Searching Algorithms

Introduction

Searching is the process of finding a particular value or record within a data structure. The efficiency of a searching algorithm depends on the data structure used and whether the data is sorted.

These algorithms operate primarily on arrays or linear memory blocks. Below is a comparison based on time and space complexity.

Algorithm	**Best Case**	**Average Case**	**Worst Case**	
Linear Search	$O(1)$	$O(n)$	$O(n)$	Unsorted array
Binary Search	$O(1)$	$O(\log n)$	$O(\log n)$	Requires sorted array
Ternary Search	$O(1)$	$O(\log_3 n)$	$O(\log_3 n)$	Requires sorted array
Jump Search	$O(1)$	$O(\sqrt{n})$	$O(\sqrt{n})$	Requires sorted array
Exponential Search	$O(1)$	$O(\log i)$	$O(\log i)$	For unbounded/infinite arrays

Table 1: Time complexity for array-based searching algorithms

Structure	**Best Case**	**Average Case**	**Worst Case**	
Singly Linked List Search	$O(1)$	$O(n)$	$O(n)$	Sequential access only; no indexing
Binary Search Tree (Balanced)	$O(1)$	$O(\log n)$	$O(n)$	$O(n)$ for unbalanced trees; efficient for sorted data
Hash Table (Separate Chaining)	$O(1)$	$O(1)$	$O(n)$	Depends on hash function and load factor
Hash Table (Open Addressing)	$O(1)$	$O(1)$	$O(n)$	Avoids pointers

Table 2: Time complexity for structure-based searching algorithms

These time complexities are not inherent properties, but rather results of how algorithms are designed and optimized for specific non-linear or pointer-based data structures such as linked lists, trees, and hash tables. By carefully structuring and managing data, we can develop algorithms that achieve these theoretical bounds.

For instance, with precise pointer manipulation, we can ensure constant-time insertions in linked lists. Maintaining balance in a binary search tree allows for logarithmic-time search and update operations. Similarly, designing efficient hash functions and managing collisions using separate chaining or open addressing enables near-constant-time lookup operations.

Thus, understanding the underlying structure and access patterns of data enables us to design searching algorithms that are both time- and space-efficient, depending on the problem's constraints and requirements.