Arrays: Search

Agenda

- Why is search important?
- What is linear search?
- What is binary search?
- How do we compare linear and binary search?

Why is search important?

- Search is a way of retrieving stored data, through a query.

- Model of computation:
  - Store “useful” data in a data structure.
  - Accept queries to retrieve information.
  - Return useful data, if present.

- Need efficient search algorithms, since querying is a frequent operation.

Why is search important?

- The mechanism of search:
  - Based on comparison operation,
  - Traverse the data structure:
    - If data “matches” query parameter, stop, and return this data.
    - Else, continue traversing.

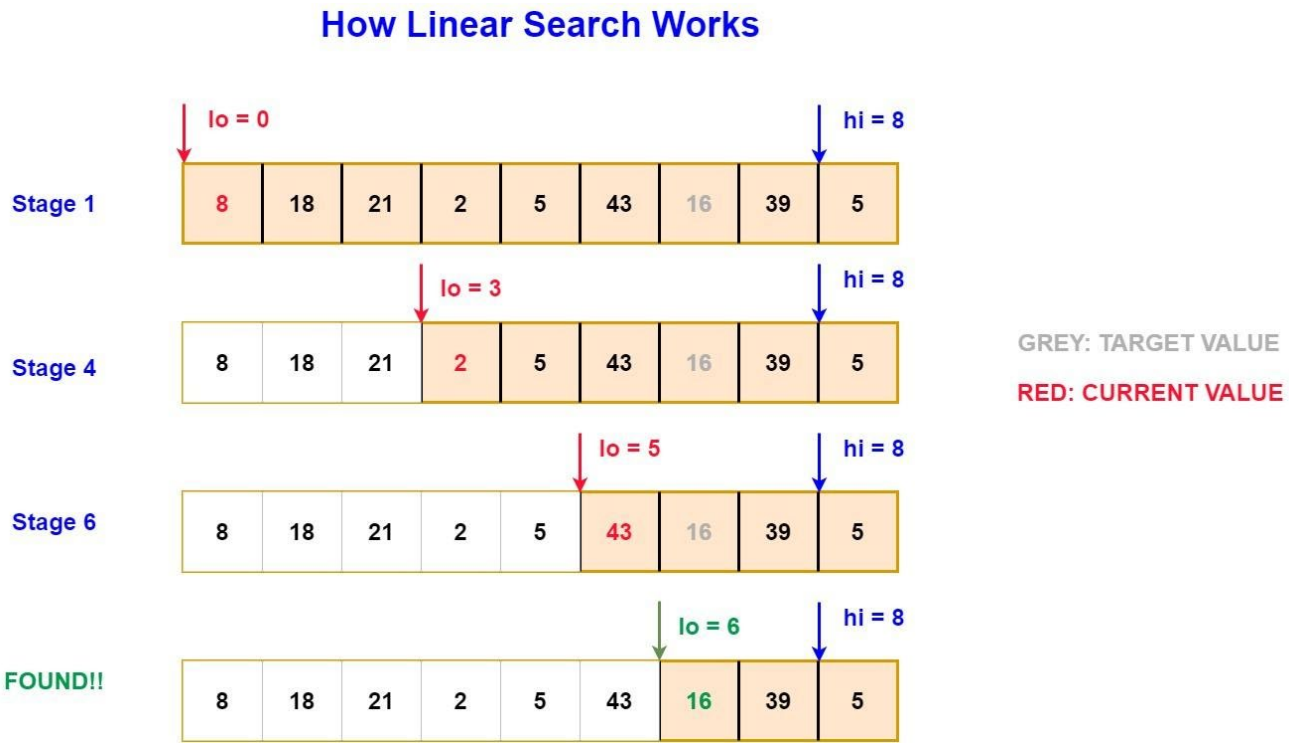
- Implementation of search:
  - Iterative: Linear search
  - Recursive: Binary search

Linear search

- Querying an array for data:
  - Simplest algorithm: Sequential / linear search
  - “Brute force”

- Iterative algorithm: Outline
  - Get the query parameter(s)
  - Start traversing the array, from the element at index 0
  - At each stage of traversal:
    - If query parameter matches the element:
      - stop, and
      - return this element - success!
    - Else, move to the element at next index
  - If reached end of array, and no match:
    - Return failure!

