Algorithms

Introduction

Outlines

- Algorithm Analysis
  - Counting the number of operations
  - How to count
    - Worst case
    - Average case
    - Best case
  - Asymptotic notations
    - Upper bounds
    - Lower bounds
    - Tight bounds

Outlines

- Algorithm Design
  - Standard methods (for easy problems)
    - Greedy algorithms
    - Divide-and-conquer
    - Dynamic programming
    - Search
  - Advanced methods (for hard problems)
    - Probabilistic/randomized algorithms
    - Approximation algorithms

Typical Problems

- Sorting
  - Given a set of values, sort the values.
- Optimization
  - Given a set of Z such that …, find X from Z such that f(X) is optimal (min or max).
    - Given a graph G, find a path P from A to B in G such that length of P is smallest.
- Calculating
  - Given input X, calculate f(X).
    - Given vertices of a polygon X, calculate the area of X.
- Others

Algorithm Analysis

- Measure the resource required in an algorithm
  - Time
    - Running time
    - Disk access
  - Space
    - Memory
- When to measure
  - Best case
    - Bad!
  - Worst case
    - Upper bound
    - Mostly used, in general
    - Often occur
  - Average case
    - Often close to worst case
    - Used for randomized algorithm

Measure Running Time

- Basically, counting the number of basic operations required to solve a problem, depending on the problem/instance size

- What is a basic operations?

- How to measure instance size

Basic Operations

- What can be done in a constant time, *regardless of the size of the input*
  - Basic arithmetic operations, e.g. add, subtract, shift, …
  - Relational operations, e.g. <, >, <=, …
  - Logical operations, e.g. and, or, not
  - More complex arithmetic operations, e.g. multiply, divide, log, …
  - Etc.
- But sometimes these are not basic operations

Non-basic Operations

- When arithmetic, relational and logical operations are not basic operations?
- Operations depend on the input size
  - Arithmetic operations on very large numbers
  - Logical operations on very long bit strings
  - Etc.

Instance Size

- Number of elements
  - Arrays, matrices
  - Graphs (nodes or vertices)
  - Sets
  - Strings
- Number of bits/bytes
  - Numbers
  - Bit strings
- Maximum values
  - Numbers

Example

Number of times the while loop repeats for each value j

Best case: 1

Worst case: j

INSERTION-SORT(A)		cost	times
for $j \leftarrow 2$ **to** *length*$[A]$		c_1	n
do *key* $\leftarrow A[j]$		c_2	$n-1$
$\triangleright$ Insert $A[j]$ into the sorted sequence $A[1\,..\,j-1]$.		0	$n-1$
$i \leftarrow j-1$		c_4	$n-1$
while $i > 0$ and $A[i] > key$	***(n+2)(n-1)/2***	c_5	$\sum_{j=2}^{n} t_j$
do $A[i+1] \leftarrow A[i]$	***n(n-1)/2***	c_6	$\sum_{j=2}^{n} (t_j - 1)$
$i \leftarrow i-1$	***n(n-1)/2***	c_7	$\sum_{j=2}^{n} (t_j - 1)$
$A[i+1] \leftarrow key$		c_8	$n-1$

From: Cormen, T., C. Leiserson, R. Rivest, and C. Stein, *Introduction to Algorithms*, MIT Press, 2001.

