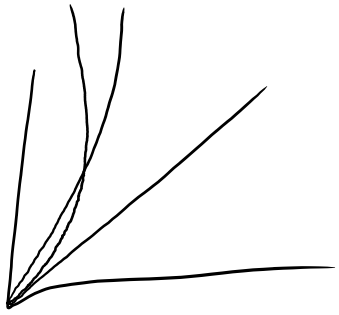
2018-08-30 More Algorithm Analysis

Thursday, August 30, 2018 3:00 PM

Recap

- Overall idea: look at how the number of commands varies
- Loop counts often vary. Thus a good place to start.
- Often good initial guesses:
  - A single loop is often O(N) -> linear
  - Nested loops are often O(N^2) -> quadratic
    - More general O(N^X) where X = # of nested loops
  - Two unrelated loops O(M + N), which reduces O(N)
- We only really care about the most significant factor
  - What approaches infinity faster?
  - E.g. O(N^32 + 2^n + 3N) reduces to O(2^n)



Quick Warmup

```
//O(N) linear relationship
while(two != NULL && two->next != NULL)
{
    one = one->next;
    two = two->next->next;
    if(one == two)
        return true;
}
```

1 cmd

List Size	# of commands	Delta increase
1	1	1
2	2	1
3	3	1
4	4	1
5	5	1
6	6	1
7	7	1
8	8	1

Δ increase = 1

$\sum 1 = X$

O(n)

New Clue #1

- If they exist, we need count function calls (especially recursive function call)

```
Int x(int y){
  if(y <= 0) return 0;
  return y + x(y - 1);
}
```

base case

inductive/recursive case

- When we count recursive functions, we want to know how quickly we reach the base case.

Y	Commands	Delta
1	2	1
2	3	1
3	4	1
4	5	1
5	6	1

→ O(N)

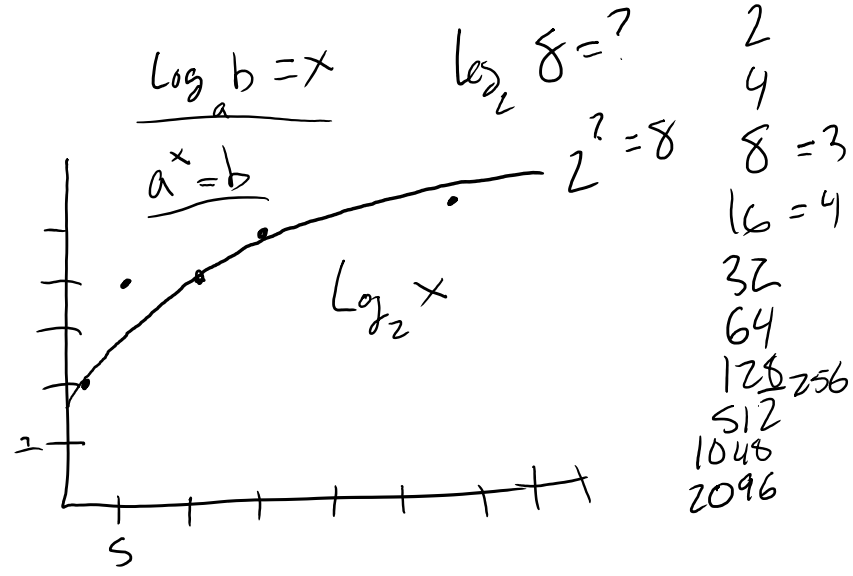
5	6	1

- Binary search algorithm allows us to quickly discover whether or not a given number exists in a sorted sequence

0	1	2	3	4	5	6	7	8	9
1	3	7	9	12	15	20	100	111	112

- Logic: pick a number. If that number is larger than what we're looking for, what we're looking for must exist to the left. If that number is smaller, then what we're looking for must exist to the right.
- Picking the middle (in terms of location) number is always the most efficient choice
- See binary_search.cpp in notes for today's lecture for code

Numbers.size()	# commands	Delta
2	2	--
4	3	1
6	4	1
8	4	0
10	4	0
12	5	1
14	5	0
16	5	0
18	5	0
20	5	0
22		
24	6	1
26		
28	6	0
32	6	0
36	6	0
40	6	
48	7	



New Rule

- Whenever we cut a problem in half every iteration of function call, we observe logarithmic behavior. Alternatively,
- Whenever we notice an exponential "jump" towards the correct solution, we observe logarithmic behavior.