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

1. **Understand Asymptotic Notation:**
   * **Explain Big O notation and how it helps in analysing algorithms.**

* Big O notation is a mathematical concept used to describe the efficiency of an algorithm. Specially it characterizes that time complexity and space complexity.
* Time complexity means how the run time of an algorithm grows relative to the size of the input and space complexity means how the memory usage of an algorithm grows relative to the size of the input.
* Big O notation provides and upper bound of the growth rate of an algorithm allowing us to analyse the worst-case scenario of its performance.
* The notation uses O which stands for “order of” followed by a function that represents how the resource consumption skills with the input size.

Examples of Big O Notations are: O(1), O(n), O(n^2), O(log n), O(n log n) etc.

* **Describe the best, average, and worst-case scenarios for search operations.**
* **Linear Search:**

**Best Case:**

**Scenario**: The element being searched for is at the first position in the list.

**Time Complexity**: O(1)

**Explanation**: Only one comparison is needed.

**Average Case:**

**Scenario**: The element being searched for e somewhere in the middle of the list.

**Time Complexity**: O(n/2), which simplifies to O(n)

**Explanation**: the search algorithm will check half of the elements before finding the target.

**Worst Case:**

**Scenario**:

The element being searched for it at the last position in the least or not present at all.

**Time Complexity**: O(n)

**Explanation**: Algorithm has to check every element in the list.

* **Binary Search:**

**Best Case:**

**Scenario**: The element being searched for is at the middle position in the list.

**Time Complexity**: O(1)

**Explanation**: Only one comparison is needed.

**Average Case:**

**Scenario**: The element being searched is at random position of the list.

**Time Complexity**: O(log n)

**Explanation**: each step of the algorithm reduces the search space by half.

**Worst Case:**

**Scenario**:

The element being searched is not present at all or the search space is repeatedly halved until only one element remains.

**Time Complexity**: O(log n)

**Explanation**: The maximum number of steps required is the number of times you can halve the least before it becomes empty.

1. **Analysis:**
   * **Compare the time complexity of linear and binary search algorithms.**

**Ans:**

* **Linear Search:**

**Algorithm:**

1. Step 1 − Start from the 0th index of the input array, compare the key value with the value present in the 0th index.
2. Step 2 − If the value matches with the key, return the position at which the value was found.
3. Step 3 − If the value does not match with the key, compare the next element in the array.
4. Step 4 − Repeat Step 3 until there is a match found. Return the position at which the match was found.
5. Step 5 − If it is an unsuccessful search, print that the element is not present in the array and exit the program.

**Time Complexity:**

**Best Case: O(1)**

The element being searched for is at the first position in the list.

**Average Case: O(n)**

The element being searched for e somewhere in the middle of the list.

**Worst Case: O(n)**

The element being searched for it at the last position in the least or not present at all.

* **Binary Search:**

**Algorithm:**

1. Step 1 − Select the middle item in the array and compare it with the key value to be searched. If it is matched, return the position of the median.
2. Step 2 − If it does not match the key value, check if the key value is either greater than or less than the median value.
3. Step 3 − If the key is greater, perform the search in the right sub-array; but if the key is lower than the median value, perform the search in the left sub-array.
4. Step 4 − Repeat Steps 1, 2 and 3 iteratively, until the size of sub-array becomes 1.
5. Step 5 − If the key value does not exist in the array, then the algorithm returns an unsuccessful search.

**Time Complexity:**

**Best Case: O(1)**

The element being searched for is at the middle position in the list.

**Average Case: O(log n)**

The element being searched is at random position of the list.

**Worst Case: O(log n)**

The element being searched is not present at all or the search space is repeatedly halved until only one element remains.

* + **Discuss which algorithm is more suitable for your platform and why.**
  + Binary Search is more suitable.

**Time Complexity :**

* Binary Search has the time complexity of O(log n), it can handle large number of data where as Linear Search has the time complexity of O(n), the time increases linearly with the large number data.
* Binary Search uses sorted array, so that the searching speed is quicker than Linear Search as it uses unsorted data.
* For faster searching in binary search, user experience is comparatively good than the linear search.