Discrete Mathematics: Lecture 10

- Today:
  - Chap 3.1: Algorithms
  - Chap 3.2: The Growth of functions
- Assignment 3 due in two weeks
- Next time:
  - Chap 3.3: Complexity of algorithms
  - Chap 4.1: Divisibility and modular arithmetic

Review of last time

- Useful summation formula
- Cardinality of sets
- Countable sets: examples and counterexamples
- Schröder-Bernstein Theorem
- $\bigcup_{n=1}^{\infty} (\mathbf{Z}^+)^n$ is countable
- There are uncomputable functions
- Matrices, matrix addition, product, power of matrices

Chap 3.1 Algorithms (算法)

- Many problems arise in discrete mathematics, *e.g.*
  - find the maximum value in a finite sequence of integers
  - find the shortest route between two cities
- Definition: An algorithm is a finite sequence of instructions for solving a problem.
- Describing an algorithm
  - Use English
  - Use a computer language: the description would be complicated and difficult to understand
  - We use pseudocode: the instructions resemble those used in programming languages, and can include any well-defined operations

An Example

Give an algorithm for finding the maximum value in a finite sequence of integers.

procedure $max(a_1, a_2, \ldots, a_n$:integers)
$max := a_1$
for i := 2 to n
    if $max < a_i$ then $max := a_i$
$\{max$ is the largest element$\}$

Properties of Algorithms

- Input. Input values in a specified set
- Output. Output values in a specified set
- Definiteness. The steps must be defined precisely
- Correctness. Should produce the correct output value
- Finiteness. Should stop after a finite number of steps
- Effectiveness. It is possible to perform each step exactly and in a finite amount of time
- Generality. Should be applicable to all input values of the problem

Searching Algorithms (查找算法)

Problem: Given a pile of assignments, find your own assignment

() Locating an element x in an ordered list of distinct elements $a_1, a_2, \ldots, a_n$, or determine that it is not in the list

The linear search (线性查找) algorithm

procedure linear search(x : integer, $a_1, a_2, \ldots, a_n$: distinct integers)
$i := 1$
while ($i \leq n$ and $x \neq a_i$)
    $i := i + 1$
if $i \leq n$ then $location := i$
else $location := 0$

Binary Search

Problem: If the pile of assignments is sorted according to increasing order of student number, can you find your assignments more efficiently?

Example: To search for 19 in the list
1 2 3 5 6 7 8 10 12 13 15 16 18 19 20 22

The binary search (二分查找) algorithm

procedure binary search(x : integer, $a_1, a_2, \ldots, a_n$: increasing integers)
$i := 1$
$j := n$
while $(i < j)$
    $m := \lfloor (i + j)/2 \rfloor$
    if $x > a_m$ then $i := m + 1$
    else $j := m$
if $x = a_i$ then $location := i$
else $location := 0$

Sorting Algorithms (排序算法)

Problem: Sort a pile of assignments according to increasing order of student number

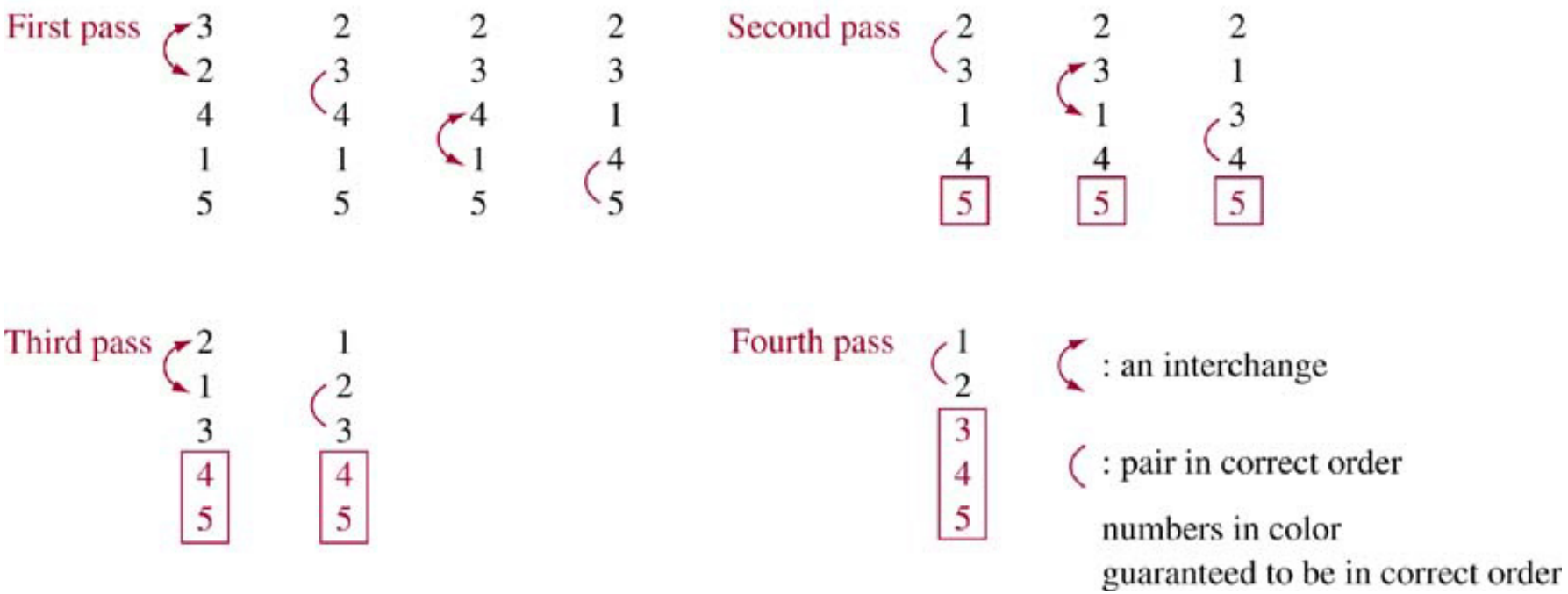
Insertion sort (插入排序): consider elements one by one, when considering the jth element, insert it into the correct position of the previously sorted $j-1$ elements

