Chapter 11: Search Algorithms

Contents

Syllabus Learning Outcomes

1.2 Fundamental Algorithms
Understand algorithms for sorting and searching methods such as insertion sort, bubble sort, quicksort, merge sort, linear search, binary search and hash table search, and use examples to explain these methods.
- 1.2.3 Implement search algorithms.
  - – Linear search
  - – Binary search
  - – Hash table search
- 1.2.4 Use examples to explain search algorithms.
- 1.2.5 Compare and describe the efficiencies of the search algorithms using Big-O notation for time complexity (worst case).
  Exclude: space complexity

2.3 Implementing Algorithms and Data Structures
Use programming language elements and constructs to implement sort and search algorithms such as insertion sort, bubble sort, quicksort, merge sort, linear search, binary search and hash table search, as well as data structures such as stacks, queues, linear linked lists and binary search trees.
- 2.3.2 Implement search programs.
  - – Linear search
  - – Binary search
  - – Hash table search

1 Introduction to Search Algorithms

A search algorithm is an algorithm that check for the existence of an element or to retrieve an element from any data structure where it is stored.

There are two common types of search algorithms.

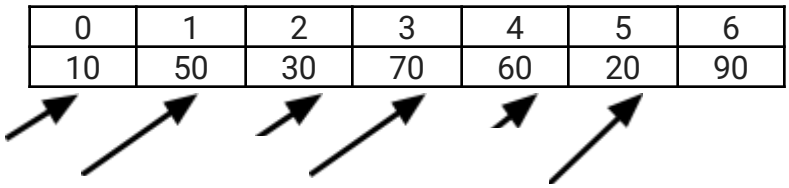
- **Linear Search**
- **Binary Search**

2 Linear Search algorithm

Linear Search is a sequential search algorithm that starts at one end and goes through each element of a list until the desired element is found, otherwise the search continues till the end of the data set.

It is one of the simplest searching algorithms and it commonly uses for-loop or while-loop to iterate through the collection.

Example: Find 20



2.1 Linear Search Algorithm in a List Using For-loop

Pseudocode of the linear search algorithm:

```
FUNCTION linear_search(A: LIST, target: INTEGER) RETURNS INTEGER
    FOR i  0 TO length of A - 1
        IF target = A[i] THEN
            RETURN i
        ENDIF
    ENDFOR
    RETURN -1
ENDFUNCTION
```

2.2 Linear Search Algorithm in a List using while-loop

Pseudocode of the linear search algorithm:

```
FUNCTION linear_search(A: LIST, target: INTEGER) RETURNS INTEGER
    i  0
    WHILE i < length of A DO
        IF target = A[i] THEN
            RETURN i
        ENDIF
        i  i + 1
    ENDWHILE
    RETURN -1
ENDFUNCTION
```

2.3 Complexity Analysis of Linear Search

In the worst case, the target might be present at the last index i.e., opposite to the end from which the search has started in the list. So the worst case complexity is O(n) where n is the size of the list.

2.4 Advantages of Linear Search

Linear search is simple to implement and easy to understand. Linear search can be used irrespective of whether the array is sorted or not. It can be used on arrays of any data type. It is a well suited algorithm for small datasets.

2.5 Drawbacks of Linear Search

Linear search has a time complexity of O(n), which in turn makes it slow for large datasets. Hence, it is not suitable for large arrays.

3 Binary Search algorithm

Binary Search is a searching algorithm used in a sorted array by repeatedly dividing the search interval in half. The idea of binary search is to use the information that the array is sorted and reduce the time complexity.

Note that to apply binary search in any data structure, the data structure must be sorted.

The basic steps to perform Binary Search are:

1. Set the low index to the first element of the array and the high index to the last element.

2. Set the middle index to the average of the low and high indices.
   - If the element at the middle index is the target element, return the middle index.
   - Otherwise, based on the value of the target element to be found and the value of the middle element, decide the next search space.
     - If the target is less than the element at the middle index, set the high index to middle index – 1.
     - If the target is greater than the element at the middle index, set the low index to middle index + 1.

3. Perform step 2 repeatedly until the target element is found or the search space is exhausted.

The Binary Search Algorithm can be implemented in the following two ways

- Iterative Binary Search Algorithm
- Recursive Binary Search Algorithm

3.1 Binary Search Using Iterative Loop

Pseudocode of Iterative Binary Search Algorithm:

```
FUNCTION binary_search_i(A: LIST, target: INTEGER) RETURNS INTEGER
    WHILE lower_bound <= upper_bound DO
        mid  (lower_bound + upper_bound)//2
        IF target = A[mid]) THEN
            RETURN mid
        ELSE
            IF target > A[mid]) THEN
                lower_bound  mid + 1
            ELSE
                upper_bound  mid - 1
            ENDIF
        ENDIF
    ENDWHILE

    RETURN -1
ENDFUNCTION
```

3.2 Binary Search using Recursive Function

The function takes in 4 parameters: a list of sorted elements, the index of the lower bound, the index of the upper bound and the target to search for.

