**Time Complexity Analysis (Singly Linked List):**

* **Add Work (addWork)**
  + **Time Complexity: O(n)**
  + **Reason: To add a task at the end, the list must be traversed until the last node is reached.**
* **Search Work (findWork)**
  + **Time Complexity: O(n)**
  + **Reason: Each node may need to be visited once to check for the ID.**
* **Traverse All (showAll)**
  + **Time Complexity: O(n)**
  + **Reason: Each node is printed in sequence from head to tail.**
* **Delete Work (removeWork)**
  + **Time Complexity: O(n)**
  + **Reason: The node must be located first before removal. The worst case occurs when the last node is deleted.**

**Advantages of Linked Lists Over Arrays:**

* **Dynamic Size:  
  Linked lists allow flexible memory usage as elements can be added or removed without resizing or reallocating memory.**
* **Efficient Insertions/Deletions:  
  Adding or removing nodes (especially at the head) is faster than arrays, which may require shifting elements.**
* **No Memory Waste:  
  Arrays allocate fixed size upfront, which may waste memory if not fully used. Linked lists allocate memory only as needed.**
* **Ideal for Unknown Size:  
  When the number of elements isn’t known in advance, linked lists avoid the need to reallocate or guess capacity.**