A2 - office hours - today 11-noon → behind BA1180
- today 6-8 pm BA2230 → BA4270
- tomorrow 11-1pm BA2230 (Help Centre)
- ~~tomorrow~~ 6-8 p.m BA2230

Examples:
. regular-expression
→ RegexTreeNode
- matching binary string (1s + 0s) to a RegexTreeNode

CSC148 fall 2013

sorting big-oh

week 9

Danny Heap
heap@cs.toronto.edu
BA4270 (behind elevators)
http://www.cdf.toronto.edu/~heap/148/F13/
416-978-5899

November 6, 2013

Computer Science
UNIVERSITY OF TORONTO

Outline

more big-oh

running time analysis

worst-case running time

size of problem

algorithm's behaviour over large input (size n) is common way to compare performance

constant: $c \in \mathbb{R}^+$ (some positive number)

logarithmic: $c \log n$

not same!

linear: cn (probably not the same c)

quadratic: cn^2)

cubic: cn^3

exponential: $c2^n$

horrible: cn^n or $cn!$

case: $\lg n$

$2^k = n$ means $\lg n = k$

this is the number of times you can divide n in half before reaching 1.

Definition.

- refresher: $a^b = c$ means $\log_a c = b$.
- this runtime behaviour often occurs when we "divide and conquer" a problem (e.g. binary search)
- we usually assume $\lg n$ (log base 2), but the difference is only a constant:

$$2^{\log_2 n} = n = 10^{\log_{10} n} \implies \boxed{\log_2 n} = \boxed{\log_2 10} \times \log_{10} n$$

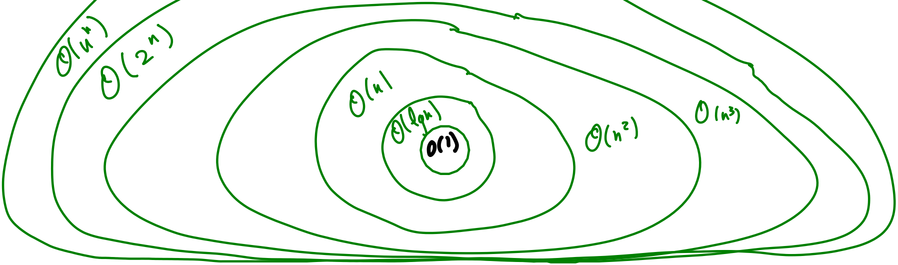
$2^{\log_2 n} = 10^{\log_{10} n}$

- so we just say $\mathcal{O}(\lg n)$.

hierarchy

Since big-oh is an **upper-bound** the various classes fit into a hierarchy:

$$\mathcal{O}(1) \subseteq \mathcal{O}(\lg n) \subseteq \mathcal{O}(n) \subseteq \mathcal{O}(n^2) \subseteq \mathcal{O}(n^3) \subseteq \mathcal{O}(2^n) \subseteq \mathcal{O}(n^n)$$



selection sort (review?)

performance

C → number of "steps" in for loop

X

$n + (n-1) + (n-2) + \dots + 1$

idea: for each position in the list, select the minimum item from that position on

$$
\begin{array}{c}
1 + 2 + \dots + n \\
n + n-1 + \dots + 2 + 1 \\
\hline
(1+n) + (1+n) + \dots + (1+n)
\end{array}
= \frac{n(n+1)}{2} \quad \frac{n^2 + n}{2} = \frac{1}{2}n^2 + \frac{n}{2}
$$

merge sort

idea: divide the list in half, (merge) sort the halves, then merge the sorted results

quick sort

idea: choose a pivot; decide where the pivot goes with respect to the rest of the list, repeat on the partitions...