CSE 101, Winter 2016

Design and Analysis of Algorithms

Instructor: Prof. Andrew B. Kahng
Webpage: http://vlsicad.ucsd.edu/~abk/
Email: abk@cs.ucsd.edu
Telephone: 858-822-4884
Office Hours: Wed 12:30-1:30pm, Fri 1-2pm, and by appt
Office: EBU3B 2134

Class URL: http://vlsicad.ucsd.edu/courses/cse101-w16/
Goals of Course

• Introduction to design and analysis of algorithms
• “Problem-solving”
• Classic Problems
  – Sorting, Path-Finding, String-Matching, Arithmetic, …
• Tools
  – Recurrence Relations, Counting Techniques, Reduction,
    Probabilistic Analysis, NP-Completeness, …
• Frameworks
  – Divide-and-Conquer, Greed, Dynamic Programming, Branch-and-
    Bound, Heuristics, …
• Pedagogical choices
  – Order of material
  – “Analyze” vs. “Create”  I stress the latter = problem-solving
  – Scope  You need to keep up
What Is An Algorithm?

• An algorithm is a method for solving a problem (on a computer)

• Problem: “Given fraction m/n, reduce to lowest terms.”

• An algorithm must be effective
  → give a correct answer and terminate

• Problem: “Given undirected graph G = (V,E) and vertices s,t ∈ V, is there a path in G from s to t ?”

• QUESTION: State an algorithm for this problem
Undirected s-t Connectivity

- A1: BFS, DFS from s.
- A2: Take a random walk in G, starting at s.
  - *Is this an algorithm? (Does it halt?)*
- A3: Take a random walk in G for $5n^3$ steps starting at s ($n = |V|$); return NO iff we don’t visit t.
  - *Is this an algorithm?*
  - *Does it “almost always” return the correct answer?*

- Do A3, A1 differ in terms of **resources** used?
  - A3 “trades” time for space, is “memoryless”.
  - A3: **probabilistic** effectiveness.
Problem-Solving

- Problem solving = “The Spirit of Computing”
- Driven by real-world necessity
  - DNA Sequencing
    - Evolutionary Trees (edit distance, Steiner trees...)
    - Finding homologues, evolutionary significance (string-matching)
  - Conformational Analysis
    - Drug Design (minimum-energy configuration)
  - Autonomous Robotics, Vehicles
    - (managing smart highways, collision avoidance / path planning, …)
  - Logistics
    - (scheduling, resource allocation, …)
    - (stowage on container ships; airline logistics; …)
  - Design of integrated circuits
    - (placement, wiring, partitioning, floorplanning, clock distribution, logic synthesis, …)
Problem-Solving

• Patterns
  – Zeitz, The Art and Craft of Problem Solving
  – Polya, How to Solve It

• Tools
  – Counting
  – Recurrence Relations
  – Data Structures
  – ...

• Concepts
  – Problem classes and “solution classes”
  – Lower bounds $\rightarrow$ at least this hard, at least this much effort
  – Reductions $\rightarrow$ solving this boils down to solving that
  – Intractability $\rightarrow$ believed impossible to solve efficiently
What is a Problem?

• A **problem** is defined by:
  – (i) **input domain**
    • e.g., all ordered pairs of positive integers
  – (ii) **output specification**
    • e.g., convert to an equivalent fraction in lowest terms

• A problem with the input specified is a **problem instance**
  – e.g., “convert the fraction 343/56 to lowest terms”

• **Types of Problems:**
  – **Decision**
    • Yes or No answer (e.g., Does there exist...?)
  – **Computation**
    • How many acyclic s-t paths in G?
  – **Construction** (more than one answer)
    • Construct (exhibit) an s-t path in G. *any s-t path, vs. shortest s-t path, vs. ...*
  – **Optimization** (set of all alternatives; cost function)
    • Determine the *shortest* s-t path in G.
Problem-Solving First Example

• Tower of Hanoi
  – Rules: (i) One disk moves at a time, and (ii) Never put a larger disk onto a smaller disk
  – If you move one disk per second, when will all 64 disks be moved?
  – A more useful question: What is the minimum # of moves needed to transfer a stack of n disks?
    • Why is this more useful? → Assumes optimal strategy, ...