Measure Disk Access

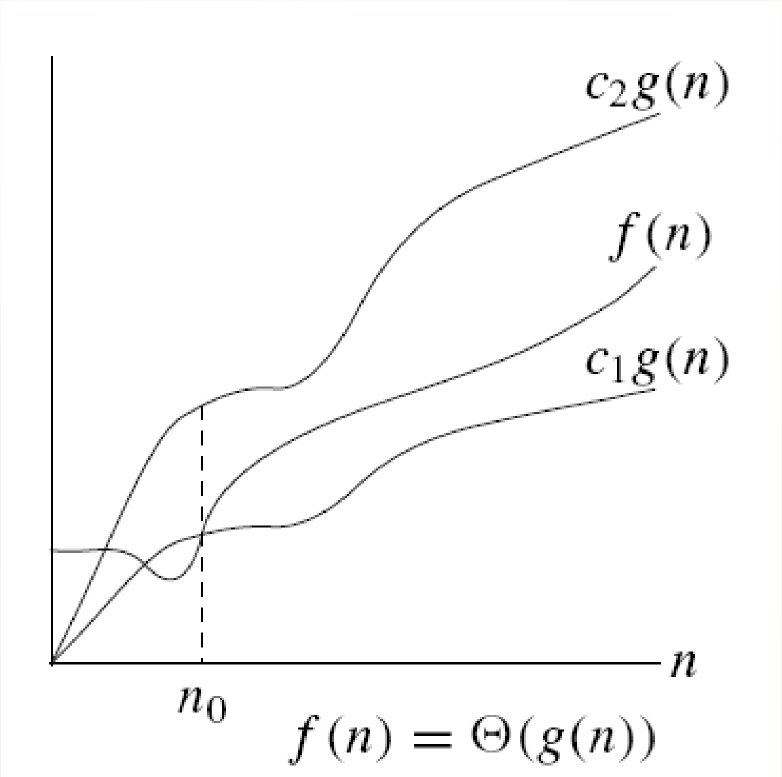
- Use for disk-based algorithm
- Measure the performance of
  - Index structures
  - Databases
  - Retrieval algorithm
- Cost of other operations is considered negligible, compared to disk access.

Measure Used Memory

- Measure the memory space required during runtime.
- Memory can be reused within the algorithm
- Mandatory in some applications with memory limitation

Asymptotic notations

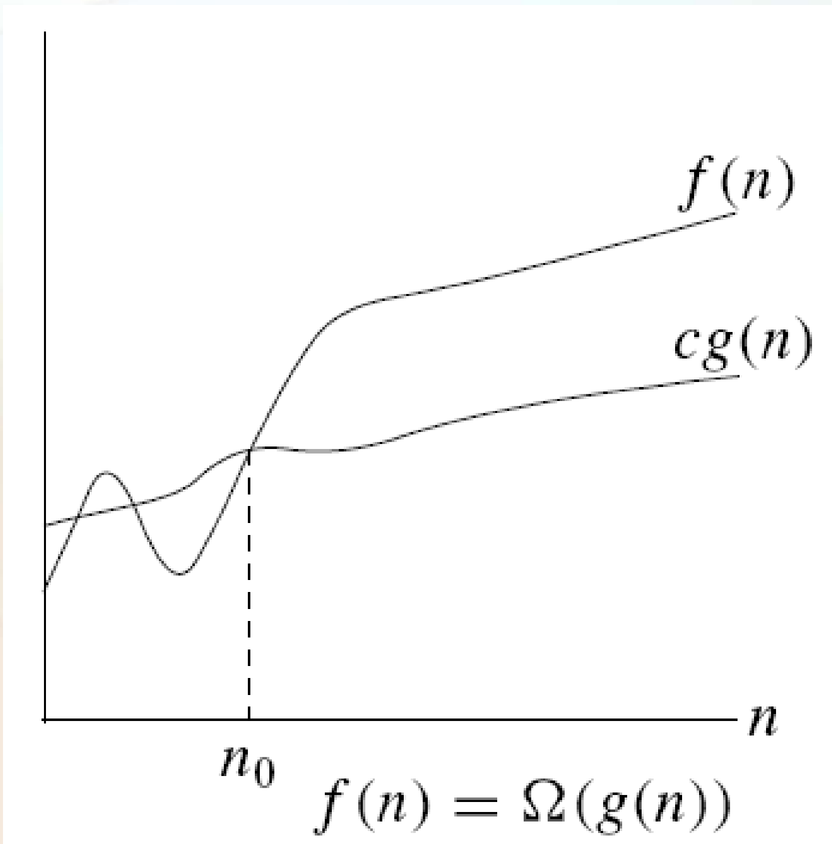
- Asymptotic tight bound Θ
  - $\Theta(g(n))$ = {$f(n)$| there are positive constants c,d and n_0 such that $c \cdot g(n) \le f(n) \le d \cdot g(n)$ for all $n \ge n_0$}



From: Cormen, T., C. Leiserson, R. Rivest, and C. Stein, *Introduction to Algorithms*, MIT Press, 2001.

