**Algorithm Analysis**

Ryan Caswell, Gabe Tallman

School of Computing, University of the Fraser Valley

COMP 251 AB1

Dr. Opeyemi Adesina

October 8th, 2025

# Algorithm Complexity

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Singley Linked List | | | | Array Based List | | | |
|  | Time | | Space | | Time | | Space | |
|  | Best Case | Worst Case | Best Case | Worst Case | Best Case | Worst Case | Best Case | Worst Case |
| add(T) | O(1) | O(1) | O(1) | O(1) | O(1) | O(m) | O(1) | O(n+k) |
| add(int, T) | O(1) | O(n) | O(1) | O(1) | O(1) | O(m+n) | O(1) | O(n+k) |
| Set(int,T) | O(1) | O(n) | O(1) | O(1) | O(1) | O(m) | O(1) | O(1) |
| Remove(int) | O(1) | O(n) | O(1) | O(1) | O(1) | O(m) | O(1) | O(1) |
| Remove(T) | O(1) | O(n) | O(1) | O(1) | O(1) | O(m) | O(1) | O(1) |
| Contains(T) | O(1) | O(n) | O(1) | O(1) | O(1) | O(m) | O(1) | O(1) |
| Get(int) | O(1) | O(n) | O(1) | O(1) | O(1) | O(1) | O(1) | O(1) |
| indexOf(T) | O(1) | O(n) | O(1) | O(1) | O(1) | O(m) | O(1) | O(1) |
| subList(int,int) | O(1) | O(n) | O(1) | O(n) | O(1) | O(m) | O(1) | O(m) |

# Findings and Results

Although both can store similar types of data, the time and space complexity of reading and writing that data differs significantly. Many of their best- and worst-case complexities are similar, however the most noticeable differences appear in reading, seen in the get method, and writing, seen in the add methods.

Array based lists perform exceptionally well in workloads that favor reading more than writing. Unlike linked lists, array-based lists allow direct access to any index with no traversal, resulting in a constant time complexity for reads. They also perform efficiently when the additions are limited to the end of the array, since no shifting is required. When implemented as a stack structure, an array-based list avoids most of the potential drawbacks with only significant overhead to complexity occurring when the array must be resized. While having a larger increment can reduce the frequency of resizing, the difference between the array’s size and capacity is wasted memory, trading time efficiency for space complexity.

Singly linked lists perform particularly well in write intensive workloads, especially when data is added to the front or rear of the list. When used in structures like queues, linked lists avoid the need to traverse the array to add, remove, or modify data in the middle, making insertions and deletions more efficient. Accessing data near the end of the list means reading all the data before it, giving it a complexity of O(n). That memory overhead is also not as space efficient as an array-based list, as each piece of data needs to record not only the data, but also the memory address of the next piece of data.

Notes:

Week 3, Slide 39

Array Based (stack)

* Advantage
  + Very cheap to reference any index
  + Efficient memory storage
* Disadvantage
  + Adding gets more expensive to earlier in the array
    - Especially beginning of large array
  + Resizing array adds expense

List Based (queue)

* Advantage
  + Very cheap to add to front or back of array
  + Very cheap to reduce
* Disadvantage
  + Dealing with middle indexes gets expensive
    - Especially near end of large array
  + Memory of overhead required (head and tail)

# AI Use

* Disabled VS Code’s “inline suggestions”.
* Used ChatGPT for some debugging, but with prompts similar to “debug, but just explain the problem without giving me the solution”.
* Explaining complex examples, step by step.