Discrete Mathematics

CS204: Spring, 2008

Jong C. Park
Computer Science Division, KAIST
Today’s Topics
Analysis of Algorithms
Recursive Algorithms
Analysis of Algorithms

- Theorem
  - Let \( p(n) = a_k n^k + a_{k-1} n^{k-1} + \ldots + a_1 n + a_0 \) be a polynomial in \( n \) of degree \( k \), where each \( a_i \) is nonnegative and \( a_k > 0 \). Then \( p(n) = \Theta(n^k) \).
  - Proof.
    - We first show that \( p(n) = O(n^k) \). Let \( C_1 = a_k + a_{k-1} + \ldots + a_1 + a_0 \).
      Then for all \( n \),
      \[
      p(n) = a_k n^k + a_{k-1} n^{k-1} + \ldots + a_1 n + a_0 \leq a_k n^k + a_{k-1} n^{k-1} + \ldots + a_1 n + a_0 n^k
      \]
      \[
      = (a_k + a_{k-1} + \ldots + a_1 + a_0) n^k = C_1 n^k.
      \]
      Therefore, \( p(n) = O(n^k) \).
    - Next, we show that \( p(n) = \Omega(n^k) \). For all \( n \),
      \[
      p(n) = a_k n^k + a_{k-1} n^{k-1} + \ldots + a_1 n + a_0 \geq a_k n^k = C_2 n^k,
      \]
      where \( C_2 = a_k \). Therefore, \( p(n) = \Omega(n^k) \).
    - It follows that \( p(n) = \Theta(n^k) \).
• Note
  – We shall use $\lg n$ to denote $\log_2 n$ (the logarithm of $n$ to the base 2).

• Example
  – Find the asymptotic tight bound for $2n + 3 \lg n$.
    • Since $\lg n < n$ for all $n \geq 1$, $2n + 3 \lg n < 2n + 3n = 5n$ for all $n \geq 1$. Thus, $2n + 3 \lg n = \mathcal{O}(n)$.
    • Also, $2n + 3 \lg n \geq 2n$ for all $n \geq 1$.
    • Thus, $2n + 3 \lg n = \Omega(n)$.
    • Therefore, $2n + 3 \lg n = \Theta(n)$. 

Analysis of Algorithms
Analysis of Algorithms

• Example
  – If $a > 1$ and $b > 1$ (to ensure that $\log_b a > 0$), by the change-of-base formula for logarithms, $\log_b n = \log_b a \log_a n$ for all $n \geq 1$.
  – Therefore, $\log_b n \leq C \log_a n$ for all $n \geq 1$, where $C = \log_b a$. Thus, $\log_b n = \Theta(\log_a n)$.
  – Also, $\log_b n \geq C \log_a n$ for all $n \geq 1$; so $\log_b n = \Omega(\log_a n)$.
  – Since $\log_b n = O(\log_a n)$ and $\log_b n = \Omega(\log_a n)$, we conclude that $\log_b n = \Theta(\log_a n)$.

• Note
  – For this reason, we sometimes simply write $\log$ without specifying the base.
• Example
  – Find the asymptotic tight bound for \( f(n) = 1 + 2 + \ldots + n \).
    • First, \( f(n) = O(n^2) \), since \( 1 + 2 + \ldots + n \leq n + n + \ldots + n \)
      \( = n \cdot n = n^2 \) for all \( n \geq 1 \).
    • Likewise, \( f(n) = \Omega(n) \), since \( 1 + 2 + \ldots + n \geq 1 + 1 + \ldots + 1 = n \) for all \( n \geq 1 \).
      However, we cannot deduce a \( \Theta \)-estimate for \( f(n) \) with this lower bound, since \( n^2 \neq n \). Thus, we need a tighter lower bound.
    • But by throwing away the first half of the terms, we get
      \[
      f(n) \geq \left\lceil \frac{n}{2} \right\rceil + \ldots + (n - 1) + n \\
      \geq \left\lceil \frac{n}{2} \right\rceil + \ldots + \left\lceil \frac{n}{2} \right\rceil + \left\lfloor \frac{n}{2} \right\rfloor \\
      = \left\lceil (n + 1)/2 \right\rceil \left\lfloor \frac{n}{2} \right\rfloor \geq (n/2)(n/2) \\
      = n^2/4
      \]
      for all \( n \geq 1 \). Thus \( f(n) = \Omega(n^2) \).
    • Therefore, \( f(n) = \Theta(n^2) \).
Analysis of Algorithms

• Examples
  – Find the asymptotic tight bound for \( f(n) = 1^k + 2^k + \ldots + n^k \).
    • \( f(n) = \Theta(n^{k+1}) \)
  – Show that \( \lg n! = \Theta(n \lg n) \).
    • Proof sketch.
      – \( \lg n! = \lg n + \lg (n-1) + \ldots + \lg 2 + \lg 1 \)
      \( \leq \lg n + \lg n + \ldots + \lg n + \lg n = n \lg n \) for all \( n \geq 1 \).
Example

- Show that if \( f(n) = \Theta(g(n)) \) and \( g(n) = \Theta(h(n)) \), then \( f(n) = \Theta(h(n)) \).