Asymptotic notations

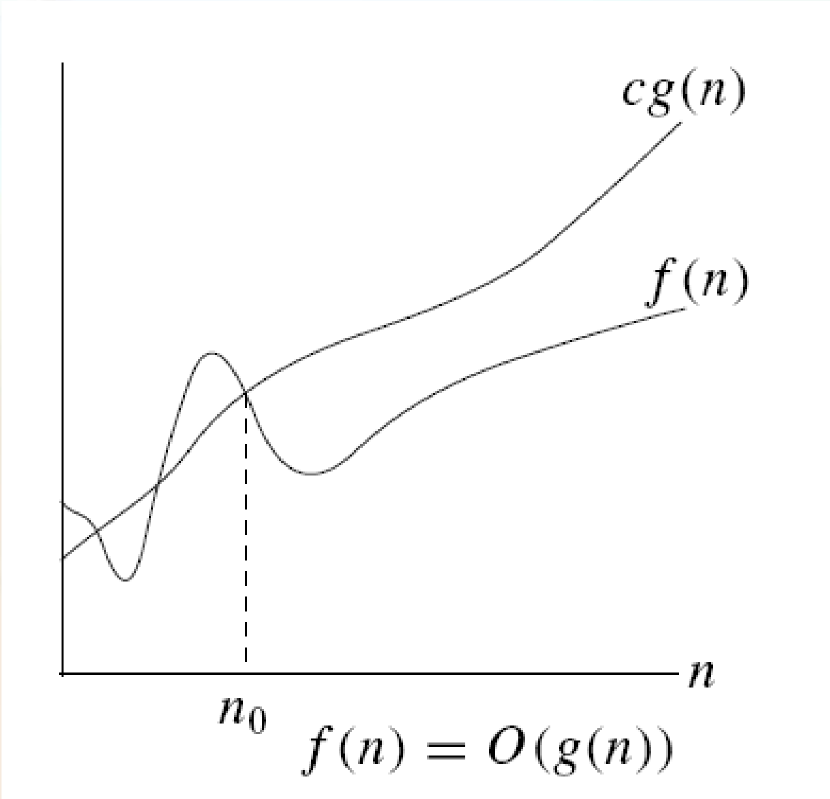
- Asymptotic lower bound Ω
  - $\Omega(g(n)) = \{f(n)|$ there are positive constants c and n_0 such that $0 \leq c \cdot g(n) \leq f(n)$ for all $n \geq n_0\}$



From: Cormen, T., C. Leiserson, R. Rivest, and C. Stein, *Introduction to Algorithms*, MIT Press, 2001.

Asymptotic notations

- Asymptotic upper bound O
  - $O(g(n)) = \{f(n) |$ there are positive constants c and n_0 such that $0 \leq f(n) \leq c \cdot g(n)$ for all $n \geq n_0\}$



From: Cormen, T., C. Leiserson, R. Rivest, and C. Stein, *Introduction to Algorithms*, MIT Press, 2001.

Algorithm Design: Standard

- Use for easy problems
- Traditional design
  - Greedy algorithms
  - Divide-and-conquer
  - Dynamic programming
  - Search

Greedy Algorithms

- Find the best way to solve the problem at hand
- This solution leads to the best overall solution
- Examples
  - Dijkstra’s algorithm for shortest path with only non-negative edge
  - Fractional knapsack

Divide-and-conquer

- Divide a problem of big instance to independent sub-problems of smaller instance
- Find the solution for each sub-problem
- The solutions for sub-problems are combined as the solution for the whole problem.
- Examples
  - Mergesort

Dynamic programming

- Divide a problem of big instance to sub-problems of smaller instance
- Find the best solution for each sub-problem
- Find the best way to combine different combinations of sub-problems to solve the whole problem.
- Examples
  - Floyd's algorithm for shortest path
  - 0/1 knapsack

Search

- View a solution to a problem as a sequence of decisions
- Create a search tree in which
- Each node in the tree is a state of the problem
- each level of the tree is one decision
- Each branch of the tree is one possible decision
- Traverse the tree
- Examples
  - Shortest path

Algorithm Design: Advanced

- Probabilistic/randomized algorithms
- Approximation algorithms
- **Use for hard problems**

Probabilistic/randomized Algorithms

- Use randomness in solving the problem
- Behavior of the algorithm is random
- Need average-case analysis
- Another set of definitions for complexity classes
- Examples:
  - Quicksort
  - Min cut

Approximation Algorithms

- Give near-optimal solution
- *r*-approximation algorithm
  - The solution is within *r* of the optimal solution
- Examples:
  - Vertex cover
  - 0/1 Knapsack