Comparison of Linked List and Dynamic arrays

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Array  Functions | Time  Complexity | Space  Complexity | Linked List  Functions | Time  Complexity | Space  Complexity |
| \_\_len\_\_ | O(1) | O(0) | size | O(1) | O(0) |
| \_\_str\_\_ | O(n) | O(0) | \_\_str\_\_ | O(n) | O(0) |
| \_\_resize | O(n) | O(n) | \_\_iter\_\_ | O(n) | O(0) |
| isEmpty | O(1) | O(0) | isEmpty | O(1) | O(0) |
| append | O(1) | O(1) | append | O(1) | O(1) |
| addAt | O(n) | O(1) | addAt | O(n) | O(1) |
| pop | O(1) | O(0) | pop | O(1) | O(0) |
| removeAt | O(n) | O(0) | removeAt | O(n) | O(0) |
| rotateOnce | O(n) | O(0) | rotateOnce | O(n) | O(0) |
| rotate | O(n2) | O(0) | rotate | O(n2) | O(0) |
| reverse | O(n) | O(0) | reverse | O(n) | O(0) |
| prepend | O(n) | O(1) | prepend | O(1) | O(1) |
| merge | O(n) | O(0) | merge | O(n) | O(0) |
| interleave | O(n) | O(0) | interleave | O(n) | O(0) |
| midElement | O(1) | O(0) | midind | O(n) | O(0) |
| find | O(n) | O(0) | find | O(n) | O(0) |
| split | O(n) | O(0) | split | O(n) | O(0) |
| setResizeFactor | O(1) | O(0) | setHead | O(1) | O(0) |
| \_\_isvalid | O(1) | O(0) | \_\_next\_\_ | O(1) | O(0) |
| \_\_getitem\_\_ | O(1) | O(0) | removeFirst | O(1) | O(0) |
| \_\_setitem\_\_ | O(1) | O(0) | getHead | O(1) | O(0) |

Linked List:

Advantages:

1. It is useful when we don’t have continuous memory use.
2. They can grow to any size we want without consuming extra space.
3. The insertion and deletion can be done easily without any element shifting.

Disadvantages:

1. Elements can’t be accessed directly but only through sequential moving through every element leading in O(n) complexity.
2. The implementation of LinkedList can be difficult considered to array due to references and pointers.

Dynamic array:

Advantages:

1. Array elements can be accessed directly which results in O(1) complexity.
2. Elements are arranged continuously which helps in better cache memory performance and execution speed.
3. They are simpler to implement compared to LinkedList.

Disadvantages:

1. They still have certain fixed size and when the size exceeds, it should be resized which can be costly.
2. Inserting and deleting elements at any index can be costly and leads to O(n).
3. Resizing can lead to inefficient memory allocation as newer arrays will have most of the continuous space empty.

Summary

|  |  |  |
| --- | --- | --- |
| Feature | Linked List | Dynamic Array |
| Size Flexibility | Dynamic, grows/shrinks as needed | Dynamic, but requires resizing |
| Memory Usage | More memory per element (pointers) | Contiguous memory, potential wastage |
| Access Time | O(n) (sequential access) | O(1) (direct access) |
| Insertion/Deletion | Efficient if position is known | Costly due to shifting elements |
| Implementation | More complex | Simpler |
| Performance | Poor cache performance | Better cache performance |