• Step 1. Define Notation
  – For a stack of n disks, call this number $T_n$

• Step 2. Look At Small Cases
  – $T_0 = 0$, $T_1 = 1$, $T_2 = 3$
Problem-Solving First Example (cont.)

- **Step 3. Can we reduce to a known problem?**
  - \( T_n \leq 2T_{n-1} + 1 \), \( n > 0 \)
  - Why?
    - Shift \((n-1)\), move largest disk, shift again
  - Why \( \leq \) inequality?
    - \( 2T_{n-1} + 1 \) suffices, but maybe can do better
  - Why does the lower bound (LB) \( T_n \geq 2T_{n-1} + 1 \) hold?
    - Must move largest disk sometime; at this instant, have \((n-1)\) on a single peg
  - \( \Rightarrow T_n = 2T_{n-1} + 1, \ T_0 = 0 \)

- **Step 4. What is a general solution for \( T_n \)?**
  - \( \{ T \} = 0, 1, 3, 7, 15, 31, 63, \ldots \)
Problem-Solving First Example (cont.)

- Looks like $T_n = 2^n - 1$
- **Step 5. let’s guess** this answer and try to prove it
  - **Claim:** $T_n = 2^n - 1$
  - **Proof:** (by mathematical induction)
    - **Basis** ($n = 0$): $T_0 = 0 = 2^0 - 1$ holds
    - **I.H.:** $T_k = 2^k - 1 \ \forall k = 0, 1, \ldots, n - 1$
    - **I.S.:** $T_n = 2T_{n-1} + 1 = 2(2^{n-1} - 1) + 1 = 2^n - 1$ (I.H.)

- **Note the kinds of steps that we applied…**
  - Establish Notation, Study Small Cases,
  - Reduce to a Known Problem, Seek a General Solution,
  - Prove the Solution
Question: What Makes One Algorithm Better (or Worse) Than Another?

- **Efficiency with respect to resources** (= one aspect)

- Example: Determinant
  - det (2 × 2 matrix) = ad − bc
  - Recursively defined det M = (…)
    - M_{1j} is the (1, j) cofactor matrix of n x n matrix M

- Problem: Give an algorithm for computing det M
  - A1: Use definition to get recursive algorithm
    - *How many multiplications?* (About n!)
  - A2: Use Gaussian elimination to get lower-triangular M’
    - If M’ is lower-triangular, det M’ = \( \prod m’_{ii} \)
    - *How many multiplications?* (About n^3)
      - det M’ / (some scalar) = det M  
        // scalar is from row operations
  - 1988 algorithms textbook (Brassard and Bratley):
    For n = 20, A1 takes 107 years; A2 takes 0.05 seconds
Problem-Solving Second Example

• Recall from before: Find m/n in lowest terms
• “Formal” Statement:
  – Input: integers m ≥ 0, n > 0
  – Output: integers m’, n’ s.t. m/n = m’/n’, (m,n) = (m’,n’), and m’, n’ are relatively prime.
• A1:
  – Cancel all 2’s
  – Cancel all 3’s
  – Cancel all 5’s
  – etc. until min (m,n) exceeded
• Why is this not so clever?
  – What’s the “worst case”?
  – We always have to check up to min (m,n)
Problem-Solving Second Example (cont.)

• A1’: // try divisors starting with largest possible
  – i $\leftarrow$ min (m,n) + 1
  – repeat i $\leftarrow$ i - 1 until ((i|m) and (i|n))
  – return i
  – may get lucky and stop after only a few divisions
  – but, worst case: m $\approx$ n, (m,n) = 1

• A2:
  – find gcd(n,m) // gcd = greatest common divisor
  – return m’ = m / gcd(n,m) , n’ = n / gcd(n,m)
  – We have recast the problem as finding gcd
- $\text{gcd}(n,m)$ [Euclid’s Algorithm] \hspace{1cm} // assume w.l.o.g. $n > m$
  \hspace{1cm} while \hspace{0.7cm} $m > 0$ \hspace{0.7cm} do
  \hspace{1.7cm} $t \leftarrow n \mod m$
  \hspace{1.7cm} $n \leftarrow m$
  \hspace{1.7cm} $m \leftarrow t$
  \hspace{1cm} return $n$

Example: Calculation of $\text{gcd}(81,21)$

$\rightarrow (21,18)$

$\rightarrow (18,3)$

$\rightarrow (3,0)$

$\text{gcd}(81,21) = 3$
Problem-Solving Second Example (cont.)

