Library Management System

1. Explain linear search and binary search algorithms?

Ans.

Linear Search

Linear Search is a simple searching algorithm that checks each element in a list or array sequentially until the desired element is found or the end of the list is reached. If a match is found, the position of the element is returned, if not, the process continues until the end of the list. The best-case time complexity for linear search is O(1) when the element is found at the very beginning. However, in the average and worst-case scenarios, where the element is at the end or not present at all, the time complexity becomes O(n). Linear search is best suited for small or unsorted datasets where simplicity is more important than efficiency.

Binary Search

Binary Search is an efficient algorithm for finding a target value within a sorted list or array by repeatedly dividing the search interval in half. The search begins by comparing the target value to the middle element of the list. If the middle element matches the target, its position is returned immediately. If the target is smaller, the search continues in the left half of the list, if it’s larger, the right half is considered. This halving process continues until the target element is found or the search space is exhausted. Binary search has a best-case time complexity of O(1) when the middle element is the target, and an average and worst-case time complexity of O(log n). It is highly efficient for large, sorted datasets where search speed is critical.

1. Compare the time complexity of linear and binary search?

Ans. The linear search algorithm has a best-case time complexity of O(1) when the target element is found at the very beginning of the list. In both the average and worst-case scenarios, where the element is at the end or not present, the time complexity is O(n) since each element must be checked sequentially. Its space complexity is O(1) because it requires no additional storage aside from a few variables.

The binary search algorithm has a best-case time complexity of O(1) when the middle element of the sorted list matches the target. Its average and worst-case time complexities are both O(log n), as the search space is halved with each comparison. Like linear search, its space complexity is O(1) when implemented iteratively, as no extra memory is needed beyond a few variables. However, binary search only works on sorted data, making it much faster than linear search for large, ordered datasets.

1. Discuss when to use each algorithm based on the data set size and order?

Ans. Here’s that explanation in clean paragraph form:

Linear search should be used when the list is unsorted, as it can scan through elements sequentially without any preconditions. While it can be used on a sorted list, it is inefficient with a time complexity of O(n), making it unsuitable for large, ordered datasets. It’s a good choice for small data sets, where the simplicity of implementation outweighs performance concerns and sorting overhead isn’t justified.

In contrast, binary search is ideal when the list is already sorted, offering a much faster O(log n) search time by repeatedly dividing the search range in half. It’s particularly effective for large data sets where search efficiency is critical. However, binary search isn’t suitable for unsorted lists without first sorting the data, which can be costly for small lists where a linear search would be more practical.