Pseudocode of Recursive Binary Search Algorithm:

```
FUNCTION binary_search_r(A, target, lb, ub) RETURNS INTEGER
    IF lb > ub THEN
        RETURN -1
    ELSE
        mid  (lb + ub) // 2
        IF target = A[mid] THEN
            RETURN mid
        ELSE
            IF target > A[mid] THEN
                RETURN binary_search_r(A, target, mid + 1, ub)
            ELSE
                RETURN binary_search_r(A, target, lb, mid - 1)
            ENDIF
        ENDIF
    ENDIF
ENDFUNCTION
```

3.3 Complexity Analysis of Binary Search

The worst case will be when the element is present in the first position. The time complexity for the worst case is O(log n).

3.4 Advantages of Binary Search

Binary search is faster than linear search, especially for large arrays. As the size of the array increases, the time it takes to perform a linear search increases linearly, while the time it takes to perform a binary search increases logarithmically.

3.5 Drawbacks of Binary Search

We require the array to be sorted. If the array is not sorted, we must first sort it before performing the search. This adds an additional O(n* log n) time complexity for the sorting step, which makes it irrelevant to use binary search.

Binary search requires that the data structure being searched be stored in contiguous memory locations.

Binary search requires that the elements of the array be comparable, meaning that they must be able to be ordered. This can be a problem if the elements of the array are not naturally ordered, or if the ordering is not well-defined.

4 Conclusion

Linear search is a simple and flexible algorithm for finding whether an element is present within an array. It sequentially examines each element of the array and is better for being used in unsorted and small data sets.

Binary search is an efficient algorithm for finding an element within a sorted array.

However, both Linear and Binary search can be less efficient than other algorithms, such as hash tables. We will learn more about hash tables in the next chapter.

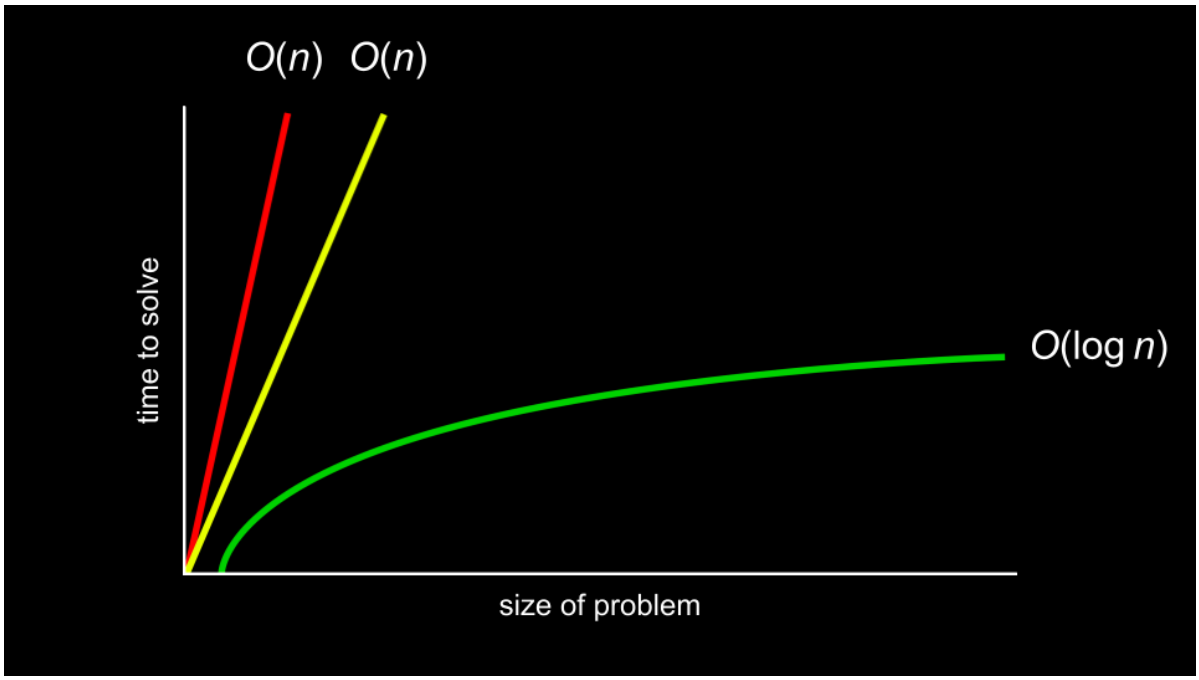
Referenced Readings

https://www.geeksforgeeks.org/

Annex

Running Time

Running time involves an analysis using big O notation. Take a look at the following graph:



In the above graph, the first algorithm is in O(n). The second is in O(n) as well. The third is in O(log n).

It's the shape of the curve that shows the efficiency of an algorithm. Some common running times we may see are:

- $O(n^2)$
- O(n log n)
- O(n)
- O(log n)
- O(1)

Of the running times above, $O(n^2)$ is considered the worst running time, O(1) is the fastest.

Linear search was of order O(n) because it could take n steps in the worst case to run.

Binary search was of order O(log n) because it would take fewer and fewer steps to run even in the worst case.
Programmers are interested in both the worst case, or upper bound, and the best case, or lower bound.

The Ω symbol is used to denote the best case of an algorithm, such as Ω (log n) .

The Θ symbol is used to denote where the upper bound and lower bound are the same, where the best case and the worst case running times are the same.