Analysis of Algorithms
CS 477/677

Instructor: Monica Nicolescu

Lecture 3

Asymptotic Notations

- A way to describe behavior of functions in the limit
  - How we indicate running times of algorithms
  - Describe the running time of an algorithm as n grows to ∞
- O notation: asymptotic "less than": f(n) "≤" g(n)
- **Ω** notation: asymptotic "greater than": f(n) "≥" g(n)
- Θ notation: asymptotic "equality": f(n) "=" g(n)

More on Asymptotic Notations

- There is no unique set of values for n_0 and c in proving the asymptotic bounds
- Prove that $100n + 5 = O(n^2)$
  - $100n + 5 \le 100n + n = 101n \le 101n^2$

    for all $n \ge 5$

    $n_0 = 5$ and $c = 101$ is a solution
  - $100n + 5 \le 100n + 5n = 105n \le 105n^2$

    for all $n \ge 1$

    $n_0 = 1$ and $c = 105$ is also a solution

Must find **SOME** constants c and n_0 that satisfy the asymptotic notation relation

Comparisons of Functions

- *Theorem:*

$$f(n) = \Theta(g(n)) \Leftrightarrow f = O(g(n)) \text{ and } f = \Omega(g(n))$$

- Transitivity:
  - $f(n) = \Theta(g(n))$ and $g(n) = \Theta(h(n)) \Rightarrow f(n) = \Theta(h(n))$
  - Same for O and Ω
- Reflexivity:
  - $f(n) = \Theta(f(n))$
  - Same for O and Ω
- Symmetry:
  - $f(n) = \Theta(g(n))$ if and only if $g(n) = \Theta(f(n))$
- Transpose symmetry:
  - $f(n) = O(g(n))$ if and only if $g(n) = \Omega(f(n))$

Asymptotic Notations in Equations

- On the right-hand side
  - $\Theta(n^2)$ stands for some anonymous function in $\Theta(n^2)$

  $2n^2 + 3n + 1 = 2n^2 + \Theta(n)$ means:

  There exists a function $f(n) \in \Theta(n)$ such that

  $$2n^2 + 3n + 1 = 2n^2 + f(n)$$

- On the left-hand side

  $2n^2 + \Theta(n) = \Theta(n^2)$

  No matter how the anonymous function is chosen on the left-hand side, there is a way to choose the anonymous function on the right-hand side to make the equation valid.

Some Simple Summation Formulas

- Arithmetic series:

$$\sum_{k=1}^{n} k = 1 + 2 + \ldots + n = \frac{n(n+1)}{2}$$

- Geometric series:

$$\sum_{k=0}^{n} x^k = 1 + x + x^2 + \ldots + x^n = \frac{x^{n+1} - 1}{x - 1} (x \neq 1)$$

  - Special case: $x < 1$:

$$\sum_{k=0}^{\infty} x^k = \frac{1}{1-x}$$

These should be known

- Harmonic series:

$$\sum_{k=1}^{n} \frac{1}{k} = 1 + \frac{1}{2} + \ldots + \frac{1}{n} \approx \ln n$$

- Other important formulas:

$$\sum_{k=1}^{n} \lg k \approx n \lg n$$

$$\sum_{k=1}^{n} k^p = 1^p + 2^p + \ldots + n^p \approx \frac{1}{p+1} n^{p+1}$$

Mathematical Induction

- Used to prove a sequence of statements ($S(1)$, $S(2)$, … $S(n)$) indexed by positive integers
- Proof:
  - **Basis step**: prove that the statement is true for $n = 1$
  - **Inductive step:** assume that $S(n)$ is true and prove that $S(n+1)$ is true for all $n \geq 1$
- Find case n "within" case $n+1$

Example

- Prove that: $2n + 1 \leq 2^n$ for all $n \geq 3$
- **Basis step:**
  - $n = 3$: $2 \times 3 + 1 \leq 2^3 \Longleftrightarrow 7 \leq 8$ TRUE
- **Inductive step:**
  - Assume inequality is true for n, and prove it for (n+1)

