GRAB A BYTE

BINARY SEARCH!

LINEAR SEARCH ALGORITHM

Last week we learned about the Linear Search Algorithm, which is an algorithm that searches through a list one-by-one starting at index 0

BINARY SEARCH ALGORITHM

Today we will be learning the Binary Search Algorithm. In the Grokking Book (linked in the Discord), in order to describe the Binary Search Algorithm, they use the example of searching a phone book

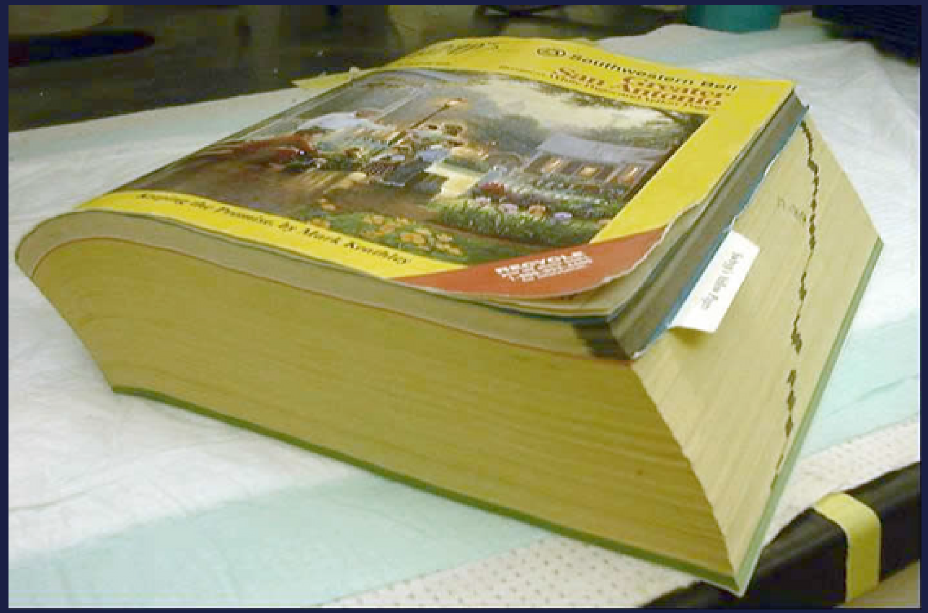
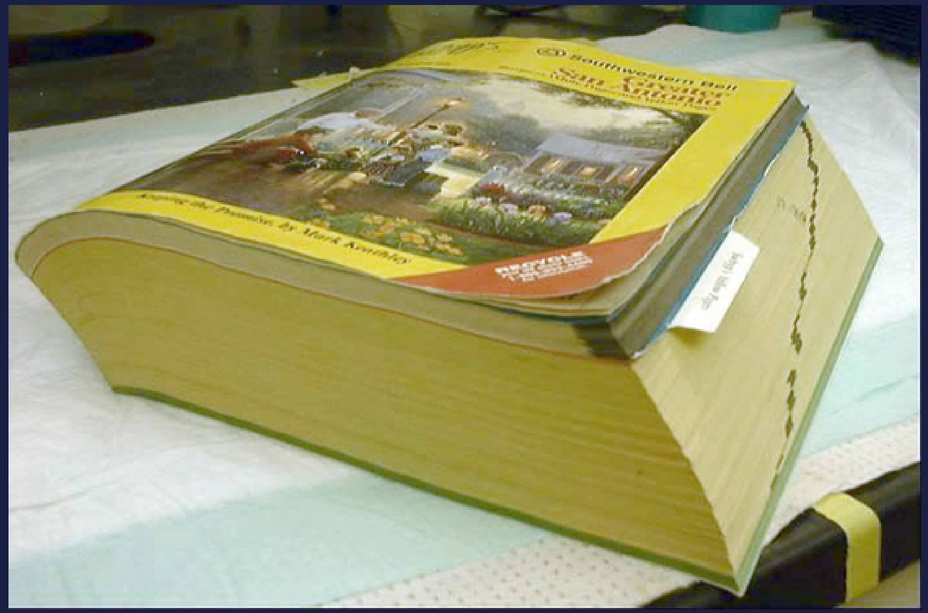


Image looking for Krispy Kreme in this phone book using Linear Search. You'd start at page 1 and look for Krispy Kreme page by Page. It would take hours... days... years...



OR you could search using a Binary Search!

