Lecture 4: Monday Jan 13, 2003

today

- insertion sort correctness (conclusion)
- mergesort
  - quick review
  - recursion tree
- asymptotic notation [CLRS Ch. 3]

announcements

- attend your section! lectures/exams/seminars vary
- seminars start this week
- quiz 1 starts next Monday

```
InsertionSort(A)
1  for j <- 2 to length[A] do
2    key <- A[j]
3    ** insert A[j] into sorted sequence A[1..j-1]
4    i <- j-1
5    while i>0 and A[i]>key do
6      A[i+1] <- A[i]
7      i <- i-1
8    A[i+1] <- key
```

recall: outer loop invariant

- at start of line 1, keys initially in `A[1...j-1]` are in `A[1...j-1]` and sorted

proving correctness with this invariant

- initialization — done last time
- termination — done last time
- maintenance — how to do this?
- informally: body of outer loop works by moving A[j-1] A[j-2] ... one position to the right, until the proper position for A[j] is found
- more formally? — inner loop invariant 

sketch of more formal proof of maintenance condition

- assume LI holds when $j = k$ so A[1] $\leq$ A[2] $\leq \ldots \leq$ A[k-1]
- must show LI holds when $j = k + 1$ (after next loop body execution)
- loop body contains another loop: use another LI!
- how do we find a useful inner loop invariant?
- answer: trace the inner loop, look for patterns

```
                    i
  1   2   3   4   5   key  x   x   x   x   x

                i
  1   2   3   4   5   5    x   x   x   x   x

            i
  1   2   3   4   4   5    x   x   x   x   x

        i
  1   2   3   3   4   5    x   x   x   x   x

    i
  1   2   2   3   4   5    x   x   x   x   x

i
  1   1   2   3   4   5    x   x   x   x   x
```

useful inner loop invariant (LI2)

- let `A*` be the contents of `A` just before inner loop
- at the start of line 5,
  - `A[ 1  ... i ] = A*[ 1  ...  i ]`
  - `A[i+2 ... j ] = A*[i+1 ... j-1]`
  - key $\leq$ `A[i+1]`

correctness of insertion sort (conclusion)

- for LI2: initialization? maintenance? termination?
- using LI2, prove LI1
- hint: when LI2 terminates, i=0 or A[i] $\leq$ key

conclusion

- proving correctness is not trivial

```
merge(A,lo,m,hi)  * CLRS p29
  * precondition: lo <= m < hi and
  *   A[lo..m] and A[m+1..hi] sorted
  * postcondition: A[lo..hi] sorted

mergesort(A,lo,hi)  * CLRS p 32
1 if lo<hi then
2   mid <- floor((lo+hi)/2)
3   mergesort(A, lo,    mid)
4   mergesort(A, mid+1, hi )
5   merge(A,lo,mid,hi)
```

- analysis: later (see text)
  - recursion: a programming technique (not really alg'm design)
  - recursion trees
  - recurrence relations

```
                  1  2  3  4  5  6  7  8  9 10 11 12 13
             A [ 13 01 11 07 06 01 04 10 12 09 08 15 03]
recursion tree
x,y <-> mergesort(A,x,y)        1,13

                 1,7                            8,13

        1,4              5,7              8,10             11,13

  1,2        3,4      5,6    7,7     8,9   10,10   11,12 13,13

1,1 2,2   3,3 4,4   5,5 6,6        8,8 9,9      11,11 12,12


--   --   --   --   --   --        --   --        --   --
13   01   11   07   06   01        10   12        08   15

-------   -------   ------    --   -------   --   -------   --
01   13   07   11   01   06   04   10   12   09   08   15   03

-----------------   ------------   ------------   ------------
01   07   11   13   01   04   06   09   10   12   03   08   15

--------------------------------   ---------------------------
01   01   04   06   07   11   13   03   08   09   10   12   15

--------------------------------------------------------------
01   01   03   04   06   07   08   09   10   11   12   13   15
```

recall: analysis issues

- issues
  - correctness
  - resources used (time/space)
  - simplicity
  - optimality
- estimating resources used
  - various input sizes $\Rightarrow t(n)$ $s(n)$
  - worst/average/best cases (WC/AC/BC)
  - machine independence $\Rightarrow$ model
- model of computation
  - simple (architecture/instruction set)
  - reflective ('typical machine'; accurate estimates)
  - our choice: RAM (random access machine)
  - two versions: uniform cost, log cost
  - choose version depending on application
  - astronomical numbers: log cost
  - reasonable size numbers: uniform cost

when presenting analysis

for algorithm run times, remember to state what you have counted (e.g. RAM instructions? log RAM instructions? data moves? data comparisons? arithmetic operations? Pentium II clock cycles? etc.)

our story so far ...

- analysis of algorithms $\Rightarrow$ analysis of functions
- to simplify alg'm analysis, want function notation which indicates 'rate of growth' (a.k.a. 'order' of complexity)

O(f(n)) "big O of f(n)"

roughly: the set of functions which, as n gets large,
grow no faster than a constant times $f(n)$

precisely: the set of functions $\{h(n) : N \mapsto R\}$ s.t. for each $h(n)$,
there are $c_0 \in R^+$ and $n_0 \in N$ s.t. $h(n) \leq cf(n)$ for all $n \geq n_0$

$h(n) = 3n^2 + 10n + 1000 \lg n \in O(n^2)$**?**

$h(n) = 3n^2 + 10n + 1000 \lg n \in O(n^3)$**?**

$$h(n) = \begin{cases} 5^n & n \leq 10^{100} \\ n^2 & n > 10^{100} \end{cases} \in O(n^2)\textbf{?}$$