- gcd(n,m) [Euclid’s Algorithm]  // assume w.l.o.g. n > m

while m > 0 do
  t ← n mod m
  n ← m
  m ← t
return n

- Claim: If n > m then gcd(n,m) = gcd(m,n-m)
  • How do you prove an equality? Prove both inequalities.

- Proof: (1st inequality) Want gcd(m,n-m) ≥ gcd(n,m)
i.e., if z|m and z|n then z|m, z|(n-m)

z|m and z|n  ⇒  m mod z = n mod z = 0
  ⇒  (n-m) mod z = 0
  ⇒  z|(n-m)
Problem-Solving Second Example (cont.)

- **gcd(n,m) [Euclid’s Algorithm]** // assume w.l.o.g. n > m

  while m > 0 do
    \[t \leftarrow n \mod m\]
    \[n \leftarrow m\]
    \[m \leftarrow t\]
  return n

- **Claim**: If \(n > m\) then \(\gcd(n,m) = \gcd(m,n-m)\)
  
  • *How do you prove an equality?  Prove both inequalities.*

- **Proof**: (2\textsuperscript{nd} inequality) Want \(\gcd(m,n-m) \leq \gcd(n,m)\)
  
  i.e., if \(z|m, z|(n-m)\) then \(z|m, z|n\)

  \[z|m \text{ and } z|(n-m) \Rightarrow [m+(n-m)] \mod z = 0\]
  
  \[\Rightarrow z|n\]
Proving That the Algorithm is “Good”

- Euclid’s Algorithm is correct. *Is it efficient?*
- How many times can we go through main loop of \( \text{gcd}(n,m) \)?
  - Suppose \( m \) halves each time? (It doesn’t...)
    \[ \Rightarrow \text{Then, } \log_2 m \text{ would be an upper bound on } \# \text{ passes} \]
  - *Is any geometric decrease good enough?*

- **Notation:**
  - \((n_i, m_i)\) are values after \( i^{\text{th}} \) pass
  - Assume \( n_0 \geq m_0 \)
  - Loop is executed a total of \( L \) times
Proving That the Algorithm is “Good”

gcd(n,m) [Euclid’s Algorithm]  \((assumes \ n > m)\)

\[
\text{while } m > 0 \text{ do} \\
\quad t \leftarrow n \mod m \\
\quad n \leftarrow m \\
\quad m \leftarrow t \\
\text{return } n
\]

• **Notation:**
  – \((n_i, m_i)\) are values after \(i^{th}\) pass
  – Assume \(n_0 \geq m_0\)
  – Loop is executed a total of \(L\) times

• **Claims:**
  – (i) \(m_i \leq n_i\) \(\forall\ 0 \leq i \leq L-1\) (true from algorithm statement)
  – (ii) \(n_{i+1} = m_i\) (true from algorithm statement)
  – (iii) \(m_{i+1} \leq n_i / 2\) **[Case 1:} m_i \leq n_i / 2 \rightarrow m_{i+1} \leq n_i / 2 \text{ since } m_{i+1} < m_i.**
  **Case 2:** \(m_i > n_i/2 \rightarrow m_{i+1} = n_i \mod m_i = n_i - m_i \leq n_i/2.\]
Proving That the Algorithm is “Good”

\[
gcd(n,m) \ [\text{Euclid’s Algorithm}] \quad (\text{assumes } n > m)
\]

\[
\text{while } m > 0 \ \text{do}
\]
  \[
  t \leftarrow n \mod m \\
  n \leftarrow m \\
  m \leftarrow t
\]
\[
\text{return } n
\]

- **Claims:**
  - (i) \( m_i \leq n_i \quad \forall \ 0 \leq i \leq L-1 \) (true from algorithm statement)
  - (ii) \( n_{i+1} = m_i \) (true from algorithm statement)
  - (iii) \( m_{i+1} \leq n_i / 2 \)

- **Theorem:** \( m_{i+2} \leq m_i / 2 \)
  - **Proof:**
    - (ii) \( \Rightarrow n_{i+1} = m_i \)
    - (iii) \( \Rightarrow m_{i+2} \leq n_{i+1} / 2 \)
  - **Corollary:** If \( n_0 \geq m_0 \geq 1 \), then \( L \leq 2 \log_2 m_0 + 1 \)
Basketball Before You Were Born

• No 3-point field goal
• Hypothetical game score: UCSD 75, UCLA 64
• Assuming no 3-pointers, in how many ways could UCSD have accumulated 75 points?

