**# CS50 Algorithms: Search Algorithms Overview**

# 1. Introduction to Search Algorithms

Search algorithms are fundamental in computer science for retrieving data efficiently from a data structure. The performance of a search algorithm is measured in terms of time complexity, often represented using:

- **Big O (O):** Represents the worst-case scenario, indicating the upper bound of an algorithm's running time.
- **Theta (Θ):** Represents the average-case scenario, showing the expected running time of an algorithm.
- **Omega (Ω):** Represents the best-case scenario, giving the lower bound of an algorithm's running time.

Understanding these notations helps in evaluating the efficiency of different algorithms under various conditions.

This document covers key search algorithms with their descriptions, pseudocode, diagrams, real-world examples, and time complexities.

---

# 2. Linear Search

## Description

Linear search is a simple searching algorithm that checks each element of the list sequentially until the target value is found or the list ends. This is useful for small datasets or unsorted lists.

## Real-World Example

Searching for a contact name in an unsorted phonebook or looking for a specific word in a printed document by reading from start to finish.

## Pseudocode

```
Algorithm LinearSearch(arr, target):
    for i from 0 to length(arr) - 1:
        if arr[i] == target:
            return i  // Found target at index i
    return -1  // Target not found
```

## Diagram

```
Given array: [3, 5, 7, 9, 11]
Target: 7

Step 1: Compare 3 with 7 (no match)
Step 2: Compare 5 with 7 (no match)
Step 3: Compare 7 with 7 (match found at index 2)
```

### Time Complexity

- **Best Case (Ω): O(1)** (when the target is found at the first position)
- **Average Case (Θ): O(n)**
- **Worst Case (O): O(n)** (when the target is at the end or not present)

---

## 3. Binary Search

### Description

Binary search is an efficient searching algorithm that works on sorted arrays by repeatedly dividing the search space in half until the target is found.

### Real-World Example

Looking up a word in a dictionary or searching for a number in a phone book by jumping to the middle and eliminating half of the remaining possibilities.

### Pseudocode

```
Algorithm BinarySearch(arr, target, low, high):
    while low <= high:
        mid = (low + high) / 2
        if arr[mid] == target:
            return mid  // Found target at index mid
        else if arr[mid] < target:
            low = mid + 1  // Search right half
        else:
            high = mid - 1  // Search left half
    return -1  // Target not found
```

### Diagram

```
Given sorted array: [1, 3, 5, 7, 9, 11]
Target: 7

Step 1: Middle element is 5 (7 > 5), search right half
Step 2: Middle element is 9 (7 < 9), search left half
Step 3: Middle element is 7 (match found at index 3)
```

### Time Complexity

- **Best Case (Ω): O(1)** (when the target is found at the first middle element)
- **Average Case (Θ): O(log n)**
- **Worst Case (O): O(log n)**

---

# 4. Other Important Search Algorithms

## 4.1 Jump Search

### Description

Jump search works on sorted arrays by jumping ahead a fixed number of steps and then performing a linear search within the block. It is more efficient than linear search but requires sorted data.

### Real-World Example

Skipping pages in a dictionary and then linearly searching within a page for a word.

### Pseudocode

```
Algorithm JumpSearch(arr, target):
    step = sqrt(length(arr))
    prev = 0
    while arr[min(step, length(arr)) - 1] < target:
        prev = step
        step += sqrt(length(arr))
        if prev >= length(arr):
            return -1
    for i from prev to min(step, length(arr)):
        if arr[i] == target:
            return i
    return -1
```

### Time Complexity

- **Best Case (Ω): O(1)**
- **Average Case (Θ): O(√n)**
- **Worst Case (O): O(√n)**

---

## 4.2 Interpolation Search

### Description

Interpolation search is an improved binary search that estimates the target's position based on values. It works best for uniformly distributed datasets.

### Real-World Example

Looking for a name in a telephone book by estimating its likely position instead of starting at the middle.

### Pseudocode

```
Algorithm InterpolationSearch(arr, target):
    low = 0, high = length(arr) - 1
    while low <= high and target >= arr[low] and target <= arr[high]:
        pos = low + ((target - arr[low]) * (high - low)) / (arr[high] - arr[low])
        if arr[pos] == target:
            return pos
        if arr[pos] < target:
            low = pos + 1
        else:
            high = pos - 1
    return -1
```

### Time Complexity

- **Best Case (Ω): O(1)**
- **Average Case (Θ): O(log log n)**
- **Worst Case (O): O(n)**

---

## 5. Summary Table of Search Algorithms

| Algorithm | Best Case (Ω) | Average Case (Θ) | Worst Case (O) |
|---|---|---|---|
| **Linear Search** | O(1) | O(n) | O(n) |
| **Binary Search** | O(1) | O(log n) | O(log n) |
| **Jump Search** | O(1) | O(√n) | O(√n) |
| **Interpolation Search** | O(1) | O(log log n) | O(n) |

This document provides a well-defined overview of fundamental search algorithms, their efficiency, real-world examples, and practical usage.

---

**End of Document**