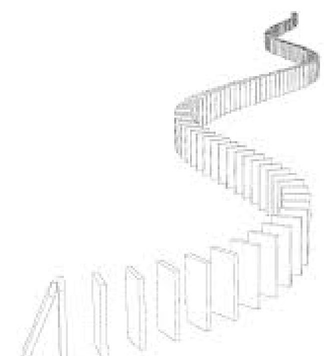
COMP 1002

Logic for Computer Scientists

Lecture 31



Admin stuff

- Assignment 5 is posted.
  - Postponed till Monday April 3rd, 7pm.
- Next week:
  - “Mini-lab” on Monday, April 3rd (+ finishing up).
    - Instead of the lecture: in EN-2007, at 1pm
  - Review for the final exam on Tuesday, April 4th.
  - Practice exam on Wednesday, April 5th
    - 9am-11am in CS-1019.

- Please do the CEQs!
  - Especially the comments!
  - As this is the first time we run this course, I would love to know what worked and what did not, and what should be done differently next time.

Tower of Hanoi game


- Rules of the game:
  - Start with all disks on the first peg.
  - At any step, can move a disk to another peg, as long as it is not placed on top of a smaller disk.
  - Goal: move the whole tower onto the second peg.
- Question: how many steps are needed to move the tower of 8 disks? How about n disks?

Tower of Hanoi game

- Rules of the game:
  - Start with all disks on the first peg.
  - At any step, can move a disk to another peg, as long as it is not placed on top of a smaller disk.
  - Goal: move the whole tower onto the second peg.
- Question: how many steps are needed to move the tower of 8 disks? How about n disks?

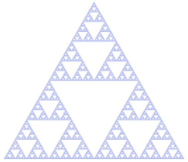
- Let us call the number of moves needed to transfer n disks H(n).
  - Names of pegs do not matter: from any peg i to any peg $j \neq i$ would take the same number of steps.
- Basis: only one disk can be transferred in one step.
  - So H(1) = 1
- Recursive step:
  - suppose we have n-1 disks. To transfer them all to peg 2, need $H(n-1)$ number of steps.
  - To transfer the remaining disk to peg 3, 1 step.
  - To transfer n-1 disks from peg 2 to peg 3 need H(n-1) steps again.
  - So H(n) = 2H(n-1)+1 (recurrence).
- Closed form: H(n) = $2^n - 1$.

Recurrence relations

- **Recurrence**: an equation that defines an n^{th} element in a sequence in terms of one or more of previous terms.
  - H(n) = 2H(n-1)+1
  - F(n) = F(n-1)+F(n-2)
  - T(n) = aT(n-1)
- A **closed form** of a recurrence relation is an expression that defines an n^{th} element in a sequence in terms of n directly.
  - Often use recurrence relations and their closed forms to describe performance of (especially recursive) algorithms.

Closed forms of some sequences

- Arithmetic progression:
  - Sequence: $c, c+d, c+2d, c+3d, \dots, c+nd, \dots$
  - Recursive definition:
    - Basis: $s_0 = \mathrm{c}$, for some c$\in \mathbb{R}$
    - Recurrence: $s_{n+1} = \mathrm{s_n} + \mathrm{d}$, where $d \in \mathbb{R}$ is a fixed number.
  - Closed form: $s_n = c + nd$
    - Closed forms are very useful for analysis of recursive programs, etc.
- Geometric progression:
  - Sequence: $c, cr, cr^2, cr^3, \dots, cr^n, \dots$
  - Recursive definition:
    - Basis: $s_0 = \mathrm{c}$, for some c$\in \mathbb{R}$
    - Recurrence: $s_{n+1} = \mathrm{s_n} \cdot r$, where r$\in \mathbb{R}$ is a fixed number.
  - Closed form: $s_n = c \cdot r^n$



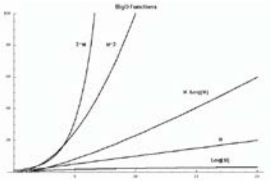
Closed form for Tower of Hanoi

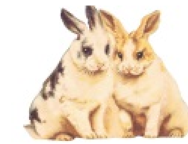
- Solving the recurrence H(n)=2H(n-1)+1
  - H(n) = $2 \cdot H(n-1)+1$

    $= 2(2H(n-2)+1)+1$ = $2^2 H(n-2)+2+1$

    $= 2^3 H(n-3)+2^2+2+1$

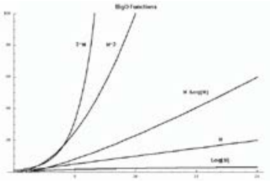
    $= 2^4\, H(n-4)+2^3+2^2+2+1 \ldots$
  - In general, $H(n) = \Sigma_{i=0}^{n-1} 2^i = 2^n - 1$
    - Proof by induction.
    - Or by noticing that a binary number 111...1 plus 1 gives a binary number 10000…0
  - So the function defined by H(n) grows exponentially
    - As a function of n.
- Solving recurrences in general might be tricky.
  - However, when the recurrence is of the form T(n)=a T(n/b)+f(n), there is a general method to estimate the growth rate of a function defined by the recurrence
  - Called the Master Theorem for recurrences.

