Sums, products and sequences

- How to write long sums, e.g., 1+2+… (n-1)+n concisely?
  - Sum notation (“sum from 1 to n”): $\sum_{i=1}^{n} i = 1 + 2 + \ \ldots + n$
    - If n=3, $\sum_{i=1}^{3} i$ = 1+2+3=6.
    - The name “i“ does not matter. Could use another letter not yet in use.
- In general, let $f: \mathbb{Z} \rightarrow \mathbb{R},\ m, n \in \mathbb{Z}, m \leq n$.
  - $\sum_{i=m}^{n} f(i) = \ f(m) + f(m+1) + \cdots + f(n)$
    - If m=n, $\sum_{i=m}^{n} f(i)$ =f(m)=f(n).
    - If n=m+1, $\sum_{i=m}^{n} f(i)$ = f(m)+f(m+1)
    - If n>m, $\boldsymbol{\sum_{i=m}^{n} f(i)}$ **=** $\boldsymbol{(\sum_{i=m}^{n-1} f(i)) + f(n)}$
    - Example: $f(x) = x^2$. $2^2 + 3^2 + 4^2 = \sum_{i=2}^{4} i^2 = 29$
- Similarly for product notation (product from m to n):
  - $\Pi_{i=m}^{n} f(i) = f(m) \cdot f(m+1) \cdot \ldots \cdot f(n)$ = $\boldsymbol{(\Pi_{i=m}^{n-1} f(i)) \cdot f(n)}$
  - For $f(x) = x,\ 2 \cdot 3 \cdot 4 \ = \Pi_{i=2}^{4} i = 24$
  - $1 \cdot 2 \cdot \ldots \cdot n = \Pi_{i=1}^{n} i = n!$ (n factorial)

Recurrences and sequences

- To define a sequence (of things), describe the process which generates that sequence.
  - Sequence: enumeration of objects $s_1, s_2, s_3, \ldots, s_n, \ldots,$
    - Sometimes use notation $\{s_n\}$ for the sequence (i.e., set of elements forming a sequence)
    - Sometimes start with s_0 rather than s_1

  - **Basis (initial conditions):** what are the first (few) element(s) in the sequence.
    - $\sum_{i=0}^{0} i = 0. \sum_{i=m}^{m} i = \text{m}.$
    - 0! = 1. 1!=1.
    - $\mathrm{A}_0 = \emptyset$

  - **Recurrence (recursion step, inductive definition)**: a rule to make a next element from already constructed ones.
    - $\sum_{i=m}^{n+1} i = (\sum_{i=m}^{n} i) + (n+1).$ Here, assume that $m \leq n$
    - (n+1)! = n! · (n+1)
    - $A_{n+1} = \mathcal{P}(A_n)$

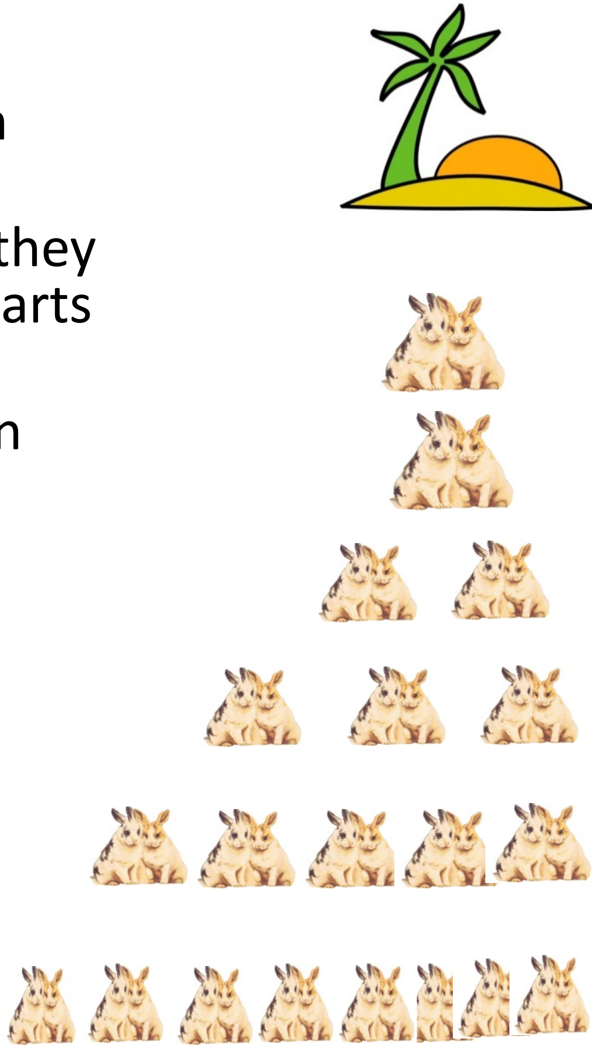
- Resulting sequences:
  - m, 2m+1, 3m+3, …
  - 1, 2,6, 24, 120, …
  - $\emptyset, \{\emptyset\}, \{\emptyset, \{\emptyset\}\}, \left\{\emptyset, \{\emptyset\}, \{\{\emptyset\}\}, \{\emptyset, \{\emptyset\}\}\right\}, \ldots$

