these UofA CS204 lecture notes

- for Fall 2001 (A1/EA1) and Winter 2002
- © R. Hayward, 2001/02 course co-ordinator
- follow CLRS, based on MIT notes for 6.046J (winter 2001)
- http://theory.lcs.mit.edu/classes/6.046/

Lecture 1: Wed Sept 05, 2001

today

- welcome to CS204 ☺
- official course info
- course overview
- announcements
- intro

official course information

- course page http://www.cs.ualberta.ca/ hayward/204/CS204.html
- grade: 21% asn. (best 7 of 8); 40% midterms; 39% final
- deferred final Jan 4, 2002, 1pm
- seminars start week of Sept 10-14 2002
- lectures follow: Intro. to Alg'ms, Cormen/Leiserson/Rivest/Stein
- assignments
  - group size up to three (name/id on page 1)
  - cite *all* external info sources (else plagiarism)
  - CS plagiarism 00/01: 21 cases 1F (plus suspension) ☹
  - due in class
    - late: until noon the next day under my door, 20% off
- prof Hayward (sec A1/EA1)
  - Ath 301 492-3895 hayward@cs.ualberta.ca
  - office hours MTuW 3:30-4:30 or by appt.

course(s) overview

- 204 Alg'ms I
  - introduction to algorithms
  - algorithms: sort/search, matrices, graphs, sets
  - design: divide/conquer, greedy, dyn. programming
  - analysis: model assumptions, worst/avg./best case, aymptotic, reduction, complexity classes P/NP, hard problems
  - calendar/outline with reading list on homepage
- 304 Alg'ms II
  - more (advanced) alg'ms/design/analysis
- 474 Formal Languages, Automata, Computability
  - more formal approach to models, complexity, computability
- other related courses
  - occasional topics courses (eg. 419 alg'c graph theory)
  - 366,466 AI, machine learning

announcements

- assignment #1 available now, due Monday Sept. 17
- NSERC and other research scholarships
  - postdoc app'ns (Cdn? grad spr 02? GPA 7.8+? Ath 232 now)
  - summer research app'ns (later)

role of alg'ms in computing [CLRS Ch. 1]

- algorithm
- typical problems
  - genome project
  - internet (routing, search, mining, ...)
  - e-commerce
  - operations research (resource management, scheduling, ...)
- hard problems
- alg'ms as a technology: time/space management

basic concepts

- problem
- algorithm
- issues for a given algorithm
  - correctness
  - analysis: measuring resource requirements
    - running time
    - space
  - optimality
- algorithm design concepts
  - divide and conquer
  - data structure
  - dynamic programming
  - exhaustive enumeration
  - greedy
  - reduction
  - others (probably won't cover)
    - backtracking
    - branch and bound

getting started [CLRS Ch. 2]

- **algorithm?** a well-defined computational procedure (namely, a sequence of elementary computational steps) which takes an input value and produces an output value, according to a specified mathematical function.
- describing alg'ms: pseudocode
- pseudocode example

```
input:  integers a,b
output: a*b

sum <- 0
for j <- 1 to b do
  sum <- sum + a
return sum
```

- pseudocode conventions [CLRS p19]
  - indentation $\leftrightarrow$ block structure
  - while/for/repeat/if/then/else
  - loop counters retain values
  - ** or $\triangleright$ comment
  - variables local (unless stated otherwise)
  - parameters passed by value
  - boolean "short circuit" evaluation `j>0 AND A[j] < key`

example: two multiplication algorithms

input: positive integers a, b
output: $a \times b$

```
************************************************
Alg'm I: iterated addition

917*45 = 917*45 +     0       sum <- 0
       = 917*44 +   917       for j <- 1 to b do
         ...        ...         sum <- sum + a
       = 917*1  + 40348       return sum
       = 917*0  + 41265
       =          41256
************************************************
Alg'm II: "school" method

       9 1 7
         4 5
   ---------                  pseudocode?
     4 5 8 5
   3 6 6 8
   ---------
   4 1 2 6 5
************************************************
```

which is better? how to compare? analysis
are there even better algorithms? how to find them? design

problem: sorting

input: sequence of n numbers $(a_1, a_2, \ldots, a_n)$

output: permutation $(a'_1, a'_2, \ldots, a'_n)$ such that $a'_1 \leq a'_2 \leq \ldots \leq a'_n$

one sorting algorithm: insertion sort

- idea   repeatedly insert $A[j]$ into sorted sublist $A[1 \ \ldots \ j-1]$
- insert how?
  - search for insert loc'n in sequence $A[j-1]$ $A[j-2]$ $\ldots$
  - move contents to right during search

```
              ?   ?   ?
 -------------------------------------------------------
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |
 -------------------------------------------------------
  1                       j                            n
```

```
InsertionSort(A)   **sorts A[1..n] in place  CLRS page 17
 for j <- 2 to length[A]
   do key <- A[j]
   ** insert A[j] into sorted sequence A[1..j-1]
   i <- j-1
   while i>0 and A[i]>key
     do A[i+1] <- A[i]
        i <- i-1
   A[i+1] <- key
```

insertion sort trace

53	21	47	62	14	38
21	53	47	62	14	38
21	47	53	62	14	38
21	47	53	62	14	38
14	21	47	53	62	38
14	21	38	47	53	62

running time

- how to measure it? implement and run (?)
- problem: run times may vary (language; machine; code; environment)
- problem: not always feasible
- idea
  - select theoretical computer model
  - estimate running time on model (number of cycles)

our model of computation: RAM

- RAM: random access machine [CLRS p21-22]
- components
  - IT: input tape (read-only)
  - OT: ouput tape (write-only)
  - CU: computation unit (inc. program)
  - M: memory locations (each can store integer)
    M[0] M[1] M[2] ...
  - program: fixed user-defined instr. sequence
- properties
  - CU instructions (each usually via register/accumulator)
    - move data between memory
    - compare data and branch
    - binary arithmetic op'ns
    - read from IT to memory
    - write from memory to OT
- e.g. RAM instructions for $z \leftarrow x + y$?

```
M[0] := M[@x]  (*wherever x is*)
M[1] := M[@y]  (*wherever y is*)
add            (*M[0] := M[0] + M[1]*)
M[@z] := M[0]  (*wherever z is gets sum*)
```