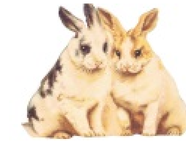




Function growth.

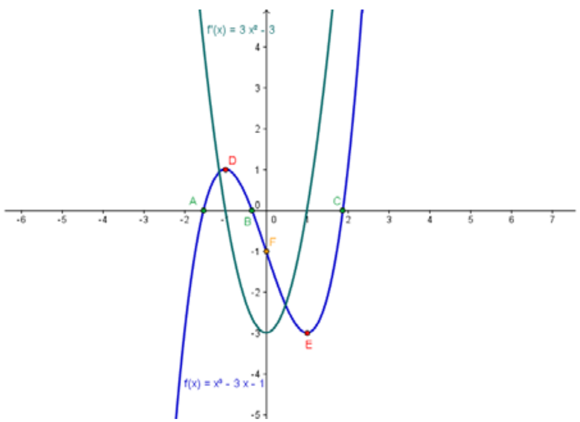
- What does it mean to “grow” at a certain speed? How to compare growth rate of two functions?
  - Is f(n)=100n larger than $g(n) = n^2$?
    - For small n, yes. For n > 100, not so much…
  - As usually program take longer on larger inputs, performance on larger inputs matters more.
  - Constant factors don’t matter that much.
- So to compare two functions, check which becomes larger as n increases (to infinity).
  - Ignoring constant factors, as they don’t contribute to the rate of growth.

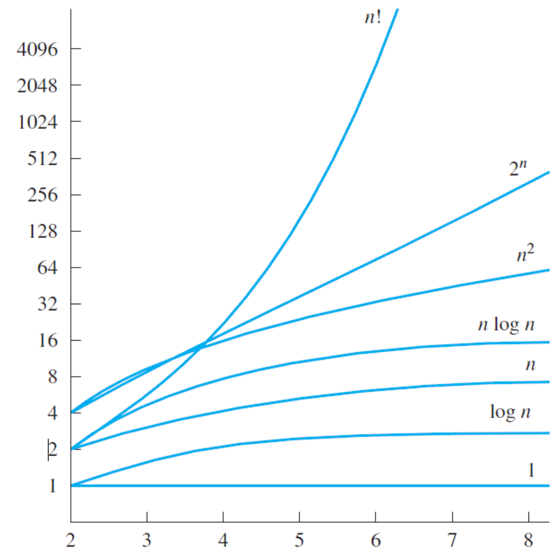




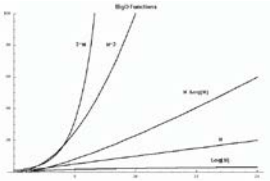
Function growth.

- How to estimate the rate of growth?
  - Plotting a graph?





- Not quite conclusive:
  - How do you know what they will do past the graphed part?



O-notation.

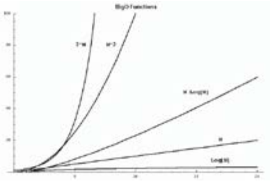
- We say that f(n) grows at least as fast as g(n) if
  - There is a value n_0 such that after n_0, $g(n)$ is always at most as large as $f(n)$
    - More precisely, compare absolute values: |g(n)| vs. |f(n)|
  - Moreover, ignore constant factors:
    - So if two functions only differ by a constant factor, consider them having the same growth rate.
  - Denote set of all functions growing at most as fast as $g(n)$ by $\boldsymbol{O(g(n))}$
    - **Big-Oh** of g(n).
    - g(n) is an **asymptotic upper bound** for f(n).
    - When both $f(n) \in O(g(n))$ and $g(n) \in O(f(n))$, write $f(n) \in \Theta(g(n))$
      - f(n) is in **big-Theta** of g(n)).