Special sequences

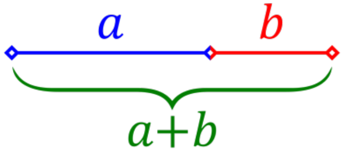
- **Arithmetic progression:**
  - Sequence: $c, c+d, c+2d, c+3d, \dots, c+nd, \dots$
  - Recursive definition:
    - Basis: $s_0 = \mathrm{c}$, for some c$\in \mathbb{R}$
    - Recurrence: $s_{n+1} = \mathrm{s_n} + \mathrm{d}$, where $d \in \mathbb{R}$ is a fixed number.
  - Closed form: $s_n = c + nd$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

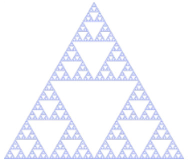
- **Geometric progression:**
  - Sequence: $c, cr, cr^2, cr^3, \dots, cr^n, \dots$
  - Recursive definition:
    - Basis: $s_0 = \mathrm{c}$, for some c$\in \mathbb{R}$
    - Recurrence: $s_{n+1} = \mathrm{s_n} \cdot r$, where r$\in \mathbb{R}$ is a fixed number.
  - Closed form: $s_n = c \cdot r^n$

Fibonacci sequence



- Imagine that a ship leaves a pair of rabbits on an island (with a lot of food).
- After a pair of rabbits reaches 2 months of age, they produce another pair of rabbits, which in turn starts reproducing when reaching 2 months of age…
- How many pairs rabbits will be on the island in n months, assuming no rabbits die?
- Basis: $F_1 = 1,\ \mathrm{F}_2 = 1$
- Recurrence: $F_n = F_{n-1} + F_{n-2}$
- Sequence: 1,1,2,3,5,8,13…
- Closed form: $F_n = \frac{((1+\sqrt{5})/2)^n - ((1-\sqrt{5})/2)^n}{\sqrt{5}}$
  - Golden ratio:
    - $\frac{a+b}{a} = \frac{a}{b} = \frac{1+\sqrt{5}}{2}$





Partial sums

- Properties of a sum:
  - $\sum_{i=m}^{n} f(i) + g(i) = \sum_{i=m}^{n} f(i) + \sum_{i=m}^{n} g(i)$
  - $\sum_{i=m}^{n} c \cdot f(i) = c \sum_{i=m}^{n} f(i)$

- Sum of arithmetic progression:
  - $s_n = c + nd$ for some c, $d \in \mathbb{R}$
  - Sequence: $c, c+d, c+2d, c+3d, \ldots, c+nd, \ldots$
  - Partial sum:
    - $\sum_{i=0}^{n} s_n = \sum_{i=0}^{n} (c + id) = \sum_{i=0}^{n} c + \sum_{i=0}^{n} id =$ $\mathrm{c(n+1) + d} \sum_{i=0}^{n} i = c(n+1) + d\,\frac{n(n+1)}{2}$

- Sum of geometric progression:
  - $s_n = c \cdot r^n$ for some c, $r \in \mathbb{R}$
  - Sequence: $c, cr, cr^2, cr^3, \ldots, cr^n, \ldots$
  - Partial sum:
    - If r=1, then $\sum_{i=0}^{n} s_n = c(n+1)$
    - If $r \neq 1$, then $\sum_{i=0}^{n} s_n = \frac{cr^{n+1} - c}{r-1}$

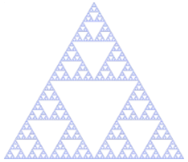
Tower of Hanoi game

- 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: $H(n) = 2^n - 1$.



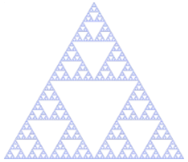
Recurrence relations

- **Recurrence**: an equation that defines an n^{th} element in a sequence in terms of one or more of previous terms.
    - Think of $F(n) = s_n$ for some sequence $\{s_n\}$
  - H(n) = 2H(n-1)+1
  - F(n) = F(n-1)+F(n-2)

- 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.

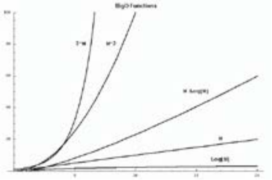
Closed forms of some sequences

- Arithmetic progression:
  - Sequence: $c, c+d, c+2d, c+3d, \ldots, c+nd, \ldots$
  - 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: $c, cr, cr^2, cr^3, \ldots, cr^n, \ldots$
  - 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:** $s_n = c \cdot r^n$



Solving recurrences

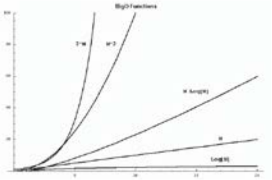
- Solving a recurrence: finding a closed form.
  - Solving the recurrence H(n)=2H(n-1)+1
    - $$\begin{aligned} H(n) &= 2 \cdot H(n-1) + 1 \\ &= 2(2H(n-2)+1)+1 = 2^2 H(n-2) + 2 + 1 \\ &= 2^3 H(n-3) + 2^2 + 2 + 1 \\ &= 2^4\, H(n-4) + 2^3 + 2^2 + 2 + 1 \ldots \end{aligned}$$
  - Closed form: $H(n) = \Sigma_{i=0}^{n-1} 2^i = 2^n - 1$
    - Proof by induction (coming in the next lecture).
    - Or by noticing that a binary number 111...1 plus 1 gives a binary number 10000…0
  - So adding one more disk doubles the number of steps.
    - We say that the function defined by H(n) **grows exponentially**
- Solving recurrences in general might be tricky.
  - 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.

