**Exercise 2: E-commerce Platform Search Function**

**Big O Notation:**

Big O notation describes an algorithm's worst-case running time or space usage as a function of input size n, ignoring constants and lower-order terms. It helps compare algorithm efficiency and predict performance for large inputs. Examples: O(1) (constant), O(n) (linear), O(n^2) (quadratic), O(log n) (logarithmic). A runtime like 3n^2 + 5n is O(n^2), focusing on the dominant term.

**Time Complexity:**

Time complexity measures an algorithm’s runtime using Big O notation, analyzed in best case (minimum time), average case (expected time), and worst case (maximum time). Worst case is key for Big O analysis.

**Linear Search:**

Linear search checks each list element sequentially until finding the target or reaching the end.

**Java Code Snippet:**

|  |
| --- |
| **public** **static** **int** **linearSearch**(**int**[] arr, **int** target) {  **for** (**int** i = **0**; i < arr.length; i++) {  **if** (arr[i] == target) {  **return** i;  }  }  **return** -**1**;  } |

**Time Complexity:**

* Best Case: O(1), target at first position.
* Average Case: O(n), checks half the list on average.
* Worst Case: O(n), target at end or absent.
* Space Complexity: O(1).

Linear search is simple but slow for large datasets, working on sorted or unsorted arrays.

**Binary Search:**

Binary search divides a sorted array in half repeatedly, comparing the target to the middle element to eliminate half the remaining elements.

**Java Code Snippet:**

|  |
| --- |
| **public** **static** **int** **binarySearch**(**int**[] arr, **int** target) {  **int** left = **0**;  **int** right = arr.length - **1**;  **while** (left <= right) {  **int** mid = (left + right) / **2**;  **if** (arr[mid] == target) {  **return** mid;  } **else** **if** (arr[mid] < target) {  left = mid + **1**;  } **else** {  right = mid - **1**;  }  }  **return** -**1**;  } |

**Time Complexity:**

* Best Case: O(1), target at middle.
* Average Case: O(log n), halves search space each step.
* Worst Case: O(log n), target at edge or absent.
* Space Complexity: O(1).

Binary search is fast but requires a sorted array.

**Conclusion**

Linear search (O(n)) is simple and ideal for small or unsorted lists. Binary search (O(log n)) is faster for large, sorted datasets but requires sorting (O(n log n)) if unsorted. Choose linear search for single searches on unsorted data and binary search for frequent searches on sorted data, based on data size and sorting status.