Section 4.3: Logarithmic Growth

Monday, March 04, 2024 11:05 AM

suppose you need to find the position of a particular entry in an ordered list

[12, 13, 27, 35, 52, 71, 89]

where is 52?

method #1: start at left end and look at each entry in the list in order until you get to the entry of interest

→ this method is $O(n)$ and is called a linear search

method #2: look at the entry in the middle of the list. If it's the entry of interest (52), stop!

If greater than, then discard the bottom half of the list and repeat

[12, 13, 27, (35), 52, 71, 89]

is 52 = 35 ? no

is 52 > 35 ? yes

[52, (71), 89]

is 52 = 71 ? no

is 52 > 71 ? no

[52]

is 52 = 52 ? yes!

step

for method #2, if your list has one million entries you need a maximum of 20 searches to find your entry of interest

---

Section 4.3: cont'd 2024/03/05

essentially, you are solving

$$2^n = 1\,000\,000$$

this requires a new function called a logarithm

logarithms

if $2^3 = 8$, then

$$3 = \log_2 8$$

the logarithm asks "what exponent on the base gives the other number?"

$\log_{\text{base}}$ other number

examples: $\log_2 16 = 4$ (because $2^4 = 16$)

$\log_2 2 = 1$

$\log_3 9 = 2$

$\log_{10} 1000 = 3$

$\log_{10} 0.1 = -1$

} for calculators, if the base is not specified, it's base 10

Unfortunately, this is not the universal default

in computing, the default is base e

↑ fifth letter of alphabet

$e = 2.71828182845905\ldots$

(irrational, like π and $\sqrt{2}$)

in many programming languages

log means $\log_e$

log10 means $\log_{10}$

to calculate logarithms of different base with a calculator (most calculators only do base 10 and base e):

if we want to solve $2^n = 1000000$

$n = \log_2(1000000)$

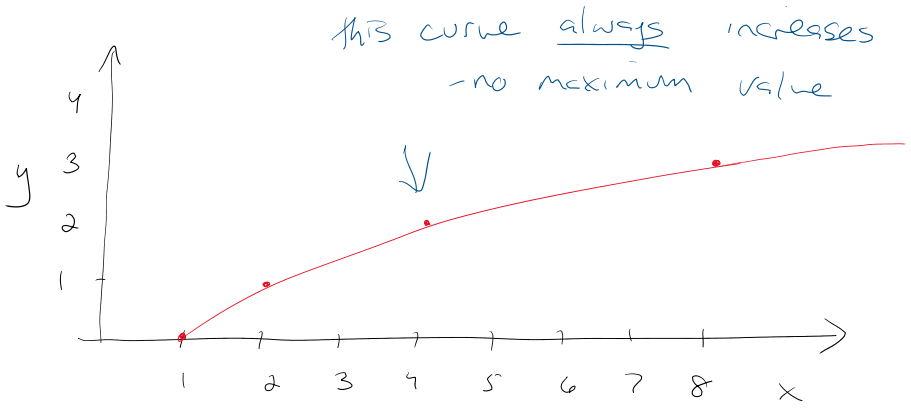
$$\log_2(1000000) = \frac{\log(1000000)}{\log 2}$$

$\log_2$

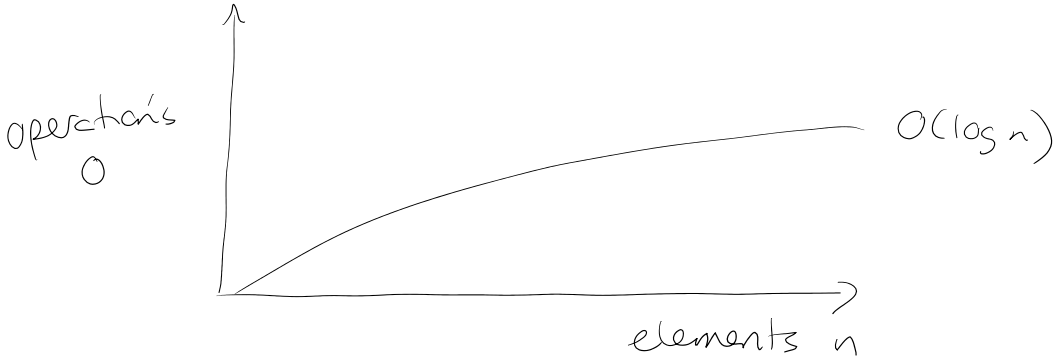
method #2 is called a binary search and is $O(\log n)$