- More generally, for real-valued functions f(x) and g(x),

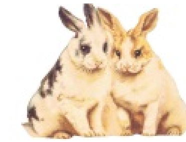
$$f(x) \in O(g(x)) \text{ iff}$$

$$\exists\, x_0 \in \mathbb{R}^{\geq 0}\, \exists c \in \mathbb{R}^{>0}\;\; \forall x \geq x_0\;\; |f(x)| \leq c \cdot |g(x)|$$

- That is, from some point x_0 on, $|f(x)|$ is bounded from above by |g(x)| (up to a constant factor).
- Usually in CS have functions $\mathbb{N} \to \mathbb{R}^{\geq 0}$, so use n for x and ignore | |.

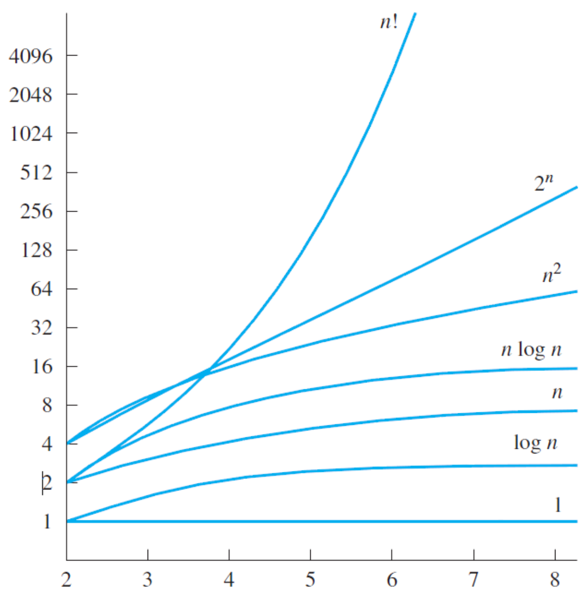


O-notation.



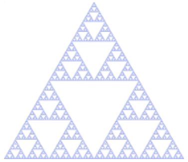
$f(n) \in O(g(n))$ iff

$$\exists\, n_0 \in \mathbb{N}\ \exists c \in \mathbb{R}^{>0}\ \ \forall n \geq n_0\ \ f(n) \leq c \cdot g(n)$$



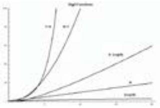
You will see a lot of O-notation in COMP 2002.

- $f(n) = n^2, g(n) = 2^n$.
  - Take c=1, $n_0 = 4$.
  - For every $n \geq n_0, f(n) \leq g(n)$
    - Proof by induction.
  - So $\mathrm{n}^2 \in O(2^n)$
- $f(n) = n^2, g(n) = 10n$.
  - Take arbitrary c and look at $n^2 \leq c \cdot 10n$.
  - No matter what c is, when $n > c \cdot 10$, $n^2 \geq c \cdot 10n$
  - So $n^2 \notin O(10n)$.
- $f(n) = n^2 + 100n, g(n) = 10n^2$.
  - Here, $f(n) \in O(g(n))$ and also $g(n) \in O(f(n))$
    - So $f(n) \in \Theta(g(n))$
    - $f(n) \in O(g(n))$: c = 20 and/or $n_0 = 100$ work.
    - $g(n) \in O(f(n))$: Take c=10, $n_0 = 1$.
  - Can ignore not only constants, but also all except the leading term in the expression.

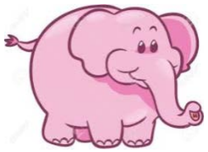
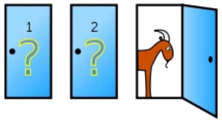


Master theorem for solving recurrences

- Let $a, b, c, d \in \mathbb{R}$ such that $a \geq 1,\ b \geq 2, c > 0$, $d\ \geq 0$, and let f(n) $\in \Theta(n^c)$
- Let T(n) be the following recurrence relation:
  - Base: T(1) = d
  - Recurrence: $T(n) = a\,T\left(\left\lceil \frac{n}{b} \right\rceil\right) + f(n)$
- Then the growth rate of T(n) is:
  - If $\log_b a < c$ then T$(n) \in \Theta\big(f(n)\big)$
  - If $\log_b a = c$ then T$(n) \in \Theta(f(n) \log n\ )$
  - If $\log_b a > c$ then T$(n) \in \Theta\big(n^{\log_b a}\ \big)$




More to come…

- You will see a lot of algorithm analysis and use of the concepts we developed in COMP 2002 and beyond.
  - **Logic, sets, relations and graphs** for specification, modeling problems and describing what you are doing.
  - **Logic, induction and models of computation** for proving program correctness and analysis of problem complexity.
  - **Recursive definitions of algorithms, counting and probability** for algorithm performance and problem solving.
    - With the million dollar problem rearing its head every now and then











Have fun!