Assume: $2k + 1 \leq 2^k$ for all $k \leq n \Rightarrow 2n + 1 \leq 2^n$

Must prove is true for k=n+1: $2(n + 1) + 1 \leq 2^{n+1}$

$2(n + 1) + 1 = (2n + 1) + 2 \leq 2^n + 2 \leq$

$\leq 2^n + 2^n = 2^{n+1}$, since $2 \leq 2^n$ for $n \geq 1$

More Examples

$$\sum_{i=1}^{n} (2i - 1) = n^2 \; \forall \, n \geq 1$$

$$n! \geq 2^{n-1} \; \forall \, n \geq 1$$

Analysis of Recursive Algorithms

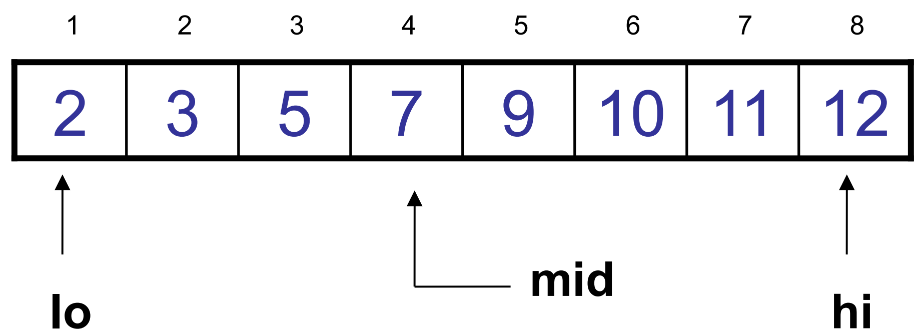
- A recursive algorithm calls itself on a smaller-sized input
  - **Base case:** the smallest instance of a problem, a condition that terminates the recursive function, returns a solution that can be computed directly
  - **Recursive case:** computes the result by making recursive calls with smaller inputs and applying simple operations to the returned values
- The running time of recursive algorithms cannot be computed only by counting primitive operations, due to the recursive calls

Recurrent Algorithms
BINARY – SEARCH

- for an ordered array A, finds if **x** is in the array A[lo…hi]

Alg.: BINARY-SEARCH (A, lo, hi, x)

if (lo > hi)
  return FALSE
mid ← $\lfloor (lo+hi)/2 \rfloor$
if x = A[mid]
  return TRUE
if (x < A[mid])
  BINARY-SEARCH (A, lo, mid-1, x)
if (x > A[mid])
  BINARY-SEARCH (A, mid+1, hi, x)



Example

- A[8] = {1, 2, 3, 4, 5, 7, 9, 11}
  - lo = 1 hi = 8 x = 7

1	2	3	4	5	6	7	8
1	2	3	4	5	7	9	11

mid = 4, lo = 5, hi = 8

1	2	3	4	5	7	9	11

mid = 6, A[mid] = x
Found!

Example

- A[8] = {1, 2, 3, 4, 5, 7, 9, 11}
  - lo = 1 hi = 8 x = 6

1	2	3	4	5	6	7	8	
1	2	3	4	5	7	9	11	mid = 4, lo = 5, hi = 8
1	2	3	4	5	7	9	11	mid = 6, A[6] = 7, lo = 5, hi = 5
1	2	3	4	5	7	9	11	mid = 5, A[5] = 5, lo = 6, hi = 5 NOT FOUND!

Analysis of BINARY-SEARCH

Alg.: BINARY-SEARCH (A, lo, hi, x)

if (lo > hi) ← constant time: c_1

return FALSE

mid ← ⌊lo+hi)/2⌋ ← constant time: c_2

if x = A[mid] ← constant time: c_3

return **TRUE**

if (x < A[mid])

BINARY-SEARCH (A, lo, mid-1, x) ← same problem of size n/2

if (x > A[mid])

BINARY-SEARCH (A, mid+1, hi, x) ← same problem of size n/2

- $T(n) = c + T(n/2)$
  - T(n) – running time for an array of size n

Recurrences and Running Time

- Recurrences arise when an algorithm contains recursive calls to itself
- What is the actual running time of the algorithm?
- Need to solve the recurrence
  - Find an explicit formula of the expression (the generic term of the sequence)

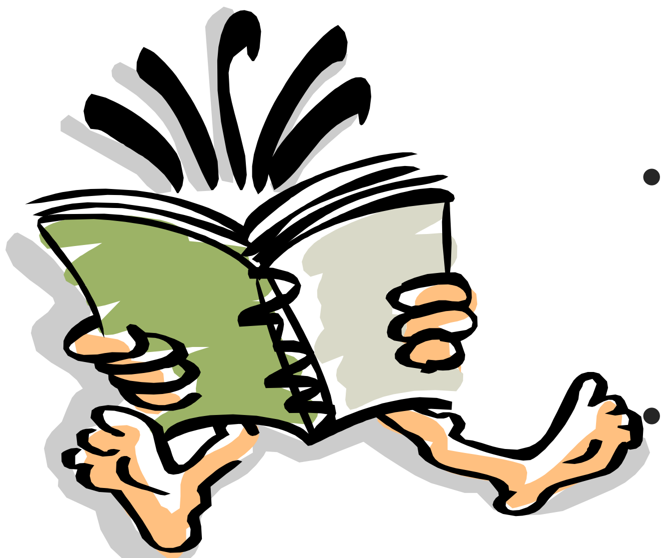
Example Recurrences

- $T(n) = T(n-1) + n$ $\Theta(n^2)$
  - Recursive algorithm that loops through the input to eliminate one item
- $T(n) = T(n/2) + c$ $\Theta(\lg n)$
  - Recursive algorithm that halves the input in one step
- $T(n) = T(n/2) + n$ $\Theta(n)$
  - Recursive algorithm that halves the input but must examine every item in the input
- $T(n) = 2T(n/2) + 1$ $\Theta(n)$
  - Recursive algorithm that splits the input into 2 halves and does a constant amount of other work

Methods for Solving Recurrences

- Iteration method
- Substitution method
- Recursion tree method
- Master method

Readings



- For this lecture
  - Chapter 4 intro
  - Apendix A
- Coming next
  - Sections 4.3, 4.4, 4.5