Linear search: Working



Linear search

Code For Linear Search

```
def linear_search(array, key):
        lo = 0
        hi = len(array)-1

        while lo <= hi:
           if key == array[lo]:

                return lo

           lo += 1

        return -1
```

Linear search

Linear Search Iterations on Array A
Target element at last index

		Array Indexes →									
lo	hi	0	1	2	3	4	5	6	7	8	
		43	39	21	18	16	8	5	5	2	← (VIOLET) Key Value
0	8	43	39	21	18	16	8	5	5	2	
1	8	43	39	21	18	16	8	5	5	2	
2	8	43	39	21	18	16	8	5	5	2	
3	8	43	39	21	18	16	8	5	5	2	
4	8	43	39	21	18	16	8	5	5	2	
5	8	43	39	21	18	16	8	5	5	2	
6	8	43	39	21	18	16	8	5	5	2	
7	8	43	39	21	18	16	8	5	5	2	
8	8	43	39	21	18	16	8	5	5	2	← (RED) Key Value Found

Linear search

Linear Search Iterations on Array A
Target element at first index

		Array Indexes									
lo	hi	0	1	2	3	4	5	6	7	8	
		43	39	21	18	16	8	5	5	2	← (VIOLET) Key Value
0	8	43	39	21	18	16	8	5	5	2	← (RED) Key Value Found
1	8	43	39	21	18	16	8	5	5	2	
2	8	43	39	21	18	16	8	5	5	2	
3	8	43	39	21	18	16	8	5	5	2	
4	8	43	39	21	18	16	8	5	5	2	Should we continue the search?
5	8	43	39	21	18	16	8	5	5	2	
6	8	43	39	21	18	16	8	5	5	2	
7	8	43	39	21	18	16	8	5	5	2	
8	8	43	39	21	18	16	8	5	5	2	

Linear search

Linear Search Iterations on Array A
Target element in the array

		Array Indexes →									
lo	hi	0	1	2	3	4	5	6	7	8	
		43	39	21	18	16	8	5	5	2	← (VIOLET) Key Value
0	8	43	39	21	18	16	8	5	5	2	
1	8	43	39	21	18	16	8	5	5	2	
2	8	43	39	21	18	16	8	5	5	2	
3	8	43	39	21	18	16	8	5	5	2	
4	8	43	39	21	18	16	8	5	5	2	
5	8	43	39	21	18	16	8	5	5	2	
6	8	43	39	21	18	16	8	5	5	2	← (RED) Key Value Found
7	8	43	39	21	18	16	8	5	5	2	Should we continue the search?
8	8	43	39	21	18	16	8	5	5	2	

Linear search: Complexity

- Analyzing time complexity
  - Best case
    - Target element is available at first index: 1 step
  - Worst case
    - Target element is available at last index: N steps
  - Average case
    - Target element is available in the middle of the array - generalizes to N/2 steps on average

Linear search: Complexity

- Linear search is an iterative algorithm, working only within the array.
  - Hence no extra space is needed.

- Analyzing space Complexity
  - Best case: Constant space
  - Worst case: Constant space
  - Average case: Constant space

Linear search: Review

- Advantages
  - If the input data is randomly ordered, it gives the the result in N steps (worst case).

- Drawbacks
  - When run on a pre-sorted array, the worst case time for linear search will still be N steps.

Linear search: Applications

- Storage scan for locating bad sectors.
- Virus scan in file system.

Binary search

- Binary search: Alternative algorithm for search.
  - Assumes that the array is pre-sorted.
  - Does not search the entire array.

- Pre-sorted array entries provides an advantage.
  - Much more efficient than linear search.

