Algorithms and Big-O

CMSC 131

Algorithms

What is an algorithm?

- Class discussion brought up ideas like recipes, directions, steps to take to solve a problem and the fact that we've been working with algorithms all semester.

What types of algorithms exist?

- Class discussion brought up ideas like efficient vs. inefficient or fast vs. slow, whether there is randomness involved, algorithms designed by other algorithms, and that some are based on mathematical definitions.

Algorithm Question

- Input: 4 digit number n
- Do the following until you encounter a steady state or a cycle:

  n1=digits sorted in increasing order

  n2=digits sorted in decreasing order

  n=n2-n1
- We discussed the idea of trying to classify the runtime of an algorithm.
- We saw that in the end there were only two possible answers and how an efficient program could take advantage of that.

Growth Rate of Functions

f(n) = 1

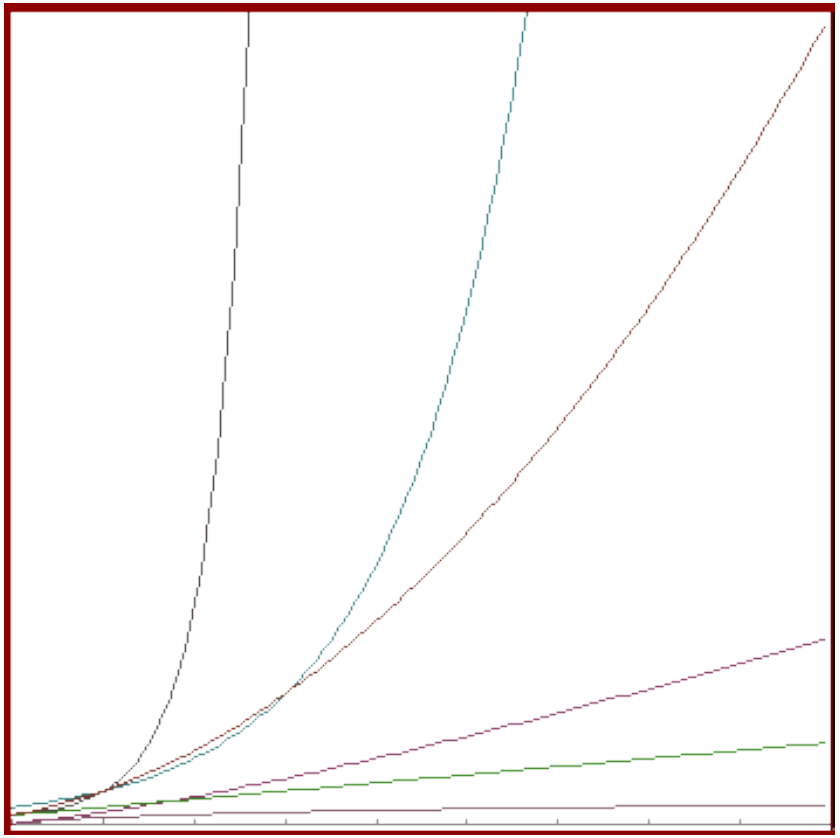
f(n) = log$n$

f(n) = $n$log$n$

f(n) = n^2

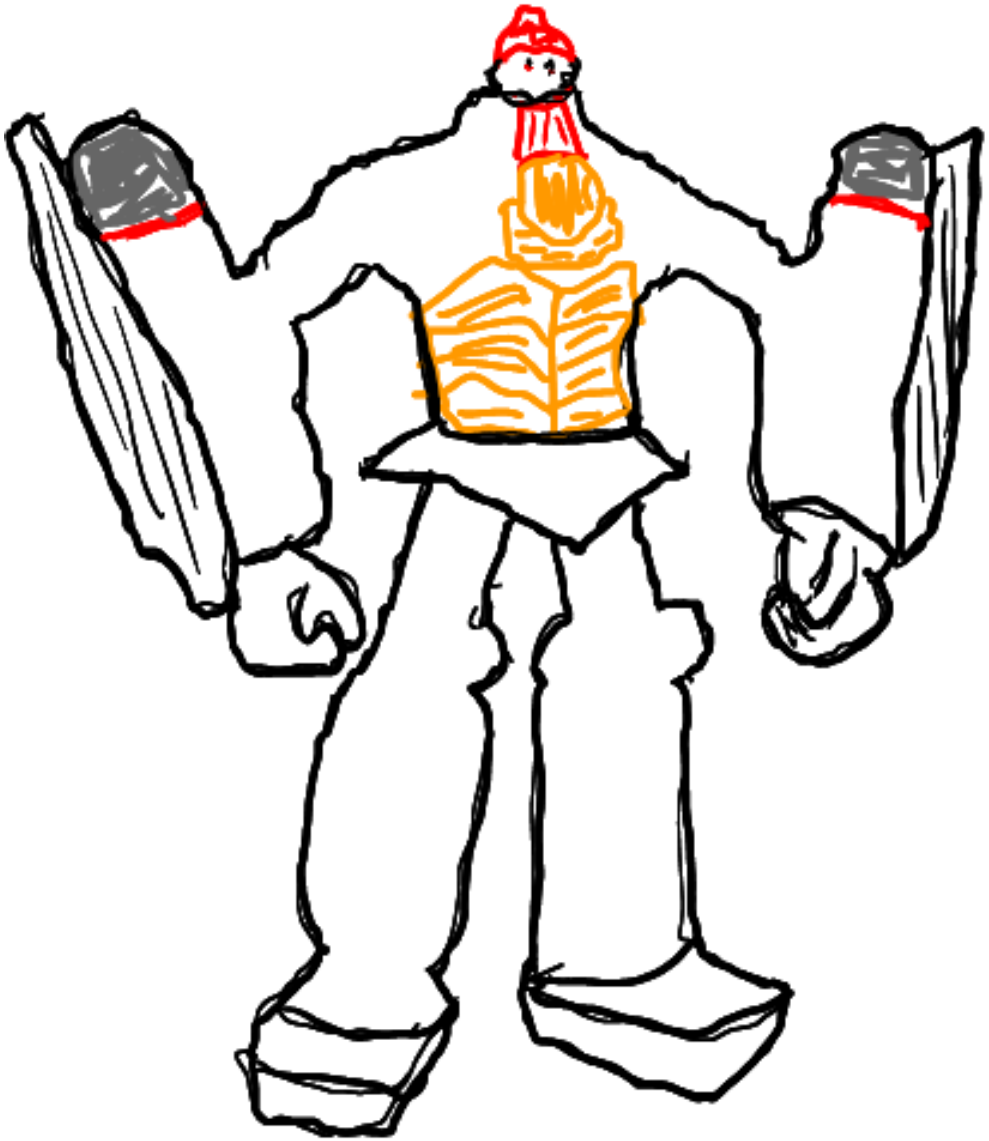
f(n) = 2^n

f(n) = n^n



- Which grows faster?
- How do you *prove* that?
- In CMSC351 this will be addressed and it will also be shown that “for sufficiently large input sizes” any low-order terms or even constant multiplicative factors are insignificant in terms of relative growth rate.

Big-O



Big-O

Something that is used to capture this notion of growth rate is "Big-O" classification.

The basic concept behind Big-O is to be able to show that the growth rate of the runtime of a particular algorithm is ***no worse than*** some basic function on the input size.

The input size might have to be "sufficiently large" to expose this relationship (ie: for small inputs maybe algorithm A is better, but is you expect to scale to large problems algorithm B is better).

Since we are talking about growth rate, both $49n^2$ and n^2-500n are "Big-O of n^2" (there is a precise mathematical definition presented in 351).