• Notation:
  – \( S(n) \equiv \# \text{ ways to score } n \text{ points} \)

• Small Cases:
  – \( S(0) = 1 \)
  – \( S(1) = 1 \)
  – \( S(2) = 2 \) \ 2 \text{ or } 1-1
  – \( S(3) = 3 \) \ 2-1 \text{ or } 1-2 \text{ or } 1-1-1

\text{Is this familiar?}
A “Recurrence Relation”

- **Problem:** What is \( S(75) \)?
  - **Notation:** write \( F(n) = S(n-1) \)
    \[
    F(1) = F(2) = 1; \quad F(n) = F(n-1) + F(n-2)
    \]

- **Fibonacci:** 1, 1, 2, 3, 5, 8, …
- **So,** \( S(75) \) is the 76\(^{th} \) Fibonacci number

- **Solving the recurrence:** See Slide 37
Choosing Between Solutions

• Criteria:
  – Correctness
  – Time resources
  – Hardware resources
  – Simplicity, clarity (practical issues)

• Will need:
  – Size, Complexity measures
  – Notion of “basic” machine operation
The Basketball Question Again

• We wanted $S(75) = F(76)$, i.e., the 76th Fibonacci number

• “Give an efficient algorithm.”
  – For now, let’s equate “efficient” with “using few ‘elementary’ machine operations”; we will ignore size of operands and other issues

• $\text{fib1}(n)$ if $n < 2$ then return $n$
  else return $\text{fib1}(n-1) + \text{fib1}(n-2)$
  – Analysis: $T(n) = 1$ if $n<2$; $T(n) = T(n-1) + T(n-2)$ otherwise
  $T(n) = F(n)$, i.e., around $(1.64)^n$

• **Question:** *What is wrong with $\text{fib1}$?*
Save Your Work! = Cache (Sub-)Solutions

• \( \text{fib2}(n) \)
  \[
  f[1] = 1; \quad f[2] = 2; \\
  \text{for } j = 3 \text{ to } n \text{ do} \\
  \quad f[j] = f[j - 1] + f[j - 2]
  \]

• Analysis: \( T(n) = n \)
  – Saving your work (“caching”) can be useful!
  – Similar example: Pascal’s triangle (binomial coefficients)
  – But, can we do better?

• Idea: Use “natural structure”
  – We are applying the recurrence \( n \) times. Are there any shortcuts?
Not Obvious, But Here Is A Shortcut…

• **fib3(n)**
  – Consider 2x2 matrix M: \( m_{11} = 0, m_{12} = 1, m_{21} = 1, m_{22} = 1 \)
  – Observe: \( [F(k) \ F(k+1)]^T = M \times [F(k-1) \ F(k)]^T \)
    \[
    [F(n+1) \ F(n+2)]^T = M^n \times [F(1) \ F(2)]^T = M^n \times [1 \ 1]^T
    \]

  – **How does this help?**
  – **Hint:** \( 76_{10} = 1001100_2 \)

• \( M^{76} = M^{64} \times M^8 \times M^4 \)

• \( \rightarrow \) fib3 uses “addition chains”
Quantifying “Better”, “Worse”

- Resources used depend on a **natural parameter**, \( n \), of the input
  - search/sort list  \# items  \( x > y \)
  - matrix mult  largest dim  \( x \times y \); \( x + y \)
  - traverse tree  \# nodes  follow ptr

- Asymptotic Notation “as \( n \) grows large”
  - \( f \in \mathcal{O}(g) \) if \( \exists c_1, c_2 > 0 \) s.t. \( f(n) \leq c_1 g(n) + c_2 \ \forall n > 0 \)
  - \( f \in \mathcal{O}(g) \) if \( \exists c > 0, N \) s.t. \( \forall n > N, f(n) \leq cg(n) \)
    - e.g., \( 200x^2 \in \mathcal{O}(2x^{2.5}) \)
  - \( f \in \Omega(g) \) if \( g \in \mathcal{O}(f) \)
  - \( f \in \varTheta(g) \) if \( g \in \mathcal{O}(f) \) and \( f \in \mathcal{O}(g) \)
  - \( f \) is \( o(g) \) if \( \lim_{n \to \infty} \frac{f(n)}{g(n)} = 0 \)
Using “Big-Oh” Notation

