Binary Search: A Fast Way to Find Things

Agenda / About this deck

First, finish our recursion activity.

Then, run through this deck very quickly. This is just focused on the code implementation of binary search (lots of overlap with Runestone).

Then, depending on time, we'll talk more conceptually about search, but also just work on project.

Introduction to Searching

- **Searching**: Imagine looking for a word in a dictionary. If you check one word at a time from the start, it would take ages for a long dictionary. Binary Search makes this faster.
- **Key Idea**: Divide the search space in half each time.

Linear Search vs. Binary Search

- **Linear Search**: O(n)
  - Check each element one-by-one.
  - Useful when data is unsorted.
- **Binary Search**: O(log n)
  - Faster, but requires sorted data.
  - Each time, cut the search space in half.

Interactive Demo: Imagine finding a word in a dictionary. How many words do you skip if you flip halfway each time?

Binary Search Intuition

- **Sorted Array Example**: [1, 3, 5, 7, 9, 11, 13]
  - Target: Find 9.
  - **Step 1**: Start with middle element (7).
  - **Step 2**: Since 9 > 7, ignore left half.
  - **Step 3**: Now search [9, 11, 13]. Middle is 11.
  - **Step 4**: Since 9 < 11, ignore right half. Find 9.

Visualization

- **Divide-and-Conquer**:
  - Use visuals to demonstrate the division of the array.
  - Show the array shrinking in size each time.
- **Pseudocode**:

```
low = 0
high = len(array) - 1
while low ⩽ high:
    mid = (low + high) // 2
    if array[mid] == target:
        return mid
    elif array[mid] < target:
        low = mid + 1
    else:
        high = mid - 1
return -1
```

Why Does It Work?

- **Core Concept**: Every decision eliminates half of the possible choices.
- **Comparison to Real Life**: Imagine finding a page in a book without an index. Flip to the middle until you narrow it down.

Practice Problem

- **Problem**: Find the number 23 in the sorted list [5, 12, 15, 23, 27, 30, 35].
- **Thinking out loud**: Can try to write out / verbalize each step.
  - "Is 23 greater or less than the middle element?"
  - Keep cutting until found.

Common Mistakes

- **Off-by-One Errors**: Carefully manage `low` and `high` boundaries.
- **Infinite Loops**: Ensure condition `low ⩽ high`.

> **Tip**: Debugging exercises can help solidify this. Run a failing example and walk through the fix.

Example mistake 1: Incorrect Mid Calculation

```
array = [1, 3, 5, 7, 9, 11, 13]
target = 9

low = 0
high = len(array) - 1
while low ⩽ high:
    mid = (low + high) / 2  # Mistake: Using `/` instead of `//` for integer division
    if array[mid] == target:
        print(f"Found target {target} at index {mid}")
        break
    elif array[mid] < target:
        low = mid + 1
    else:
        high = mid - 1
else:
    print("Target not found")
```

Mistake 2: Off-by-One Error

```
# Mistake 2: Off-by-One Error
array = [1, 3, 5, 7, 9, 11, 13]
target = 9

low = 0
high = len(array)  # Mistake: Should be `len(array) - 1`
while low ⩽ high:
    mid = (low + high) // 2
    if array[mid] == target:
        print(f"Found target {target} at index {mid}")
        break
    elif array[mid] < target:
        low = mid + 1
    else:
        high = mid - 1
else:
    print("Target not found")
```

Example Mistake 3: Infinite Loop Condition

```
array = [1, 3, 5, 7, 9, 11, 13]
target = 9

low = 0
high = len(array) - 1
while low < high:  # Mistake: Should be `low ⩽ high`
    mid = (low + high) // 2
    if array[mid] == target:
        print(f"Found target {target} at index {mid}")
        break
    elif array[mid] < target:
        low = mid + 1
    else:
        high = mid - 1
else:
    print("Target not found")
```

Fully Worked Example of Binary Search

```
def binary_search(array, target):
    low = 0
    high = len(array) - 1

    while low ⩽ high:
        mid = (low + high) // 2
        # Debug statement to show the current state of low, high, and mid
        print(f"Low: {low}, High: {high}, Mid: {mid}, Mid Value: {array[mid]}")

        if array[mid] == target:
            return mid
        elif array[mid] < target:
            low = mid + 1
        else:
            high = mid - 1

    return -1  # Target not found

# Example usage
array = [1, 3, 5, 7, 9, 11, 13]
target = 9
result = binary_search(array, target)
if result ≠ -1:
    print(f"Target {target} found at index {result}")
```

Binary Search Applications

- **Applications**:
  - Finding an element in a sorted array.
  - Searching in a phonebook.
  - Problems like "Guess the Number" (using minimum attempts).

Conclusion & Recap

- **Summary**: Binary Search cuts the search space in half each time, leading to a time complexity of O(log n).
- **Key Question**: Why does Binary Search need a sorted array?

> **Engagement**: Think of any real-life activities where a binary search-like strategy is used?

https://www.cs.usfca.edu/~galles/visualization/Search.html