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
Outline

more big-oh
algorithm’s behaviour over large input (size $n$) is common way to compare performance

- **constant**: $c \in \mathbb{R}^+$ (some positive number)
- **logarithmic**: $O(\log n)$
- **linear**: $cn$ (probably not the same $c$)
- **quadratic**: $cn^2$
- **cubic**: $cn^3$
- **exponential**: $c2^n$
- **horrible**: $cn^n$ or $cn!$
case: $\lg n$

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

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:

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

so we just say $O(\lg n)$. 
Since big-oh is an upper-bound the various classes fit into a hierarchy:

\[ O(1) \subseteq O(\log n) \subseteq O(n) \subseteq O(n^2) \subseteq O(n^3) \subseteq O(2^n) \subseteq O(n^n) \]
selection sort (review?)

performance

\[ C \rightarrow \text{number of "steps" in for loop} \]

\[ \times \]

\[ n + (n-1) + (n-2) + \ldots + 1 \]

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

\[
\sqrt{\frac{1 + 2 + \ldots + n}{n + n-1 + \ldots + 2 + 1}} = \frac{n(n+1)}{2} \]

\[
\frac{n^2 + n}{2}
\]
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...