• Definition: \( f(n) \) is monotonically growing (non-decreasing) if \( n_1 \geq n_2 \Rightarrow f(n_1) \geq f(n_2) \)

• Theorem: For all constants \( c > 0, a > 1, \) and for all monotonically growing \( f(n) \), \( (f(n))^c \in O(a^{f(n)}) \)

• Corollary \((take \ f(n) = n)\): \( \forall \ c > 0, a > 1, \ n^c \in O(a^n) \)
  – Any exponential in \( n \) grows faster than any polynomial in \( n \)

• Corollary \((take \ f(n) = \log_a n)\): \( \forall \ c > 0, a > 1, \ (\log_a n)^c \in O(a^{\log_a n}) = O(n) \)
  – Any polynomial in \( \log n \) grows slower than \( n^c', c'>0 \)

  • Exercise: \( f \in O(s), g \in O(r) \Rightarrow f+g \in O(s+r) \)
  • Exercise: \( f \in O(s), g \in O(r) \Rightarrow f\times g \in O(s\times r) \)

• So, we can count operations in an asymptotic sense. \( But, \ what \ is \ an \ “operation” \ ??? \)
What Do We Measure?

• Traditional metrics:
  – Program Size static
  – Runtime dynamic
  – Memory Usage dynamic

• Best Case (not informative)
  – e.g., Bubble Sort? Insertion Sort? Quicksort?

• Worst Case (easiest, most common)
  – $t_A(I) \equiv$ time used by algorithm $A$ on instance $I$
  – $D(n) \equiv$ set of all instances of size $n$
  – $WC_A(n) = \max \{t_A(I) \mid I \in D(n)\}$ max time taken by alg $A$ over all instances of size $n$

• Average Case (useful, but often less tractable)
  – $p(I) \equiv$ probability that instance $I$ occurs
  – $AC_A(n) = \sum_{I \in D(n)} p(I)t_A(I)$ average time taken by alg $A$ over all instances of size $n$

• Amortized Effort (avg over series of operations)
Can Characterize **Problem** Complexity

- **Upper Bounds:**
  - Alg A has UB $f(n)$: $\forall I \in D(n), t_A(I) \leq f(n)$
  - Problem P has UB $f(n)$: $\exists$ Alg A for P with UB
  - P has UB $O(f)$: $\exists$ Alg A with UB $g(n)$; $g \in O(f)$

- **Lower Bounds:**
  - Alg A has LB $f(n)$: $\exists$ infinitely many $n$ s.t. $\exists I \in D(n)$ where $t_A(I) \geq f(n)$
  - Problem P has LB $f(n)$: $\forall$ Alg A for P, $\exists$ infinitely many $m$ s.t. $\exists I \in D(m)$ for which $t_A(I) \geq f(m)$

- **How Do We Argue UB?**
  - Constructively (, reductions)

- **How Do We Argue LB?**
  - e.g., comparison tree model, reductions
Comparison-Based LB Arguments - Sorting

- Observe: Sorting \equiv Identifying Permutation
- Binary Tree: Root at level (height) 0
- **Theorem:**
  - \( \exists c > 0 \) s.t. \( \forall \) algorithms which use comparisons to sort, and \( \forall \) input sizes \( n \), at least one input requires \( cn \log n \) comparisons
- **Fact:**
  - Binary tree of height \( h \) has at most \( 2^h \) leaves
- Observe:
  - \( n! \) leaves needed \( \Rightarrow \) decision (comparison) tree must have \( h \geq \log(n!) \), where \( h \) is max \# comparisons needed to sort input of size \( n \) using the corresponding algorithm
Fun (!), Interesting, Useful Questions

• **MaxMin**
  - Given a list of N numbers, return the largest and smallest.

• **Finding a Celebrity**
  - Given a set S of N people, assume that for any pair I, J exactly one of the following is true: I “knows” J, or J “knowns” I. Further, define a “celebrity” as someone who knows no one (and who is therefore known by everyone else). Given the “knows” relation over S, determine whether S contains a celebrity.

