***Data Structure***

i)***Dynamic Array***

* Insert: O(n)
* Delete: O(n)
* Size: O(1)
* IsEmpty: O(1)
* Rotate: O(n)
* Reverse: O(n)
* Append: O(1)
* Prepend: O(n)
* Merge: O(m + n)
* Interleave: O(m + n)
* Middle: O(1)
* IndexOf: O(n)
* Split: O(n)
* Resize: O(n)
* Display: O(n)

ii)***Singly linked list***:

* Append at First: O(1)
* Append at Last: O(1)
* Append at Specific Index: O(n)
* Middle Node: O(n)
* Delete First: O(1)
* Delete Last: O(n)
* Delete at Specific Index: O(n)
* Is Empty: O(1)
* Reverse: O(n)
* Merge Two Lists: O(m + n)
* Rotate: O(n)
* Display: O(n)

Advantage of dynamic array:

1. **Dynamic Sizing:** Adjusts size automatically for efficient memory use.
2. **Random Access:** Fast access to items by position.
3. **Cache Friendliness:** Stores items together for quick retrieval.
4. **Ease of Use:** Simple to work with, often built into programming languages.
5. **Efficient Addition:** Quick adding of items, even with resizing.
6. **Flexibility:** Adapts to different data sizes without waste.
7. **Reduced Overhead:** Uses less extra memory compared to other structures.
8. **Advanced Operations:** Built-in tools for sorting, searching, and more.

**Disadvantage of Dynamic Array:**

1. **Expensive Resizing**: Enlarging a dynamic array can slow down performance (O(n)).
2. **Memory Overhead**: Dynamic arrays often use more memory than necessary.
3. **Costly Insertions/Deletions**: Adding or removing items can be slow O(n).
4. **Limited Flexibility**: Less flexible than linked lists for insertions/deletions.
5. **Potential for Waste:** Allocated capacity may exceed actual data size, wasting memory.

**Advantage of linked list:**

1. **Efficient Memory Use**: Linked lists adjust their size without wasting memory, as they grow and shrink dynamically.
2. **Fast Insertions/Deletions:** Adding or removing items is quick, especially at the beginning or end of the list**.**
3. **No Fixed Size Requirement**: Linked lists don't need a predefined size and can expand as needed.
4. **Memory Utilization:** They use memory effectively even if it's fragmented, as they don't require continuous allocation.
5. **Ease of Implementation for Complex Structures**: Linked lists are great for implementing stacks, queues, and similar structures**.**
6. **Scalability:** They handle large datasets well, adapting smoothly to size fluctuations.
7. **Memory Management Control:** Provides control over memory allocation, crucial for specific programming tasks**.**

**Disadvantages of Linked List:**

1. **Inefficient Random Access:** Accessing elements by index needs to go through the list, leading to O(n) time complexity for random access.
2. **Memory Overhead:** Each node in a linked list requires extra memory for pointers, increasing memory usage compared to arrays.
3. **Cache Inefficiency:** Non-contiguous memory storage leads to poorer cache performance compared to arrays.
4. **Costly Search Operations:** Searching for an element requires traversing the list, resulting in O(n) time complexity.
5. **Lack of Efficient Slicing:** Extracting sublists involves traversing the list, making it less efficient than array slicing.
6. **Space Overhead for Pointers:** Additional memory is needed for storing pointers, especially in large lists.
7. **Fragile Pointers:** The list's integrity relies on correct pointer manipulation, which can lead to bugs and memory leaks if mishandled.
8. **No Constant-Time Size Retrieval:** Determining the size requires traversing the entire list, resulting in O(n) time complexity.
9. **No Built-in Language Support:** Many languages lack built-in support for linked lists, requiring manual implementation.
10. **Less Efficient for Certain Operations:** Linked lists may be less efficient for algorithms requiring heavy random access or efficient slicing compared to arrays.

**Conclusion:**

*The choice between dynamic arrays and linked lists depends on the specific needs of the application:*

* *Dynamic Arrays: Better suited for scenarios requiring frequent random access, efficient memory usage, and predictable time complexity for most operations. They are ideal when the size of the data structure is known in advance or does not change frequently. Additionally, dynamic arrays are simpler to implement and offer built-in language support in many programming languages.*
* *Linked Lists: Excel in scenarios where frequent insertions and deletions are needed, especially at the beginning or middle of the list. They are flexible in size and memory usage, making them suitable for dynamic data sets with unpredictable growth patterns. Linked lists are also efficient for implementing certain data structures like stacks and queues.*

*In conclusion, neither data structure is universally better than the other. The choice should be based on the specific requirements of the application, considering factors such as access patterns, frequency of insertions/deletions, memory constraints, and performance considerations.*