Example: Sort 3,2,4,1,5

procedure insertion sort($a_1, a_2, \ldots, a_n$: real numbers with $n \geq 2$)
for $j := 2$ to n
    $i := 1$
    while $a_j > a_i$
        $i := i + 1$
    $m := a_j$
    for $k := i$ to $j - 1$
        $a_{k+1} := a_k$
    $a_i := m$

Bubble sort (冒泡排序)

compare adjacent elements, interchange them if they are in the wrong order



procedure bubble sort($a_1, a_2, \ldots, a_n$: real numbers with $n \geq 2$)
for $i := 1$ to $n - 1$
    for $j := 1$ to $n - i$
        if $a_j > a_{j+1}$ then interchange a_j and a_{j+1}

Greedy algorithms (贪心算法) (optional)

- Many algorithms are designed to solve optimization problems: find a solution that minimizes or maximizes some parameter
- Example: find a route between two cities with smallest total mileage
- Greedy algorithms: make the best choice at each step
- Either prove the solution is optimal or give a counterexample where the solution is not optimal
- Example: make n cents change with quarters, dimes, nickels, and pennies, and use the least total number of coins

Making change (2)

At each step, choose the largest denomination $\leq$ the remaining value. Example: 67

procedure change($c_1, c_2, \ldots, c_r$: values of denominations of coins, where $c_1 > c_2 > \ldots > c_r$; n: a positive integer)
for $i := 1$ to r
    while $n \geq c_i$
        add a coin of value c_i to the change
        $n := n - c_i$

We will prove that the above algorithm produces the optimal solution when we have quarters, dimes, nickels, and pennies.

However, it is not the case if we have only quarters, dimes, and pennies. Consider $n = 30$.

Making change (3)

Lemma

If n is a positive integer, then n cents in change using quarters, dimes, nickels, and pennies using the fewest coins possible has at most two dimes, at most one nickel, at most four pennies, and cannot have two dimes and a nickel. The amount of change in dimes, nickels, and pennies cannot exceed 24 cents.

Theorem: When the coins are quarters, dimes, nickels, and pennies, the greedy algorithm produces change using the fewest coins possible.

The halting problem (停机问题)(optional)

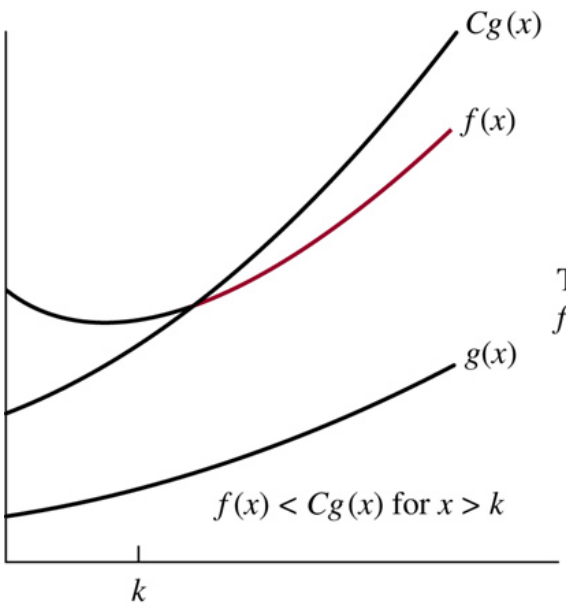
- Question: Can any problem solved by an algorithm?
- Answer: No
- The halting problem: decide if a program will terminate on an input
- One of the most famous theorems in computer science
- Theorem: The halting problem is unsolvable.
  Proof:
  Assume that the problem is solvable by algorithm $H(P, I)$.
  Write a program $K(P)$: it loops forever iff $H(P, P)$ returns "halts".
  Then K loops forever on K iff $H(K, K)$ returns "halts" iff K terminates on K.

