Algorithm Definition

- An algorithm is a set of instructions for accomplishing a task. Every piece of code could be called an algorithm.
- **Detailed definition**: An algorithm is nothing more than a step-by-step procedure for solving a problem. The algorithms you'll use most often as a programmer have already been discovered, tested, and proven. If you want to understand them but refuse to slog through dense multipage proofs, this is the book for you. This fully illustrated and engaging guide makes it easy to learn how to use the most important algorithms effectively in your own programs.

Simple search algorithm

- It's a naive searching algorithm for finding an element in a list (array). Basically it means iterating over each element in the array and checking if it's the needed element or not.

Binary search algorithm

- It is a basic searching algorithm for finding an element in a sorted list in O(log n) time. Basically, Binary Search can be used anywhere where you feel like searching an element from a list in the least possible time.
- **Requirements**: sorted data (list, set, tuple)

Steps

- The binary search function takes a sorted array and an item. If the item is in the array, the function returns its position.

> Divide the array into two parts and keep track of what part of the array you have to search through. If the item is not the middle item, check if it's higher or lower than the middle item, take the next part of the array, divide it into two parts and keep going until you find the needed item.

- **Running Time**: Binary search runs in logarithmic time (or log time, as the natives call it).
- **Example** : If the array is 100 items long, it takes at most 7 guesses. If the list is 4 billion items, it takes at most 32 guesses.

Code

In [1]:

```
def binary_search(list, item):
  # keep track of what part of the list your searching
  # initialised with the full list
    low = 0
    high = len(list) - 1

    while low <= high:
        # check the middle element
        mid = int((low + high) / 2)
        guess = list[mid]
        if guess == item:
            return mid
        if guess > item:
            high = mid - 1
        else:
            low = mid + 1
    return None

my_list = [1, 3, 5, 7, 9]

# Find the index of 3 in the list
print(binary_search(my_list, 3)) # 1

# Find the index of -1 in the list
print(binary_search(my_list, -1)) # None
```

```
1
None
```

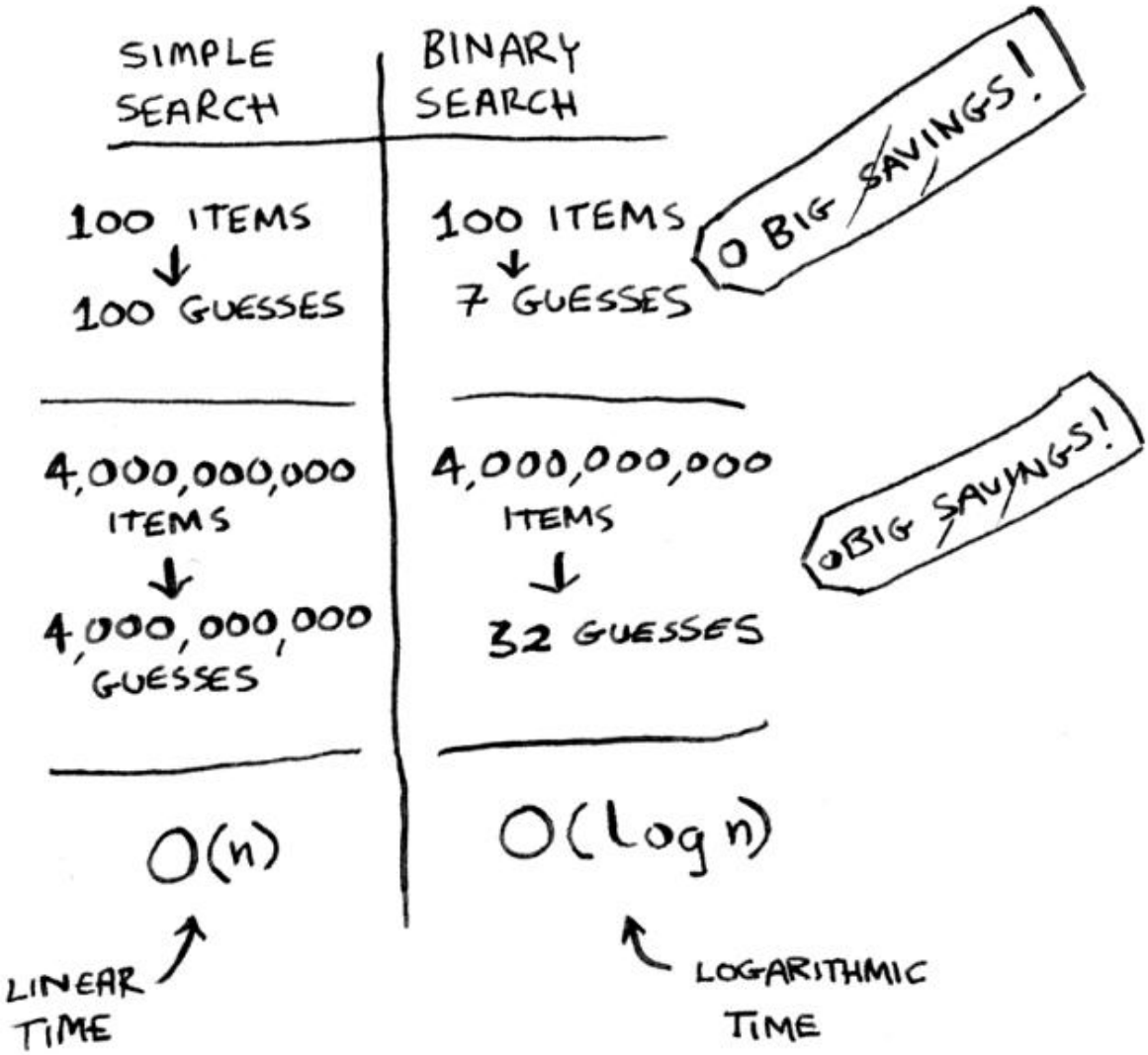
Differences between simple search and binary search

> In a simple search the time is exponential, that is, if you have a list with 100 elements, the search time is O(n). So it would take 100 tries to find the desired number.

> In a binary search, the search time is O(log n), so if you have a list with 100 elements, the search time is O(log 100), so the search time is 7 and not 100 anymore.

Example of their running Time

- Let, list = 4 billion
- Linear time - Simple Search, starting at the beginning of the list until you find the right element, could take 4 billion steps
- Logarithmic time - Finding the right element takes log 4 billion = 32 (rounded)
- **Lesson**: Algorithm run times grow at different rates. Timing two different algs is not enough because it doesn't reflect how the run time will grow at different scales.



Big O notation

the Big O tells you how fast an algorithm is. Binary lookup needs log n operations to check a list of size n. in Big O notation it would be O(log n), whereas the linear search is O(n) since they are just linear actions. She tries to take into account the worst case and the middle case to come up with a metric.

- It's called Big O notation because you put a “big O” in front of the number of operations (it sounds like a joke, but it’s true!) Big O notation is about the worst-case scenario

Runtime Examples with Big O

These are the five most common:

- O(log n), also known as logarithmic time. EX binary search
- O(n), also known as linear time. EX simple search
- O(n * log n), also known as linear log. EX quicksort sorting
- O(n2) also known as quadratic. EX sort by selection
- O(n!), also known as exponential. EX insertion sort

Practices

You have a name and want to find the phone number for that name in a phone book

- Answer : O(log n)

You have a phone number and you want to find its owner in a phone book

- Answer : O(n)

You want to read each person's number from the phone book

- Answer : O(n)

You want to read numbers only from names starting with A

- Answer : O(n)