what does the graph of a logarithm look like?

x	$y = \log_2 x$
1	0
2	1
4	2
8	3
16	4

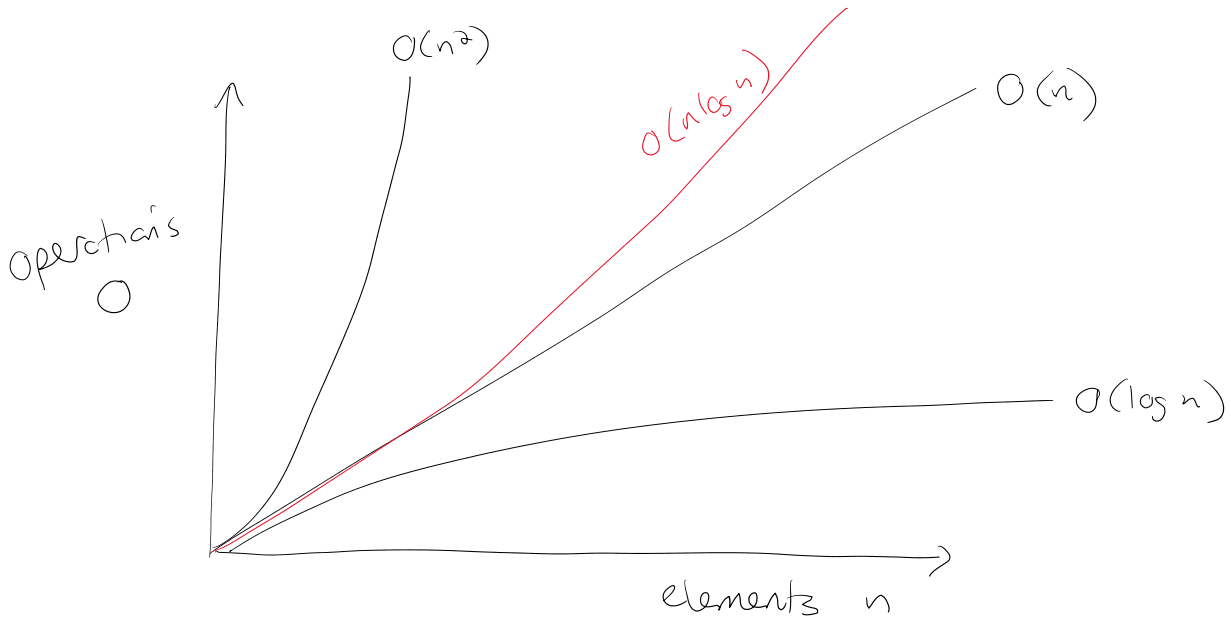


what you need to know for this class



this curve doesn't have a maximum value
- is always increasing without limit
- will always eventually get larger than any particular value

but what about $O(n \log n)$? ("linearithmic")



Section 4.3: cont'd 2024/03/04

example: consider procedures where the number of operations needed for a task of size n is given below. Find Big O for each procedure.

a) $n + \log n + 2^n$ answer: $O(2^n)$

b) $\log n\,(3 + 2n) = 3\log n + 2n\log n$ $O(n \log n)$

c) $n(n-1)(n-2)\cdots 3\cdot 2\cdot 1 = n!$ $O(n!)$

d) $5\log n + 4$ $O(\log n)$