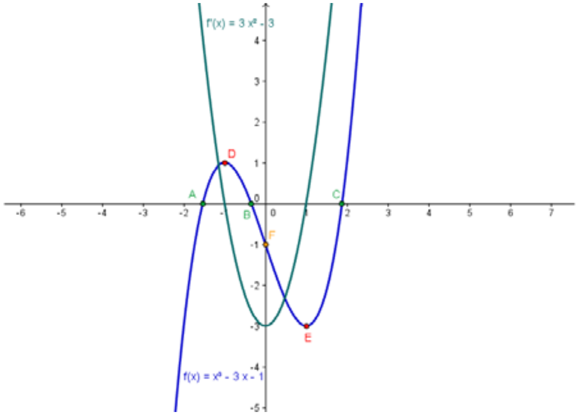
- What does it meant to “grow” at a certain speed? How to compare growth rate of two functions?
    - Is f(n)=100n larger than $g(n) = n^2$?
        - For small n, yes. For n > 100, not so much…
    - As usually program take longer on larger inputs, performance on larger inputs matters more.
    - Constant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

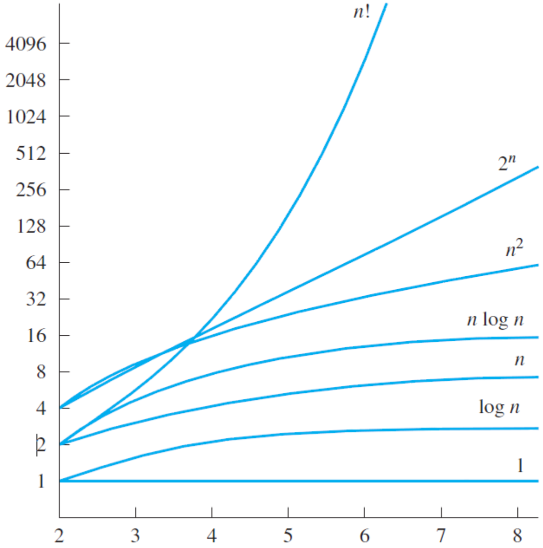
- So to compare two functions, check which becomes larger as n increases (to infinity).
    - Ignoring constant factors, as they don’t contribute to the rate of growth.



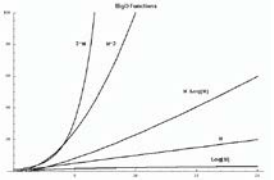
Function growth.

- 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.

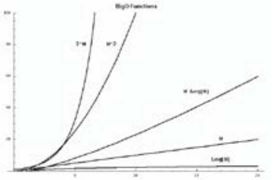
- Denote set of all functions growing at most as fast as $g(n)$ by $\boldsymbol{O(g(n))}$
  - **Big-Oh** of g(n).
  - g(n) is an **asymptotic upper bound** for f(n).
  - When both $f(n) \in O(g(n))$ and $g(n) \in O(f(n))$, write $f(n) \in \Theta(g(n))$
    - f(n) is in **big-Theta** of g(n)).

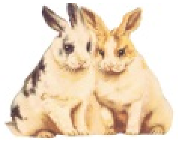
- More generally, for real-valued functions f(x) and g(x),

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

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

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





O-notation.

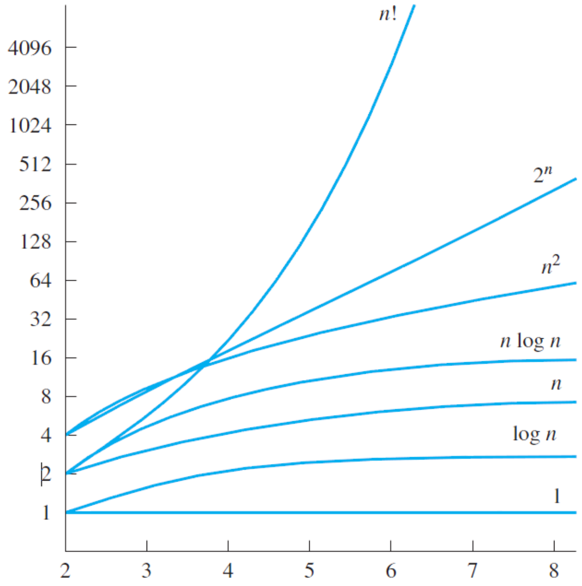
$f(n) \in O(g(n))$ iff

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

- $f(n) = n^2, g(n) = 2^n$.
  - 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.



You will see some O-notation in COMP 1000 and a lot in COMP 2002.