Lecture 15: Friday February 7, 2003

today

- quicksort [CLRS Ch. 7]
  - analysis: AC
  - improvements
    - splitter: sampling, randomization
    - small sublists
    - stack size: tail recursion
  - in practice?
    - unix `qsort`
    - strong concentration

recall quicksort

```
QS(A,lo,hi)                     (*sort A[lo ... hi]
if lo < hi then
  spI <- partition(A,lo,hi)  (*split index
  QS(A, lo ,spI-1)
  QS(A,spI+1, hi )
```

```
partition(A,lo,hi)          (*upon completion,
         A[lo ... lastSmallI]    smaller  keys
         A[1+lastSmallI]           splitter
         A[2+lastSmallI ... hi]  larger   keys
 return                    lastSmallI+1
```

```
 * * * * * * * * *                  ? ? ? ? ?  ^
--------------------------------------------------
|    <= spK        |    > spK       |          | spK
--------------------------------------------------
 lo    lastSmallI                    next       hi
```

recall: a QS recursion tree

- QS: run time proportional to number key comparisons
- to count KC, only need know (at each call) rank of split key

```
(x,y)  <->  QS(A,x,y), splitter ends up at A[z]
  z
***************************************************************
                  (1,16)
                    6


          (1,5)                   (7,16)
            4                       9


  (1,3)       (5,5)   (7,8)                       (10,16)
    2           5       7                           16

(1,1) (3,3)               (8,8)              (10,15)
  *     *                   *                  13

                                   (10,12)          (14,15)
                                     10               14

                                        (11,12)         (15,15)
                                           12              *

                                    (11,11)
                                       *
```

recall QS run time

- worst case: $n(n-1)/2$ key comparisons
- best case: $\approx n \lg n$ key comparisons
- average case?

QS run time: average case

- when you see ‘average’, ask: **average over what?**
- ask: over what **distribution** (set of values) is average computed?
- if no other info’, assume equiprobable dist’n
- equiprobable dist’n: all inputs equally likely
- also known as uniform dist’n
- $Q(n) = q_n$: average number of KC in QS of n keys, assuming equiprobable input dist’n (each of $n!$ input permutations is equally likely)

determining q_n

- $q_0 = 0$ why?
- $q_1 = 0$ why?
- $q_2 = 1$ 2! inputs: $\frac{1+1}{2}$
- $q_3 = \frac{8}{3}$ 3! inputs: $2 + \frac{1+0+1+1+0+1}{6}$
- $q_4 = ?$
- $q_n = n - 1 + \frac{1}{n} \sum_{j=1}^{n} (q_{j-1} + q_{n-j})$
- why?
- for each input, $n-1$ KC to partition
- each input equiprobable $\Rightarrow$ each splitter equiprobable
- if the splitter has rank j
  - the sublist sizes are $j-1$ and $n-j$
  - each sublist of $j-1$ smaller keys equiprobable (lucky!)
  - each sublist of $n-j$ larger keys equiprobable (lucky!)
  - to recursively sort smaller sublist: q_{j-1} KC
  - to recursively sort smaller sublist: q_{n-j} KC

determining q_n (cont'd)

$$q_n = n - 1 + \frac{1}{n}\sum_{j=1}^{n}(q_{j-1} + q_{n-j})$$

$$= n - 1 + \frac{2}{n}\sum_{j=1}^{n} q_{j-1}$$

$$nq_n - (n-1)q_{n-1} = 2q_{n-1} + 2(n-1)$$

$$nq_n = (n+1)q_{n-1} + 2(n-1)$$

$$\frac{q_n}{n+1} = \frac{q_{n-1}}{n} + \frac{2(n-1)}{n(n+1)} = 2\sum_{j=1}^{n}\frac{j-1}{j(j+1)} = 2\sum_{j=1}^{n}\frac{1}{j} - 4\sum_{j=1}^{n}\frac{j}{j+1}$$

$$q_n = 2(n+1)H(n) - 4n = 2(n+1)(\ln n + \gamma) - 4n$$

$$= 2n\ln n - (4-2\gamma)n + 2\ln n + 2\gamma$$

$$\in 2n\ln n - o(n)$$

$$\in \Theta(n\log n) \qquad \gamma = 0.577\ldots \qquad H(n) = \sum_{j=1}^{n} 1/j$$

- QS average case run time in $\Theta(n\log n)$

QS space requirements

- not an in place sorting algorithm, because extra space required for all subproblems on the stack
- worst case: can have $\Theta(n)$ subproblems on stack

QS improvements

- small sublists: use insertion sort
  - can determine best crossover size (about 20)
- split selection
  - sampling
    * use A[mid] instead of A[hi] mid = (lo+hi)/2
    * better: median of { A[lo], A[mid], A[hi] }
  - randomization
    * select split using pseudo-random number generator
- space! stack size
  - remove tail recursion
  - make sure smaller subproblem executed first
  - only need $\Theta(\log n)$ extra space