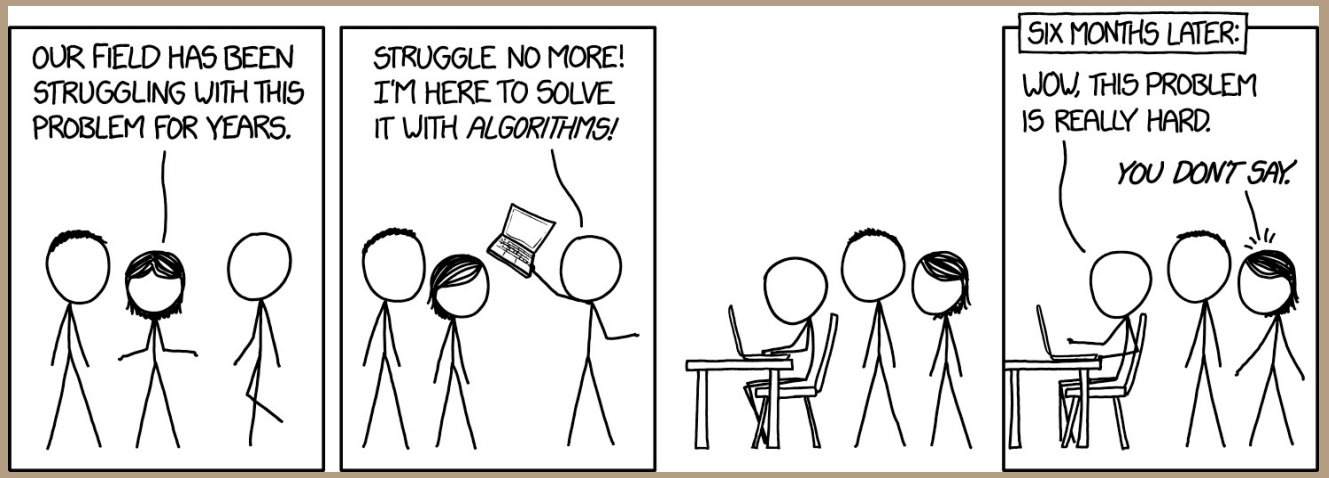
Complexity & Algorithms — Spring '26



Background

Recap

- Any syllabus questions?
- HW0 is posted
- Reading posted for next week

Expectations

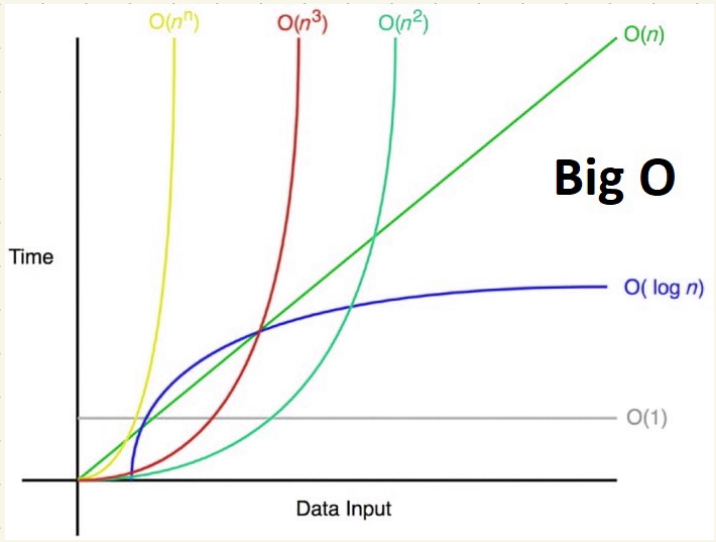
When I say "give an algorithm", "show how to compute...", etc, what do I mean?

Background

① Big-O & little-o & Θ:

Formally, $f(n) = O(g(n))$

if:



Why?

Example: $5n^3 - 6$ and $14n^3 + 3n^2 + 11$

(2) Logarithms: useful identities!

Find it in your discrete math reference, ie ⟶

$$\log_b(xy) = \log_b(x) + \log_b(y)$$

$$\log_b\left(\frac{x}{y}\right) = \log_b(x) - \log_b(y)$$

$$\log_b(x^y) = y \log_b(x)$$

$$\log_b(\sqrt[y]{x}) = \frac{\log_b(x)}{y}$$

Recall Dfn: $\log_a b =$

Identities: how do exponents behave?

