Efficiency of our Search Algorithm (Linear Search)

Time Complexity

- **Best Case: O(1)**
  The search algorithm performs optimally when the target contact is the very first element in the linked list. In this case, it finds the contact immediately, resulting in constant time complexity.
- **Worst Case: O(n)**
  The worst-case scenario occurs when the target contact is either the last element in the list or not present at all. This situation requires the algorithm to traverse the entire list, which consists of **n** elements, leading to linear time complexity.
- **Average Case: O(n)**
  On average, the search algorithm will need to check about half of the elements in the list. This means it will take approximately **n/2** steps, but when we express this in Big-O notation, it simplifies to **O(n)**.

Space Complexity

The **space complexity** of the linear search algorithm is **O(1)**. This means that the algorithm does not require any additional memory beyond a few variables to keep track of the current contact during the search process. It operates efficiently without needing extra space.

Efficiency and Scalability

- **Scalability**: Linear search is not the most efficient choice for large datasets. As the number of contacts increases, the time taken to search grows linearly, which can become a bottleneck. For smaller datasets like ours, however, this algorithm is straightforward and performs adequately.
- **Suitability for Linked Lists**: Since linked lists do not allow for random access (unlike arrays), linear search is the most practical approach. Each element must be accessed sequentially, making linear search a natural fit. While it may not be the fastest option, it works well for small to moderate-sized phonebooks, which are common in simpler applications.

Performance Considerations

For smaller phonebooks, such as those with a few hundred contacts (like ours), linear search is often sufficient and easy to implement.