Chap 3.2 The growth of functions

- We feel that binary search is more efficient than linear search
- But how to formally compare the efficiency of two algorithms?
- In Chap 3.3, we will estimate the number of comparisons used by the two algorithms
- Now we study the big-O notation, which we use to compare the growth of two functions as the size of inputs grows

Big-O notation

- Definition: Let f and g be functions from numbers to numbers. We say that $f(x)$ is $O(g(x))$ if there exist constants C and k, called witnesses, such that for all $x > k$, $|f(x)| \leq C|g(x)|$.
- Notation: When $f(x)$ is $O(g(x))$, we write $f(x) = O(g(x))$ or $f(x) \in O(g(x))$



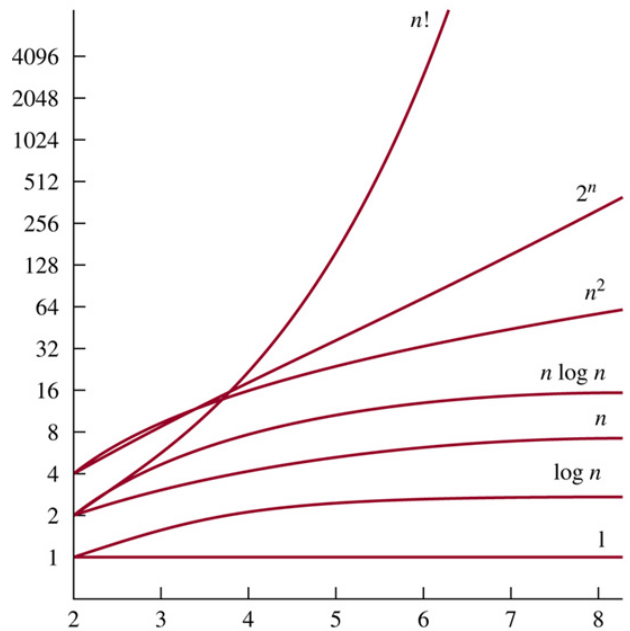
The part of the graph of $f(x)$ that satisfies $f(x) < Cg(x)$ is shown in color.

Examples

- The constants C and k are called witnesses to the relationship $f(x)$ is $O(g(x))$. Note that when there is a pair of witnesses, there are infinitely many pairs of witnesses
- A useful approach for finding a pair of witnesses: first select a value of k for which the size of $|f(x)|$ can be estimated when $x > k$ and see if we can use this estimate to find a value of C
- $f(x) = x^2 + 2x + 1$ is $O(x^2)$
- $7x^2$ is $O(x^3)$
- n^2 is not $O(n)$

Some important big-O results

- Theorem: Let $f(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0$, where $a_0, \ldots, a_n$ are real numbers. Then $f(x)$ is $O(x^n)$.
- Example: Give big-O estimates for $n!$ and $\log n!$
- Example: show that $n = O(2^n)$, $\log n = O(n)$, and $\log_b n = O(n)$,
- 1, $\log n$, n, $n \log n$, n^2, 2^n, $n!$



The growth of combinations of functions

- Theorem: Suppose $f_1(x) = O(g_1(x))$ and $f_2(x) = O(g_2(x))$. Then $(f_1 + f_2)(x) = O(\max(|g_1(x)|, |g_2(x)|))$
- Corollary: Suppose $f_1(x) = O(g(x))$ and $f_2(x) = O(g(x))$. Then $(f_1 + f_2)(x) = O(g(x))$
- Theorem: Suppose $f_1(x) = O(g_1(x))$ and $f_2(x) = O(g_2(x))$. Then $(f_1 f_2)(x) = O((g_1 g_2)(x))$
- Example: $f(n) = 3n \log n! + (n^2 + 3) \log n$
- Example: $f(x) = (x + 1) \log(x^2 + 1) + 3x^2$

Big-Omega notation

- Definition: Let f and g be functions from numbers to numbers. We say that $f(x)$ is $\Omega(g(x))$ if there exist positive constants C and k such that for all $x > k$, $|f(x)| \geq C|g(x)|$.
- Proposition: $f(x) = \Omega(g(x))$ iff $g(x) = O(f(x))$
- Example: $f(x) = 8x^3 + 5x^2 + 7$ is $\Omega(x^3)$

Big-Theta notation

- Definition: Let f and g be functions from numbers to numbers. We say that $f(x)$ is $\Theta(g(x))$, or $f(x)$ is of order $g(x)$, if $f(x) = O(g(x))$ and $f(x) = \Omega(g(x))$.
- Proposition: $f(x) = \Theta(g(x))$ iff $g(x) = \Theta(f(x))$
- Example: $3x^2 + 8x \log x$ is $\Theta(x^2)$
- Theorem: Let $f(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x_1 + x_0$, where $a_0, \ldots, a_n$ are real numbers with $a_n \neq 0$. Then $f(x)$ is $\Theta(x^n)$.