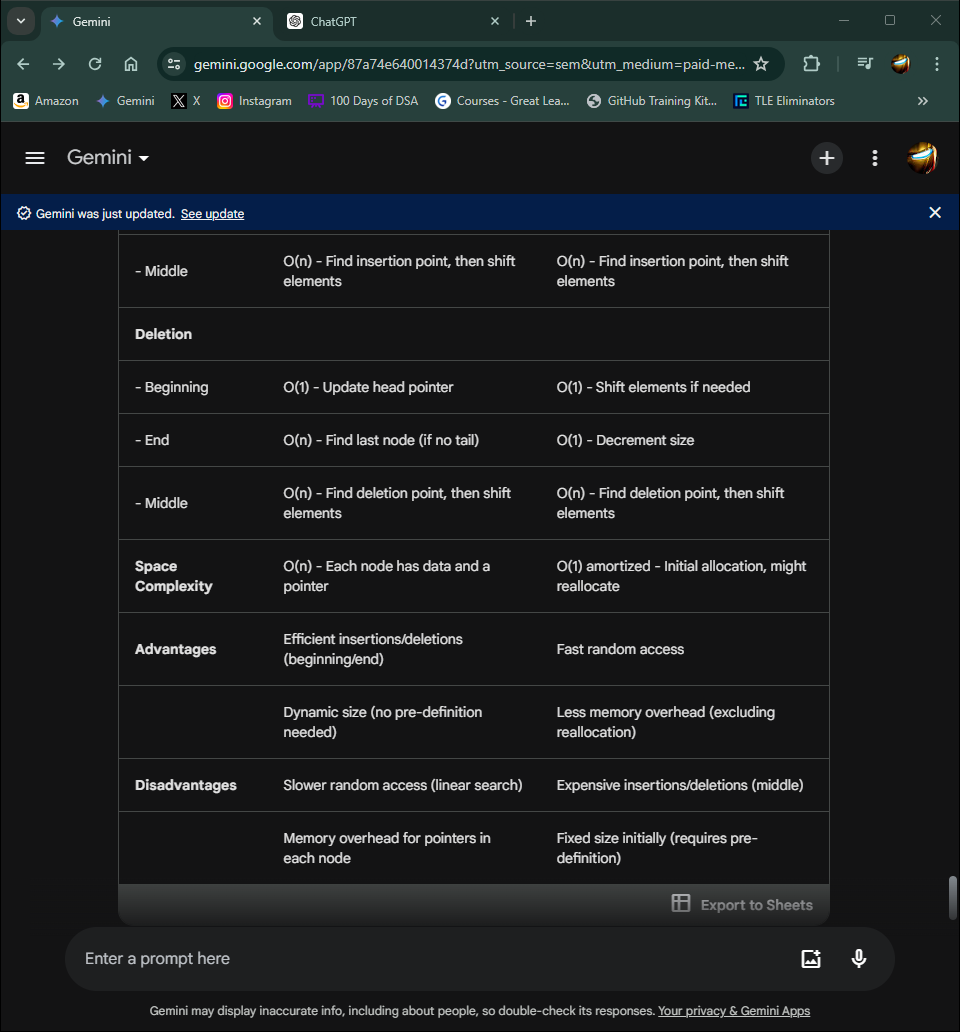
**Dynamic Array:**

**Advantages:**

* **Fast random access:** Accessing elements by index is very efficient, offering constant-time retrieval (O(1)).
* **Less memory overhead:** Dynamic arrays don't have additional pointers per element, potentially using less memory compared to linked lists (excluding reallocation overhead).

**Disadvantages:**

* **Expensive insertions/deletions (especially in the middle):** Shifting elements to accommodate insertions or deletions in the middle can be time-consuming, leading to O(n) complexity.
* **Fixed size (initially):** You need to specify an initial size for a dynamic array, which might be inefficient if the data size is unknown or highly variable.



**Choosing the Right Data Structure:**

* **Frequent insertions/deletions (especially at the beginning/end):** Use Linked Lists
* **Random access to elements is a priority:** Use Dynamic Arrays
* **Data size is unknown beforehand, and dynamic resizing is crucial:** Use Linked Lists
* **Memory usage is a concern, and frequent reallocations are unlikely:** Use Dynamic Arrays
* **You have a good estimate of the initial data size:** Use Dynamic Arrays