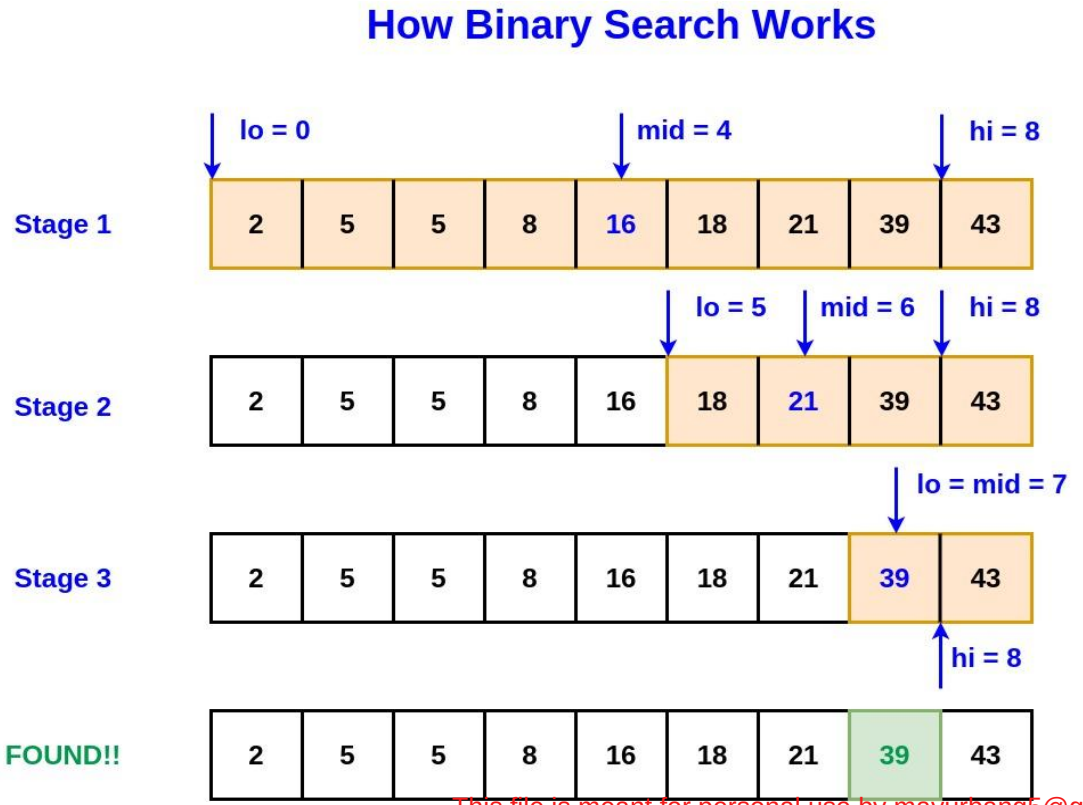
Binary search

Iterative implementation: On a pre-sorted array

1. Compare element to be searched, with array entry at the middle:
    - If they are equal, return success.
    - If the element is lesser, repeat step 1 on left-half of array.
    - If element is greater, repeat step 1 on right-half of the array.

1. If array entry not found, return failure.

Binary search



Binary search

Code For Binary Search

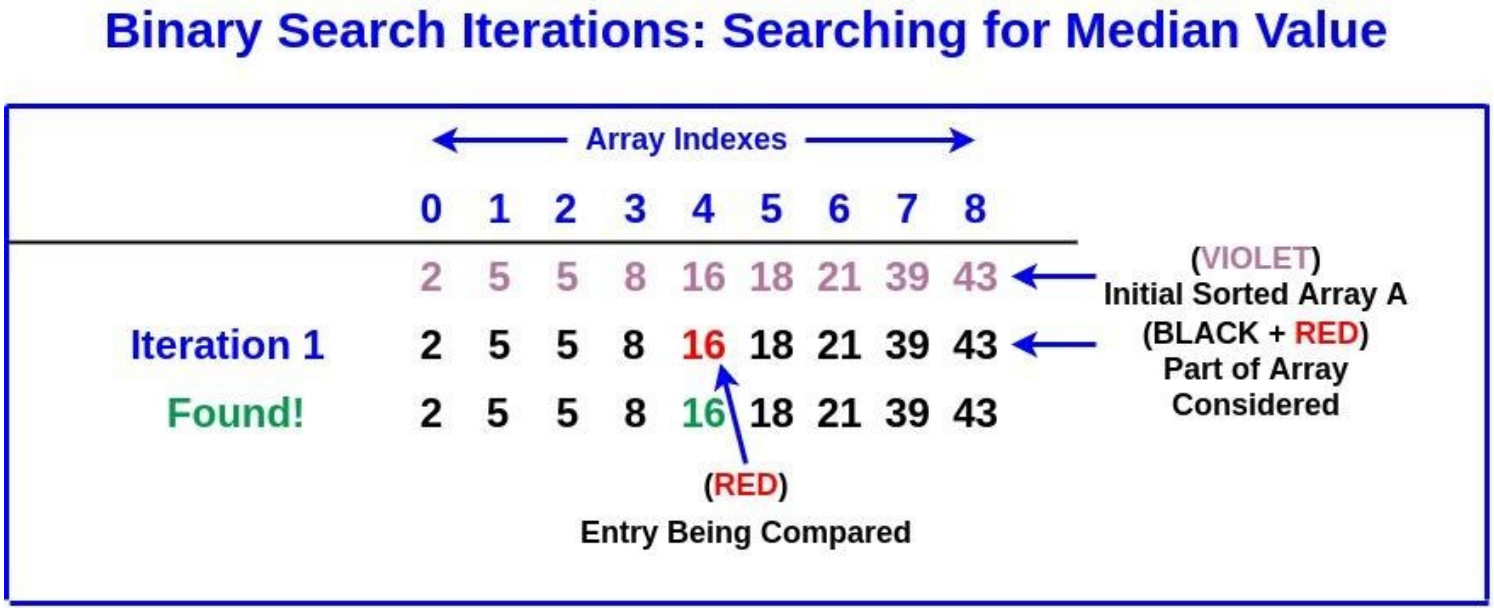
```
def binary_search(array, key):
            lo = 0
            hi = len(array)-1


        while lo <= hi:
            mid = lo + (hi - lo)/2

           if key < array[mid]:
                 hi = mid-1
           elif key > array[mid]:
                 lo = mid+1
             else:
                 return mid


           return -1
```

