Search and Root Finding: The Bisection Principle

The Bisection Method and Binary Search are powerful algorithms that rely on the same core principle: **repeatedly halving a search space to quickly converge on a target.**

1. The Bisection Method (Root Finding)

The Bisection Method is a technique used in numerical analysis to find the roots (or solutions) of equations. In the context of finding the n-th root of a number A, the equation is $\mathbf{x^n = A}$.

How it Works

The method works by defining an interval $[a, b]$ where the solution is guaranteed to exist. We then perform iterative steps:

1. **Find Midpoint:** Calculate the midpoint of the interval, $m = \frac{a+b}{2}$.
2. **Test:** Raise the midpoint to the power of n (m^n).
3. **Halve the Interval:**
    - If m^n is greater than A, the root must be in the lower half, so we set $b = m$.
    - If m^n is less than A, the root must be in the upper half, so we set $a = m$.
4. **Repeat:** The process continues until the interval $[a, b]$ is smaller than a desired precision (e.g., 0.000001).

This method is guaranteed to converge on the correct answer because the search space is cut in half with every single iteration.

Domain: Continuous (finding a precise value on a number line). **Goal:** To find a specific numerical value.

2. Binary Search (Element Finding)

Binary Search uses the exact same halving principle, but applies it to finding an element's position within a **sorted list or array**.

How it Works

For Binary Search to work, the list *must* be sorted.

1. **Find Midpoint (Index):** Determine the middle index of the current search segment.
2. **Test:** Check the value at the midpoint index against the target element.
3. **Halve the List:**
    - If the midpoint value equals the target, the search is complete!
    - If the midpoint value is greater than the target, the target must be in the left half, so we discard the right half.

- If the midpoint value is less than the target, the target must be in the right half, so we discard the left half.

4. **Repeat:** The process continues until the element is found or the search segment is empty.

Domain: Discrete (finding an index within a sorted list). **Goal:** To find the location of an element.

3. Brute Force Search (Linear Search)

The Brute Force method (or Linear Search) is the simplest way to search a list. It is the baseline against which faster algorithms are compared.

How it Works

1. Start at the first element of the list.
2. Check if the current element matches the target.
3. If it doesn't match, move to the next element.
4. Repeat until the element is found or the end of the list is reached.

Key Difference: Unlike Binary Search, Brute Force does **not** require the list to be sorted. However, in the worst case (the element is the last one or not present), it must check every single item.

4. Method Comparison (Efficiency)

The efficiency of an algorithm is measured by how the required steps (time) grow as the input size (N) increases.

Feature	Brute Force (Linear Search)	Binary Search	Bisection Method (Root)
Search Space	Entire list (no requirement to be sorted)	Sorted list only	Continuous numerical interval
Growth Rate	**Linear ($O(N)$)**: If the list doubles in size, the time roughly doubles.	**Logarithmic ($O(\log N)$)**: If the list doubles, only one extra step is needed.	**Logarithmic ($O(\log(1/\epsilon))$)**: Steps increase slowly as required precision ϵ increases.
Analogy	Checking every house on a street one by one.	Opening a phone book to the middle, checking if the name is before or after, and repeating on half the book.	Continuously zooming in on a map location.

Logarithmic/Exponential growth is incredibly fast. For a list with 1,000,000 items:

- **Brute Force:** Up to 1,000,000 checks.
- **Binary Search:** Maximum of ≈ 20 checks (2^{20} is over a million).

Comparison Examples: The Power of Halving

Imagine you are looking for a name in a **sorted** list. The "Max Checks" column shows the *worst-case* scenario—the most work the computer would ever have to do.

#	List Size (N)	Brute Force (Max Checks)	Binary Search (Max Checks)	Difference
1	32 (A class of 32 students)	32	5	Imagine asking only 5 questions to find 1 person out of 32!
2	100 (People in a movie theater)	100	7	$2^7 = 128$. Binary Search only needs 7 checks to find 1 out of 100.
3	1,000 (A neighborhood)	1,000	10	10 steps vs. 1,000 steps. That's 100 times faster!
4	10,000 (A small town)	10,000	14	You save 9,986 steps by halving.
5	100,000 (A big city)	100,000	17	Brute Force takes hours; Binary Search takes less than 20 seconds.
6	1,000,000 (A very large city)	1,000,000	20	Binary Search finds the answer in just 20 steps, which is almost instant!

5. Python Implementations

The python code demonstrates all three methods.