Another:

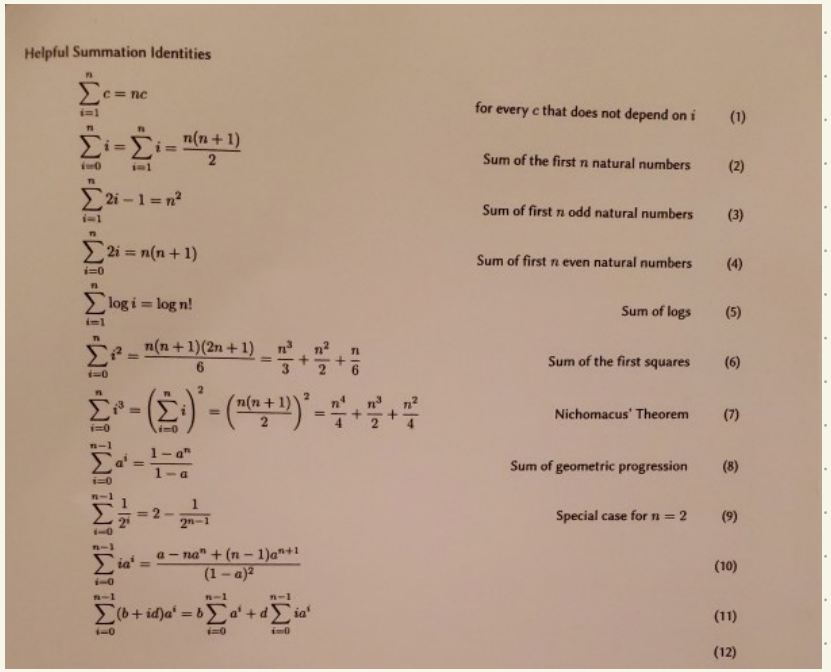
$$\log_a b = \frac{\log_x b}{\log_x a} \quad \text{with any base } x$$

Use it: Show $8 \log_{10} n$ is $O(\log_2 n)$:

③ Summations:

again, your discrete math book has a table. Find it.

Ex:



Helpful Summation Identities

$$\sum_{i=1}^{n} c = nc \qquad \text{for every } c \text{ that does not depend on } i \qquad (1)$$

$$\sum_{i=0}^{n} i = \sum_{i=1}^{n} i = \frac{n(n+1)}{2} \qquad \text{Sum of the first } n \text{ natural numbers} \qquad (2)$$

$$\sum_{i=1}^{n} 2i - 1 = n^2 \qquad \text{Sum of first } n \text{ odd natural numbers} \qquad (3)$$

$$\sum_{i=0}^{n} 2i = n(n+1) \qquad \text{Sum of first } n \text{ even natural numbers} \qquad (4)$$

$$\sum_{i=1}^{n} \log i = \log n! \qquad \text{Sum of logs} \qquad (5)$$

$$\sum_{i=0}^{n} i^2 = \frac{n(n+1)(2n+1)}{6} = \frac{n^3}{3} + \frac{n^2}{2} + \frac{n}{6} \qquad \text{Sum of the first squares} \qquad (6)$$

$$\sum_{i=0}^{n} i^3 = \left(\sum_{i=0}^{n} i\right)^2 = \left(\frac{n(n+1)}{2}\right)^2 = \frac{n^4}{4} + \frac{n^3}{2} + \frac{n^2}{4} \qquad \text{Nichomacus' Theorem} \qquad (7)$$

$$\sum_{i=0}^{n-1} a^i = \frac{1-a^n}{1-a} \qquad \text{Sum of geometric progression} \qquad (8)$$

$$\sum_{i=0}^{n-1} \frac{1}{2^i} = 2 - \frac{1}{2^{n-1}} \qquad \text{Special case for } n = 2 \qquad (9)$$

$$\sum_{i=0}^{n-1} ia^i = \frac{a - na^n + (n-1)a^{n+1}}{(1-a)^2} \qquad (10)$$

$$\sum_{i=0}^{n-1} (b + id)a^i = b\sum_{i=0}^{n-1} a^i + d\sum_{i=0}^{n-1} ia^i \qquad (11)$$

(12)

Notation reminder: $\sum_{i=1}^{n} f(i)$

Note: $f(i)$ is any function!

$$\sum_{i=1}^{n} 5 =$$

$$\sum_{i=1}^{n} i =$$

(4) Induction: There is a template!

Base case:

Ind hypothesis:

Ind. step:

Think of this as "automating" a proof:

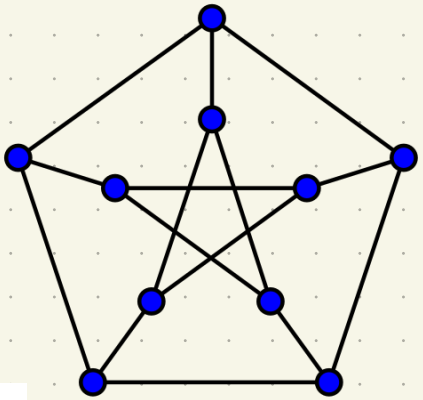
Example $\sum_{i=1}^{n} F_i = F_{n+2} - 1$

Recall F_n :