Binary search iterations



Binary search

Binary Search Iterations: Searching for Minimum Value

	Array Indexes →								
	0	1	2	3	4	5	6	7	8
	2	5	5	8	16	18	21	39	43
Iteration 1	2	5	5	8	16	18	21	39	43
Iteration 2	2	5	5	8	16	18	21	39	43
Iteration 3	2	5	5	8	16	18	21	39	43
Found!	2	5	5	8	16	18	21	39	43

← (VIOLET) Initial Sorted Array A

← (GREY) Not Considered

Binary search

Binary Search Iterations: Searching for Maximum Value

	Array Indexes →								
	0	1	2	3	4	5	6	7	8
	2	5	5	8	16	18	21	39	43 ← (VIOLET) Initial Sorted Array A
Iteration 1	2	5	5	8	16	18	21	39	43
Iteration 2	2	5	5	8	16	18	21	39	43
Iteration 3	2	5	5	8	16	18	21	39	43
Iteration 4	2	5	5	8	16	18	21	39	43
Found!	2	5	5	8	16	18	21	39	43

Binary search: Complexity

- Analyzing time complexity:
  - Best case
    - When the middle element of array is searched for: 1 step.
  - Worst case
    - When first/last element is searched for.
    - When element(s) just next to middle element of array is searched for .
  - Average case
- Analysis: for worst case and average case (next…).

Binary search: Complexity

- For any M, ceiling(M) = smallest integer larger than M
- Analyzing time complexity: Worst case
  - Simplifying assumption: $N = 2^k$, for some k
    - The part of the array actively searched for, is halved k times: k steps
    - k is $\log_2 N$, so $\log_2 N$ steps

Binary search: Complexity

- Analyzing time complexity: Worst case
  - Even if N is not an exact power of 2, worst case can be demonstrated to have ceiling($\log_2 N$) steps.
  - Proportional to $\log_2 N$.

- Average case:
  - We normally take the ceiling value, of a fraction of $\log_2 N$ steps.
  - Proportional to $\log_2 N$.

Binary search: Complexity

- Analyzing space complexity:
  - Binary search is an in-place, iterative search algorithm.
    - No additional space is needed.
  - Best case: Constant space (to store searched value).
  - Worst case: Constant extra space.
  - Average case: Constant extra space.

Binary search: Review

- Binary search v/s linear search
  - Advantages
    - Very efficient compared to linear search.
    - Works well for arrays that are never updated.
  - Drawbacks
    - Requires pre-sorted array to work.
    - Does not work well for arrays that are frequently updated.

Binary search: Applications

- English to english dictionary
- Telephone directory

Summary

We explored linear and binary search algorithms as follows:

- Algorithm mechanism and pseudocode
- Algorithm iterations on varying input
- Time and Space Complexity
- Applications

Thank You