If you look in the middle first and see if Krispy Kreme is higher or lower, then check the middle of that half, etc.

Lets consider an array of integer values:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

If we are looking for the number 21. in a Binary Search we would start in the middle

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24

It checks the middle value to see if it is 21, higher than 21 or lower than 21

NO! HIGHER!

If we are looking for the number 21. in a Binary Search we would start in the middle

~~0 1 2 3 4 5 6 7 8 9 10 11 12~~ 13 14 15 16 17 18 19 20 21 22 23 24

It checks the middle value to see if it is 21, higher than 21 or lower than 21

NO!
GO HIGHER!

If we are looking for the number 21. in a Binary Search we would start in the middle

~~0 1 2 3 4 5 6 7 8 9 10 11 12~~ 13 14 15 16 17 18 19 20 21 22 23 24

Now it checks the NEW middle and determineds if it is 21, higher than 21, or lower than 21

NO!

GO HIGHER!

If we are looking for the number 21. in a Binary Search we would start in the middle

~~0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18~~ 19 20 21 22 23 24

Now it checks the NEW middle and determineds if it is 21, higher than 21, or lower than 21

NO!
GO HIGHER!

If we are looking for the number 21. in a Binary Search we would start in the middle

~~0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18~~ 19 20 21 22 23 24

Now it checks the NEW middle and determineds if it is 21, higher than 21, or lower than 21

NO!

GO LOWER!

If we are looking for the number 21. in a Binary Search we would start in the middle

~~0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18~~ 19 20 21 ~~22 23 24~~

Now it checks the NEW middle and determineds if it is 21, higher than 21, or lower than 21

NO!
GO LOWER!

If we are looking for the number 21. in a Binary Search we would start in the middle

~~0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18~~ 19 20 21 ~~22 23 24~~

Now it checks the NEW middle and determineds if it is 21, higher than 21, or lower than 21

NO!
GO HIGHER!

If we are looking for the number 21. in a Binary Search we would start in the middle

~~0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20~~ 21 ~~22 23 24~~

Now it checks the NEW middle and determineds if it is 21, higher than 21, or lower than 21

NO!
GO HIGHER!

If we are looking for the number 21. in a Binary Search we would start in the middle

~~0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20~~ 21 ~~22 23 24~~

Now it checks the NEW middle and determineds if it is 21, higher than 21, or lower than 21

21 ITERATIONS VS. 4

That is a big difference, and will make a bigger difference when its 1,000 elements, 1,000,000 elements, and 1,000,000,000 elements or more!!!

SPEED...

This can be a big difference with speed the more iterations. For example, lets say we have a program that takes 1ms per iterations

Elements	Linear Search	Binary Search
100	100ms	7ms
10,000	10sec	14ms
1,000,000,000	11days	32 minutes

THE PSEUDOCODE

```
array = [1, 2, 3, ..., 98, 99, 100]
low = 0
high = len(array)-1
mid = null
target = 21

while low <= high:
    mid = array[(low + high) / 2]
    if (mid == target) return mid
    elif (mid > target) high = mid - 1
    elif low = mid + 1
    else return none
```

EXAMPLES REPLIT! PLEASE GO TO:

HTTPS://REPLIT.COM/

@RIKKIEHRHART/GRABABYTE

O NOTATION!

What is O Notation?

- aka “Big O Notation” is a way to describe how efficient an algorithm is as the size of the input grows.
- It tells us how *long* an algorithm might take or how much *work* it might need to do
- Essentially, how many steps or iterations at the *worst*

Common O Notations:

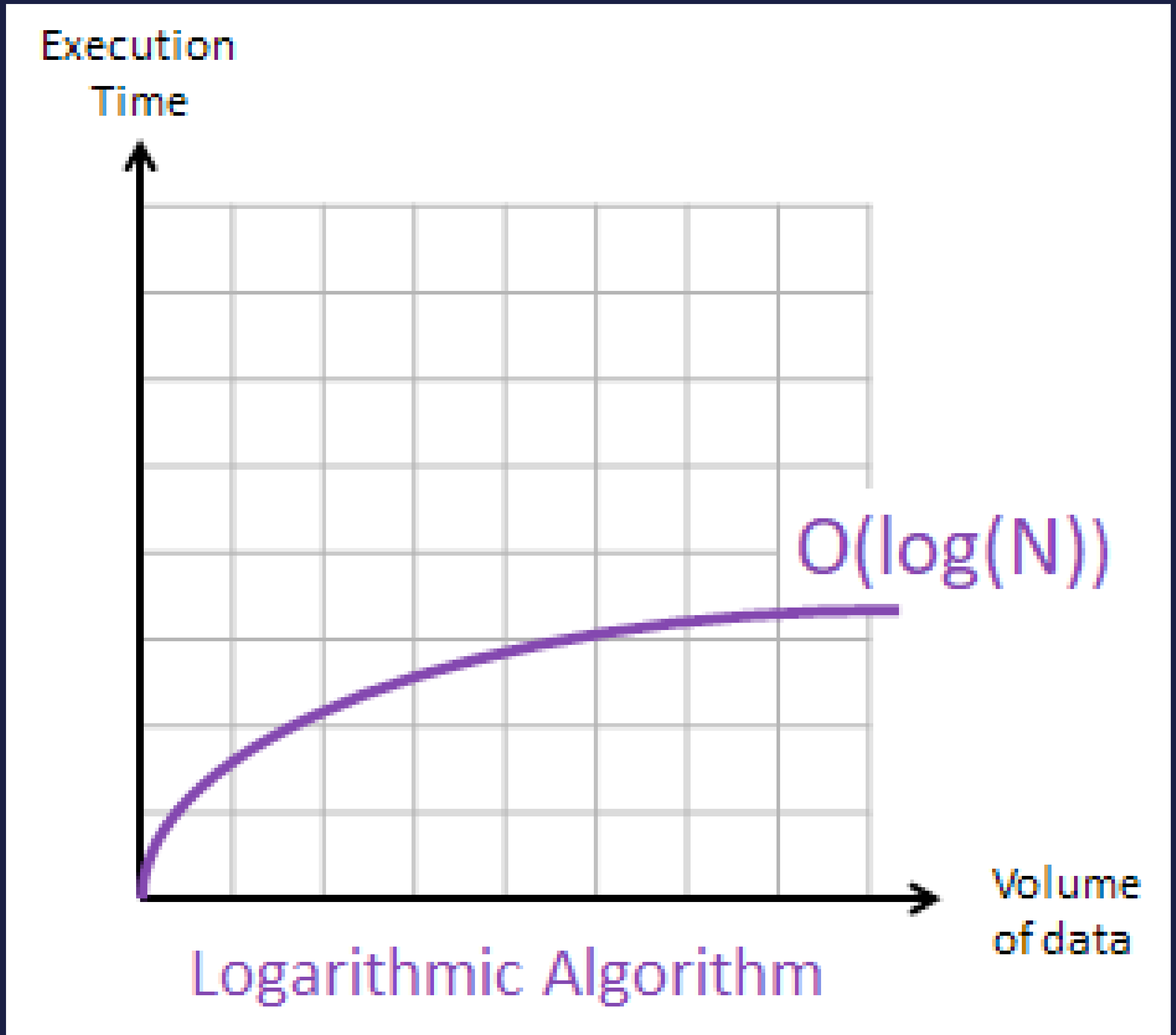
- Linear Time | O(n) - we covered in Linear Search
- Logarithmic Time | O(log n) - covering today!
- Quadratic Time | O(n^2) - cover with Bubble Sort
- Log-Linear Time | O(n log n) - cover with Merge Sort
- Constant Time | O(1) - cover with Hashing

LOGARITHMIC TIME | O(LOG N)

Logarithmic Time O Notation is how many times we can divide a problem in half until we reach the desired result.

Because it is an O Notation, it calcutes how many iterations if our desired result is in the worst position.

LOGARITHMIC TIME



Essentially, as the list gets bigger, the time it takes cuts in half each iteration.

This makes the Binary Search sooooooooooooooooooooo ooooooooooo much faster than a Linear Search

UP NEXT

Feb 12 - Bubble Sort
Feb 19 - Selection Sort
Feb 26 - Insertion Sort
Mar 5 - Merge Sort
Mar 12 - Quick Sort
SPRING BREAK!
Mar 26 - Breadth-First Search (BFS)

Apr 2 - Depth-First Search (DFS)
Apr 9 Hashing
Apr 16 - Dijkstra's Algorithm
Apr 23 - Dynamic Programming (Knapsack Problem)
Apr 30 - Union-Find
May 7 - Kruskal's Algorithm
May 14 - Prim's Algorithm

Questions? - rikki.ehrhart@g.ausitncc.edu

If you'd like the opportunity to run a Grab a Byte algorithm workshop, please let me know!