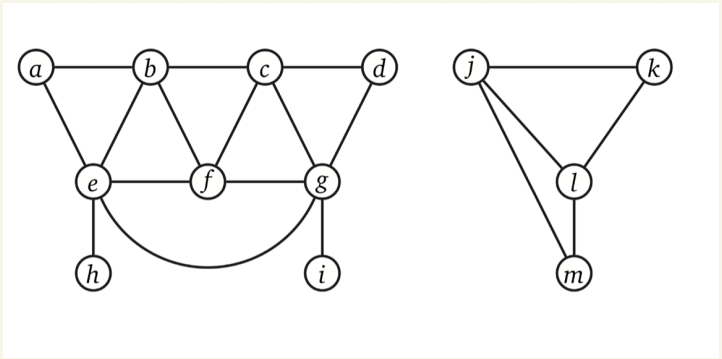
Induction on structures:

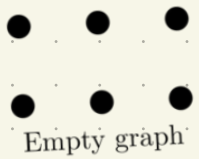
Consider graphs: $G = (V, E)$

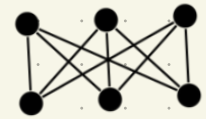
Theorem: In any undirected graph, the number of vertices of odd degree is even.

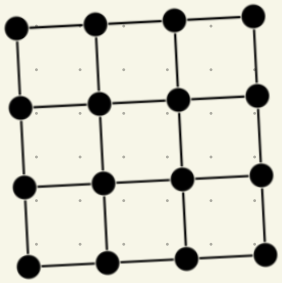


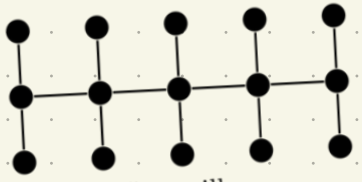
Examples:




Empty graph


Complete bipartite


Grid


Caterpillar

Note here:

Proof by induction on the number of edges:

Base case:

Inductive hypothesis:

Inductive Step:

Another: Every rooted binary tree of height h has $\leq 2^{h+1} - 1$ nodes

Recall: height$(T) = \begin{cases} \\ \\ \\ \end{cases}$

Proof:

5 Pseudo code & runtime:

Discrete math examples (from Rosen textbook)

ALGORITHM 1 Finding the Maximum Element in a Finite Sequence.

procedure $max(a_1, a_2, \ldots, a_n$: integers)
$max := a_1$
for $i := 2$ **to** n
    if $max < a_i$ **then** $max := a_i$
return max {max is the largest element}

This book:

FIBONACCIMULTIPLY$(X[0..m-1], Y[0..n-1])$:
  $hold \leftarrow 0$
  for $k \leftarrow 0$ to $n+m-1$
    for all i and j such that $i+j=k$
      $hold \leftarrow hold + X[i] \cdot Y[j]$
    $Z[k] \leftarrow hold \bmod 10$
    $hold \leftarrow \lfloor hold/10 \rfloor$
  return $Z[0..m+n-1]$

Pseudocode conventions here:

Variable assignment:

Boolean comparison:

Arrays: A[0..n-1]
- each element:

Loops:

Pseudocode format:

In a pinch, pretend you're in Python or Ruby → high level & readable.

I realize this is not a "definition" - that is the point!

It's about effective communication.

Next reading: recursion

Most of you indicated you'd seen it before. Topics here:

- Towers of Hanoi
- Merge sort
- Recap of recurrences & "Master theorem"
- Linear time selection
- Multiplication (again)
- Exponentiation

A high level note on recursion:

Recursion really can be simpler & useful!

often depends upon the language and setup.

Counter-intuitive, but that's often due to lack of practice.

Often considered slower?

Recursion

- If you can solve directly (usually because input is small), do it!
- Otherwise, reduce to simple (usually smaller) instances of the same problem.

Recursion Fairy

- Helps to solidify that "black box" mentality, so you don't keep unpacking the next level.

(She's also called the "induction hypothesis".)

Classic example

Our book

```
QUICKSORT(A[1..n]):
  if (n > 1)
      Choose a pivot element A[p]
      r ← PARTITION(A, p)
      QUICKSORT(A[1..r−1])      ⟨⟨Recurse!⟩⟩
      QUICKSORT(A[r+1..n])      ⟨⟨Recurse!⟩⟩
```

```
PARTITION(A[1..n], p):
  swap A[p] ↔ A[n]
  ℓ ← 0                         ⟨⟨#items < pivot⟩⟩
  for i ← 1 to n−1
      if A[i] < A[n]
          ℓ ← ℓ + 1
          swap A[ℓ] ↔ A[i]
  swap A[n] ↔ A[ℓ + 1]
  return ℓ + 1
```

Algorithm 1 Quicksort

```
1: procedure QUICKSORT(A, p, r)
2:     if p < r then
3:         q = PARTITION(A, p, r)
4:         QUICKSORT(A, p, q − 1)
5:         QUICKSORT(A, q + 1, r)
6:     end if
7: end procedure
8: procedure PARTITION(A, p, r)
9:     x = A[r]
10:    i = p − 1
11:    for j = p to r − 1 do
12:        if A[j] < x then
13:            i = i + 1
14:            exchange A[i] with A[j]
15:        end if
16:        exchange A[i] with A[r]
17:    end for
18: end procedure
```

QuickSort Pseudocode Example

Another version