strike on ... any vote next week (april 1st)
- A3 - discuss pruning, game-state-tree Wed or Mon.
→ lab #10 / quiz cancelled

CSC148 winter 2015

efficiency

week 11

Danny Heap

heap@cs.toronto.edu

BA4270 (behind elevators)

http://www.cdf.toronto.edu/~heap/148/W14/

416-978-5899

March 25, 2015

Outline

searching

height analysis

sorting

big-Oh on paper

big-Oh examples

__contains__

Suppose v refers to a number. How efficient is the following statement in its use of time?

```
v in [97, 36, 48, 73, 156, 947, 56, 236]
```

must look at all elements (say n of them)

Roughly how much longer would the statement take if the list were 2, 4, 8, 16,... times longer? → proportional

Does it matter whether we used a built-in Python list or our implementation of **LinkedList**? also linear

add order...

Suppose we know the list is sorted in ascending order, see sorted_list.py

How does the running time scale up as we make the list 2, 4, 8, 16,... times longer? → 1 step longer for each doubling – trivial.

$\lg(n)$

$\log_2$

Key insight: the number of times I repeatedly divide n in half before I reach 1 is the same as the number of times I double 1 before I reach (or exceed) n: $\log_2(n)$, often known in CS as $\lg n$, since base 2 is our favourite base.

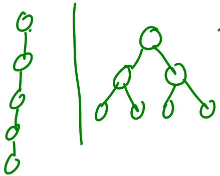
For an n-element list, it takes time proportional to n steps to decide whether the list contains a value, but only time proportional to $\lg(n)$ to do the same thing on an ordered list. What does that mean if n is 1,000,000? What about 1,000,000,000?

Small increment

trees

How efficient is __contains__ on each of the following:

- our general **Tree** class? – linear – looks at every node
- our general **BTNode** class? – also linear
- our **BST** class? – if "well balanced" ~ lg n



The last case should probably be answered "depends..."

node packing...

height 4 — Complete BT

maximum number of nodes in a tree of height:			min # nodes	
0	0		0	
1?	1		1	
2?	3		2	
3?	7		4	
4?	15		8	
5?	31		16	
n? h	$2^h - 1$		2^{h-1}	

invert node packing...

$2^{h-1} \le n \le 2^h - 1$

if $n \leq 2^h - 1$, then $\underbrace{n + 1 \leq 2^h}$. Take lg from both sides:

$n \ge 2^{h-1}$

$\underbrace{\lg(n) \ge h-1}$

$\lg(n) + 1 \ge h$

$$\lg(n+1) \leq h$$

$\lg(n) \sim h$

...where h is the height of the tree

notice that $\lg(n) \equiv \lg(n+1)$, so if our BST is tightly packed (AKA balanced), we use proportional to $\lg(n)$ time to search n nodes

complete BST

logarithmic operations on BST

sorting

how does the time to sort a list with n elements vary with n?
it depends:

- bubble sort
- selection sort
- insertion sort
- some other sort?

quick sort

idea: break a list up (partition) into the part smaller than some value (pivot) and not smaller than that value, sort those parts, then recombine the list:

```
def qs(L):
    ''' (list) -> list
    '''
    if len(L) < 2:
        # copy of L
        return L[:]
    else:
        return (qs([i for i in L if i < L[0]]) +
                [L[0]] +
                qs([i for i in L[1:] if i >= L[0]]))
```

counting quick sort: $n = 7$

$$\text{qs}([4, 2, 6, 1, 3, 5, 7])$$

$$\text{qs}([2, 1, 3]) + [4] + \text{qs}([6, 5, 7])$$

$$\text{qs}([1])+[2]+\text{qs}([3]) \quad + \quad [4] \quad + \quad \text{qs}([5])+[6]+\text{qs}([7])$$

$$[1] \; + \; [2] \; + \; [3] \quad + \quad [4] \quad + \quad [5] \; + \; [6] \; + \; [7]$$

$$[1, 2, 3] \quad + \quad [4] \quad + \quad [5, 6, 7]$$

$$[1, 2, 3, 4, 5, 6, 7]$$