• **Reduction**
  - **SORTING problem**
    - Input: a set of numbers
    - Output: the elements of the set, in sorted order
  - **CONVEX HULL problem**
    - Input: a set of points in $\mathbb{R}^2$
    - Output: the convex hull of these points, i.e., polygon vertices in order

→ Is “ease” of SORTING “related” to “ease” of CONVEX HULL?
Administrative Notes, January 5

- Slides will be posted in advance of lectures
  - Any updated slides and notes will be posted after lecture
- Programming assignments will be due Fridays of Weeks 3, 5, 7, 9
- Homeworks will be due Thursdays on Weeks 2, 4, 6, 8, 10
- Please pay attention to the class webpage and to Piazza!
  - Piazza = discussion boards and announcements
  - Gradesource = posted grades
  - Discussion sections, TA OH’s and my OH’s are all posted on the class webpage
- **HW #1** is posted (due Thursday of Week 2 at the beginning of class)
EXTRA SLIDES
Motivation for a Resource Model

When we count big-O time complexity, what operations take “unit time?"

- Suppose \texttt{factorial}, \texttt{mod} are “unit-cost” on some computer.
  \begin{verbatim}
  WILSON(n)
  if (n-1)! +1 \equiv 0 \mod n then return TRUE
  else return FALSE
  \end{verbatim}
  – one-step primality testing – but this sounds fishy…

- What if \texttt{return max}_i \texttt{x}_i was “unit-cost”?
  – Is this reasonable, given that there is a speed-of-light limit to signal propagation on wires, and finite (non-zero) dimensions of transistors and wires?
  – Physical models (what can be embedded in our 3-D world) are increasingly relevant!
The RAM (Random-Access Machine) Model

- finite stored program
- finite collection of registers
  - each stores single integer or real
- array of n words of memory
  - each stores single integer or real
  - has unique address in [1, ..., n]
- In one step:
  - Perform arithmetic, logical operation on register content
  - \( R_j := M_{R_k} \) or \( M_{R_j} := R_k \) (access contents of word whose address is in register)
  - JNZ, HALT, etc.
The RAM Model (cont.)

- **Q:** On a RAM machine, how large a number can be manipulated in constant time?

- Two variants:
  - uniform cost
  - log cost

- **Exercise:** What are costs for each, under the two variants?
  1. **sum_1_to_N(n)**
     ```plaintext
     sum ← 0
     for i ← 1 to n do sum ← sum + i
     return sum
     ```
  2. **fib4(n)**
     ```plaintext
     i ← 1, j ← 0
     for k ← 1 to n do
       j ← i + j
       i ← j - i
     return j
     ```

- Other: Turing, pointer machines; straight-line program, decision/comparison tree, ...
Addendum: Solving the Fibonacci Recurrence

• **Problem:** What is $S(75)$?
  
  – **Notation:** write $F(n) = S(n-1)$
    
    $F(1) = F(2) = 1$; $F(n) = F(n-1) + F(n-2)$
  
  – **Guesses:** try $F(n) = a^n$ for some $a$
    
    $a^n = a^{n-1} + a^{n-2} \Rightarrow a^2 = a + 1 \Rightarrow a^2 - a - 1 = 0$
    
    Roots: $a_1 = (1 + \sqrt{5})/2$; $a_2 = (1 - \sqrt{5})/2$
    
    Inspection: $F(n)$ seems close to $(a_1)^n$ *What’s missing?*
  
  – **Use all of the information**
    
    $F(1) = 1$; $F(2) = 1$ (initial conditions)
  
  – Homogeneous linear recurrence: any linear combination of $(a_1)^n$, $(a_2)^n$ is also a solution.
    
    - $c_1 (a_1)^1 + c_2 (a_2)^1 = F(1) = 1$ ; $c_1 (a_1)^2 + c_2 (a_2)^2 = F(2) = 1$
    
    - Get $c_1 = 1 / \sqrt{5}$, $c_2 = -1 / \sqrt{5}$
    
    - 1845 result of Lame (see Knuth, volume 2, section 4.5.3): If $m,n \leq F(k)$, then $L$ in $\gcd(m,n) \leq k$, with equality when $(m,n) = (F(k-1),F(k))$.  