Which of the following are Big-O of n^2?

1. ½ n^2
2. 50 n^2
3. 100000 n
4. n*log*n
5. 1/1000000th n^3

When does input size matter?

In some problems it is known that there is a threshold where you should switch from one algorithm to another.

In the current Java sorting method for things like sorting an array, the method uses the size of the array being processed at various stages of to decide between applying one of two different sorting algorithms on that array.

Tower of Lego

- Assume we have a large supply of typical style Lego blocks which are 1" cubes.
- We want to build a tower 20" tall (don't worry about whether it will top over).
- We have to use special robot cranes to do it, and these cranes can move one stack and place it on top of another stack in 30 seconds if neither stack is more than 10" high and 60 seconds if either is more than 10" high.
- How do you build the 20" tall tower in the least amount of time?

Tower of Lego

- Assume we have a large supply of typical style Lego blocks which are 1" cubes.
- We want to build a tower 40" tall (don't worry about whether it will top over).
- We have to use special robot cranes to do it, and these cranes can move one stack and place it on top of another stack in 30 seconds if neither stack is more than 10" high and 60 seconds if either is more than 10" high.
- How do you build the 40" tall tower in the least amount of time?

Tower of Lego

- Assume we have a large supply of typical style Lego blocks which are 1" cubes.
- We want to build a tower 80" tall (don't worry about whether it will top over).
- We have to use special robot cranes to do it, and these cranes can move one stack and place it on top of another stack in 30 seconds if neither stack is more than 10" high and 60 seconds if either is more than 10" high.
- How do you build the 80" tall tower in the least amount of time?