

|  |
| --- |
| DSA  Time and Space Complexity PWIOI DATE = 20\05\2023     * Time Complexity   Time complexity refers to the amount of time an algorithm takes to run as a function of the length of its input. It is typically expressed using Big O notation, which provides an upper bound on the growth rate of an algorithm's running time relative to the size of the input.  For example, if an algorithm has a time complexity of O(n), where n is the size of the input, it means that the algorithm's running time grows linearly with the size of the input. If the input size doubles, the running time of the algorithm will also approximately double.   * Space Complexity   Space complexity refers to the amount of memory an algorithm uses as a function of the size of its input. Similar to time complexity, it is also expressed using Big O notation, providing an upper bound on the amount of memory used by an algorithm relative to the size of the input.  For example, if an algorithm has a space complexity of O(n), it means that the amount of memory used by the algorithm grows linearly with the size of the input. |
| Page 1 |

|  |  |  |
| --- | --- | --- |
| SPACE COMPLEXITY   * LINKED LIST * DYNAMIC ARRAY | | TIME COMPLEXITY   * LINKED LIST * DYNAMIC ARRAY |
| TIME COMPLEXITYSPACE COMPLEXITY  * LINKED LIST   In a linked list, each element (or node) contains two parts: the data and a reference (or pointer) to the next node in the sequence. Therefore, the space complexity of a linked list primarily depends on the number of nodes it contains.  Each node in a linked list stores the actual data element.  The space required for data storage depends on the size and type of the data being stored. For example, if each node stores an integer, the space required for data storage would be fixed per node.  Overall, the space complexity of a linked list is O(n), where n is the number of nodes in the list. This is because the memory required for storing the data and pointers grows linearly with the size of the list.   * **Dynamic Arrays (or Resizable Arrays):**   In a dynamic array, elements are stored in a contiguous block of memory, similar to a traditional array. However, dynamic arrays have the ability to resize themselves dynamically as elements are added or removed. Let's analyze the space complexity of dynamic arrays:  When a dynamic array is created, it typically starts with an initial capacity, which determines the size of the underlying array.  The space complexity for the initial allocation is O(n), where n is the initial capacity of the array.  Overall, the space complexity of a dynamic array is O(n), where n is the number of elements stored in the array. This accounts for the space required for storing the actual elements as well as any additional space required for resizing and managing the array's capacity. | LINKED LIST  * Access: O(n) - Traversing the list from the beginning to the desired position takes linear time. * Search: O(n) - Similar to access, searching for an element requires traversing the list sequentially. * Insertion (at beginning): O(1) - Inserting a node at the beginning of a linked list only involves updating the head pointer. * Insertion (at end, with tail pointer): O(1) - If a tail pointer is maintained, inserting at the end also takes constant time. * Deletion: O(1) or O(n) depending on the position of the element - Deletion at the beginning or with a reference to the node can be done in constant time, while deletion elsewhere requires traversal.  DYNAMIC ARRAY  * Access: O(1) - Accessing elements by index in an array is constant-time operation. * Search: O(n) - Searching for an element requires linear time, as all elements may need to be checked. * Insertion (at end, if capacity allows): O(1) amortized - If there is space available, inserting at the end is constant time. However, if resizing is required, it can be O(n) due to copying elements to a new array. * Insertion (at other positions): O(n) - Inserting an element at a specific position involves shifting subsequent elements, resulting in linear time complexity. * Deletion: O(n) - Deleting an element involves shifting subsequent elements to fill the gap, resulting in linear time complexity for deletion in the middle. | |
| * **Advantages and Disadvantages:** * **LINKED LIST :** * Advantages:   + Dynamic size: Linked lists can easily grow or shrink in size, making them suitable for scenarios where the size of the data structure changes frequently.   + Efficient insertion and deletion: Insertion and deletion operations can be efficient, especially for large lists, as they only require updating pointers, not shifting elements.   + No need for contiguous memory: Linked lists do not require contiguous memory allocation, which can be advantageous in environments with fragmented memory. * Disadvantages:   + Poor random access: Accessing elements by index is inefficient as it requires traversing the list from the beginning.   + Extra memory overhead: Each node in a linked list requires additional memory for storing pointers, leading to higher space complexity compared to arrays.   + **Dynamic Arrays:** * Advantages:   + Random access: Dynamic arrays provide constant-time access to elements using array indices, making them suitable for scenarios where efficient random access is required.   + Better cache locality: Since elements are stored in contiguous memory locations, dynamic arrays exhibit better cache locality, resulting in faster access times compared to linked lists.   + More memory efficient: Dynamic arrays can be more memory efficient than linked lists, especially when the capacity is managed properly and there is little memory fragmentation. * Disadvantages:   + Fixed capacity: Dynamic arrays have a fixed capacity, and resizing operations can be costly in terms of time complexity.   + Costly insertion and deletion: Insertion and deletion operations in the middle of a dynamic array require shifting elements, resulting in a time complexity of O(n).   + Inefficient for large deletions: Removing a large portion of elements from a dynamic array may result in wasted memory if the array's capacity is not adjusted accordingly.   + **CONCLUSION :**   In summary, linked lists and dynamic arrays each have their own set of advantages and disadvantages. Linked lists excel in scenarios requiring dynamic size and efficient insertion/deletion, while dynamic arrays are preferred for their constant-time random access and better cache performance. Understanding the trade-offs between these data structures is essential for selecting the most appropriate one for a given application. | | |
| Page 2 | | |