EECS 336: Introduction to Algorithms **Lecture 2**
Philosophy, Tractibility, Big-Oh

Reading: Chapters 2 & 3.

Announcements:

- Lecture notes on Canvas.
- Prerequisites:
  - EECS 212: Discrete Math.
  - EECS 214: Data Structures.
- Homework:
  - work with lab partner (meet up after class)
  - must communicate solution well.
  - peer review (can you tell if a solution is good)
  - automatic late policy for 25% of grade.

Last Time:

- fibonacci numbers

Today:

- philosophy
- computational tractability
- runtime analysis & big-oh
- graphs & graph traversals

Algorithms Design and Analysis

gives rigorous mathematical framework for thinking about and solving problems in CS and other fields.

Goals

- quickly compute solutions to problems.
- understand the essence of problem.
- identify general algorithm design and analysis approaches.

Three Steps

1. problem modeling: abstract problem to essential details.
2. algorithm design
3. algorithm analysis
   - efficiency,
   - correctness, and
   - (sometimes) "quality".

Note: design and analysis of good algorithms requires deep understanding of problem.

Computational Tractability

"is a problem solvable by a computer?"

Def: problem is *tractable* if worst-case runtime to compute solution is polynomial in size of input.

Question: What is "a problem"?

Answer: worst cases instances of a given size.

Question: Other possibilities?

- every instance?
- typical instances?
- random instances?

Question: Benefits?

- usually doable.
- often tight for typical or random instances.
- analyses "compose"

Def: $T(n)$ = worst case runtime of instances of size n.

- size n measured in bits, or
- number of "components".

Example: Fibonacci Numbers

fib(k) has $n = \log k$ bits.

- recursive: $T(n) \approx 2^{2^n}$.

- dynamic program / iterative alg: $T(n) \approx 2^n$.
- repeated squaring: $T(n) \approx n$.

Question: What is "solvable by a computer"?

Answer: $T(n) =$ polynomial.

- want to solve "large" instances.
- want to scale well.

  i.e., $T(cn) \leq dT(n)$.

$\Rightarrow$ $T(n)$ should be *polynomial*.

Example:

$$T(n) = n^k$$
$$T(cn) = (cn)^k = \underbrace{c^k}_{d} n^k = dn^k.$$

Efficient vs. Brute-force

- brute-force: "try all solutions, output best one"
- often runtime of brute-force $\geq$ exponential time
- efficient algorithms require exploiting structure of problem.

Main goals for algorithm design

1. show problem is tractable
   exists algorithm with polynomial runtime.
2. show problem is intractable
   for all algorithms, runtime is super-polynomial.

Question: Which is easier?

Answer: showing tractable.

Runtime Analysis

"bound $T(n)$ for algorithm"

Big-Oh Notation

Def: $T(n)$ is $O(f(n))$ if $\exists n_0, c > 0$ such that $\forall n > n_0,\ T(n) < cf(n)$.

Question: why?

Answer:

- exact analysis is too detailed.
- constants vary from machine to machine.

Example:

$$\begin{aligned} T(n) &= an^2 + bn + d \\ &= O(n)?\ O(n^2)?\ O(n^3)? \\ T(n) &\leq an^2 + bn^2 + dn^2 \\ &= \underbrace{(a+b+d)}_{c} n^2 \\ &\leq cn^3 \end{aligned}$$

Fact 1: $f = O(g) \& g = O(h) \Rightarrow f = O(h)$.

Fact 2: $f = O(h) \& g = O(h) \Rightarrow f + g = O(h)$.

Fact 3: $g = O(f) \Rightarrow g + f = O(f)$.

Proof: (of Fact 2)

$f = O(h) \Rightarrow \exists c, n_0$ such that $\forall n > n_0,\ f(n) < ch(n)$

$g = O(h) \Rightarrow \exists c', n_0'$ such that $\forall n > n_0',\ g(n) < c'h(n)$

$\Rightarrow \forall n > \max(n_0, n_0'),\ f(n) + g(n) \leq (c' + c)h(n)$

$\Rightarrow f + g = O(n)$.

QED

Note:

- be succinct: do not write $O(n^2 + n)$, $O(5n)$, etc.
- be tight: if $T(n)$ is n^2 do not say $T(n)$ is $O(n^3)$.

Logarithms and Big-Oh

Def: $\log_b n = \ell \Leftrightarrow b^\ell = n$

- $\log_{10} n$ = number of digits to represent n.
- $\log_2 n$ = number of bits to represent n.

Fact 4: $\forall b, c, \log_b n = O(\log_c n)$

Fact 5: $\forall b, x, \log_b n = O(n^x)$.

Proof: (of Fact 4)

$$\begin{aligned} \log_c n = \ell &\Rightarrow n = c^\ell \\ \log_b n &= \log_b(c^\ell) \\ &= \ell \log_b c \\ &= \log_c n \underbrace{\log_b c}_{d} \\ &= O(\log_c n) \end{aligned}$$

Common Runtimes

$O(\log n)$ – logarithmic

$O(n)$ – linear

$O(n \log n)$

$O(n^2)$ – quadratic

$O(n^3)$ – cubic

$O(n^k)$ – polynomial

$O(2^n)$ – exponential

$O(n!)$

Lower bounds

Def: $T(n)$ is $\Omega(f(n))$ if $\exists n_0, c > 0$ such that $\forall n > n_0, T(n) > cf(n)$.

Exact bounds

Def: $T(n)$ is $\Theta(f(n))$ if $T(n)$ is $O(f(n))$ and $\Omega(f(n))$.

Graphs

"encode pair-wise relationships"

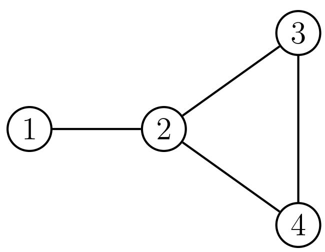
Examples: computer networks, social networks, travel networks, dependencies.

vertices

$G = (V, E)$

edges

Example:



- $V = \{1, 2, 3, 4\}$
- $E = \{(1, 2), (2, 3), (2, 4), (3, 4)\}$

Concepts

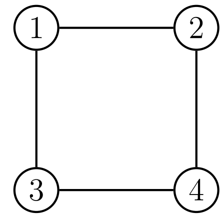
- degree
- neighbors
- paths, path length
- distance
- connectivity, connected components
- directed graphs.

Graph Traversals

"visit all the vertices in a connected component of graph"

- Breadth First Search (BFS).

  Example:

  BFS from 1: 1, 2, 3, 4 or 1, 3, 2, 4.

- Depth First Search (DFS).

  Example: DFS from 1: 1, 2, 4, 3 or 1, 3, 4, 2.