**Big O Notation:**

**Big O Notation** describes the upper bound of an algorithm’s time or space complexity in terms of input size n. It represents the worst-case scenario.

**Time Complexity of Searching:**

1. Linear Search: O(n)
   1. Searches linearly from first element to last element.
   2. Best case: O(1), when key is the first element in array
   3. Worst case: O(n), when key is the last element of array or not in array
   4. Average case: O(n), when key is somewhere in middle
   5. Works on both sorted and unsorted arrays, complexity remains same with both arrays
2. Binary Search: O(log2(n))
   1. Divides the array/ search space into 2 halves, and iteratively searches until key is found
   2. Works only on sorted data either in desc order or ascending order.
   3. Best case: O(1), when key is exactly in the middle of the array
   4. Worst case: O(log2(n)), when key is not in array
   5. Average case: O(log2(n)), when key is present somewhere in array
   6. If data is not sorted
      1. To sort data: O(n\*log2(n)) {any built-in sort}
      2. Binary Search: O(log2(n))
      3. Total Complexity: O(n\*log2(n))
3. In this exercise,
   1. Time complexity for linear search : O(n)
   2. Binary Search
      1. Comparator: O(1)
      2. Sort: O(n\*log2(n))
      3. Binary Search: O(log2(n))
      4. Total Complexity: O(n\*log2(n))

In this exercise, linear search is more efficient and better than binary search in both implementation and complexity ways

Since, it is not necessary that productList (for List of Ids or List of Objects) will be sorted  
To apply binary search we need to sort, which increases the complexity of binary search.

Hence, In this case, Linear Search is more Efficient.