- Proof.
  - Because \( f(n) = \Theta(g(n)) \), there are constants \( C_1 \) and \( C_2 \) such that \( C_1|g(n)| \leq |f(n)| \leq C_2|g(n)| \) for all but finitely many positive integers \( n \).
  - Because \( g(n) = \Theta(h(n)) \), there are constants \( C_3 \) and \( C_4 \) such that \( C_3|h(n)| \leq |g(n)| \leq C_4|h(n)| \) for all but finitely many positive integers \( n \). Therefore, \( C_1C_3|h(n)| \leq C_1|g(n)| \leq |f(n)| \leq C_2|g(n)| \leq C_2C_4|h(n)| \) for all but finitely many positive integers \( n \).
  - It follows that \( f(n) = \Theta(h(n)) \).
• Definition
  – If an algorithm requires $t(n)$ units of time to terminate in the best case for an input of size $n$ and $t(n) = O(g(n))$, we say that the best-case time required by the algorithm is of order at most $g(n)$ or that the best-case time required by the algorithm is $O(g(n))$.
  – If an algorithm requires $t(n)$ units of time to terminate in the worst case for an input of size $n$ and $t(n) = O(g(n))$, we say that the worst-case time required by the algorithm is of order at most $g(n)$ or that the worst-case time required by the algorithm is $O(g(n))$. 
If an algorithm requires $t(n)$ units of time to terminate in the average case for an input of size $n$ and $t(n) = O(g(n))$, we say that the average-case time required by the algorithm is of order at most $g(n)$ or that the average-case time required by the algorithm is $O(g(n))$. 

Analysis of Algorithms
Example

- Determine, in theta notation, the best-case, worst-case, and average-case times required to execute the following algorithm.

Algorithm 4.3.17: Searching an Unordered Sequence

Input: $s_1, s_2, \ldots, s_n$, $n$, and key (the value to search for)

Output: The index of key, or if key is not found, 0

1. $linear\_search(s, n, key) \{
2.     for i = 1 to n
3.         if (key == s_i)
4.             return i \ // successful search
5.         return 0 \ // unsuccessful search
6. }$
• **Note**
  – A *recursive function* (pseudocode) is a function that invokes itself.
  – A *recursive algorithm* is an algorithm that contains a recursive function.

• **Example**
  – $n! = n(n - 1)(n - 2)\cdots 2 \cdot 1 = n \cdot (n - 1)!$
Recursive Algorithms

• Theorem
  – The following algorithm returns the value of $n!$, $n \geq 0$.
• Proof.
  – Use induction on $n$.

Algorithm 4.4.2: Computing $n$ Factorial

1. $factorial(n) \{
2. \quad$ if $(n == 0)$
3. \quad return 1
4. \quad return $n \ast factorial(n - 1)$
5. \}

Recursive Algorithms

• Example
  – A robot can take steps of 1 meter or 2 meters.
  – Find the number of ways the robot can walk $n$ meters.

```
Algorithm 4.4.6: Robot Walking

Input:   $n$
Output:  $walk(n)$

$walk(n)$ {
  if ($n == 1 \vee n == 2$)
    return $n$
  return $walk(n - 1) + walk(n - 2)$
}
```
Recursive Algorithms

• Example
  – Prove that the “Robot Walking” algorithm is correct.

• Note
  – Fibonacci sequence \( \{f_n\} \)
    • \( f_1 = 1, f_2 = 1, f_n = f_{n-1} + f_{n-2} \) for all \( n \geq 3 \).
  – Show that \( walk(n) = f_{n+1} \) for all \( n \geq 1 \).
    • Proof.
      – Use induction on \( n \).
  – Use mathematical induction to show that
    \[ \sum_{k=1}^{n} f_k = f_{n+2} - 1 \] for all \( n \geq 1 \).
Summary

- Analysis of Algorithms
- Recursive Algorithms