Class 13: Quicksort, Problems and Procedures

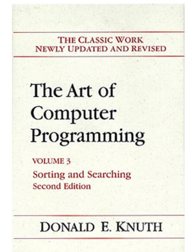
Queen's University, Belfast, Northern Ireland

CS150: Computer Science
University of Virginia
Computer Science

David Evans
http://www.cs.virginia.edu/evans

Why are we spending so much time on sorting?

- Reason 1: its important
- **Reason 2:** it is a well defined problem for exploring algorithm design and complexity analysis



800 pages long!

A sensible programmer rarely (if ever) writes their own code for sorting – there are sort procedures provided by all major languages

Art of Computer Programming, Donald E. Knuth

- Volume 1 (1968): Fundamental Algorithms
- Volume 2: Seminumerical Algorithms
  - Random numbers, arithmetic
- **Volume 3: Sorting and Searching**
- Volume 4: Combinatorial Algorithms (in preparation, 2005)
- Volume 5: Syntactic Algorithms (estimated for 2010)
- Volume 6, 7: planned

The first finder of any error in my books receives $2.56; significant suggestions are also worth $0.32 each. If you are really a careful reader, you may be able to recoup more than the cost of the books this way.

Recap: insertsort-tree

```
(define (insertel-tree cf el tree)
  (if (null? tree)
      (make-tree null el null)
      (if (cf el (get-element tree))
          (make-tree (insertel-tree cf el (get-left tree))
                     (get-element tree)
                     (get-right tree))
          (make-tree (get-left tree)
                     (get-element tree)
                     (insertel-tree cf el (get-right tree))))))
```

n = number of elements in tree

$\theta(\log n)$

```
(define (insertsort-tree cf lst)
   (define (insertsort-worker cf lst)
      (if (null? lst) null
          (insertel-tree cf (car lst)
                    (insertsort-worker cf (cdr lst)))))
   (extract-elements (insertsort-worker cf lst)))
```

n = number of elements in lst

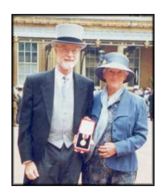
$\theta(n \log n)$

Can we do better?

- Making all those trees is a lot of work
- Can we divide the problem in two halves, without making trees?

Quicksort

- Sir C. A. R. (Tony) Hoare, 1962
- Divide the problem into:
  - Sorting all elements in the list where
    (cf (car list) el)
    is true (it is < the first element)
  - Sorting all other elements
    (it is >= the first element)
- Will this do better?



Quicksort

```
(define (quicksort cf lst)
  (if (null? lst) lst
      (append
       (quicksort cf
          (filter (lambda (el) (cf el (car lst)))
                   (cdr lst)))
       (list (car lst))
       (quicksort cf
           (filter (lambda (el) (not (cf el (car lst))))
                    (cdr lst))))))
```


How much work is quicksort?

```
(define (quicksort cf lst)
  (if (null? lst) lst
      (append
       (quicksort cf
          (filter (lambda (el) (cf el (car lst)))
                   (cdr lst)))
       (list (car lst))
       (quicksort cf
           (filter (lambda (el) (not (cf el (car lst))))
                    (cdr lst))))))
```

What if the input list is sorted?

Worst Case: $\Theta(n^2)$

What if the input list is random?

Expected: $\Theta(n \log_2 n)$

Comparing sorts

```
> (testgrowth insertsort-tree)
n = 250, time = 20
n = 500, time = 80
n = 1000, time = 151
n = 2000, time = 470
n = 4000, time = 882
n = 8000, time = 1872
n = 16000, time = 9654
n = 32000, time = 31896
n = 64000, time = 63562
n = 128000, time = 165261
(4.0 1.9 3.1 1.9 2.1 5.2 3.3 2.0 2.6)
```

```
> (testgrowth quicksort)
n = 250, time = 20
n = 500, time = 80
n = 1000, time = 91
n = 2000, time = 170
n = 4000, time = 461
n = 8000, time = 941
n = 16000, time = 2153
n = 32000, time = 5047
n = 64000, time = 16634
n = 128000, time = 35813
(4.0 1.1 1.8 2.7 2.0 2.3 2.3 3.3 2.2)
```

Both are $\Theta(n \log_2 n)$

Absolute time of quicksort much faster

Good enough for VISA?

n = 128000, time = 35813

36 seconds to sort 128000 with quicksort

$\Theta(n \log_2 n)$

How long to sort 800M items?

```
> (log 4)
1.3862943611198906
> (* 128000 (log 128000))
1505252.5494914246
> (/ (* 128000 (log 128000)) 36)
41812.57081920624
> (/ (* 128000 (log 128000)) 41812.6)
35.99997487578923
> (/ (* 800000000 (log 800000000)) 41812.6)
392228.6064130373
```

392000 seconds ~ 4.5 days

Are there any procedures more complex than simulating the universe ($\Theta(n^3)$) ?

Permuted Sorting

- A (possibly) really dumb way to sort:
  - Find all possible orderings of the list (permutations)
  - Check each permutation in order, until you find one that is sorted
- Example: sort (3 1 2)

  All permutations:

  (3 1 2) (3 2 1) (2 1 3) (2 3 1) (1 3 2) (1 2 3)

  is-sorted? is-sorted? is-sorted? is-sorted? is-sorted? is-sorted?

permute-sort

```
(define (permute-sort cf lst)
  (car
   (filter (lambda (lst) (is-sorted? cf lst))
           (all-permutations lst))))
```


is-sorted?

```
(define (is-sorted? cf lst)
  (or (null? lst) (= 1 (length lst))
      (and (cf (car lst) (cadr lst))
           (is-sorted? cf (cdr lst)))))
```


all-permutations

```
(define (all-permutations lst)
  (flat-one
   (map
    (lambda (n)
      (if (= (length lst) 1)
          (list lst) ; The permutations of (a) are ((a))
          (map
           (lambda (oneperm)
             (cons (nth lst n) oneperm))
           (all-permutations (exceptnth lst n)))))
    (intsto (length lst)))))
```



```
> (time (permute-sort <= (rand-int-list 5)))
cpu time: 10 real time: 10 gc time: 0
(4 14 14 45 51)
> (time (permute-sort <= (rand-int-list 6)))
cpu time: 40 real time: 40 gc time: 0
(6 29 39 40 54 69)
> (time (permute-sort <= (rand-int-list 7)))
cpu time: 261 real time: 260 gc time: 0
(6 7 35 47 79 82 84)
> (time (permute-sort <= (rand-int-list 8)))
cpu time: 3585 real time: 3586 gc time: 0
(4 10 40 50 50 58 69 84)
> (time (permute-sort <= (rand-int-list 9)))
```

Crashes!

How much work is permute-sort?

```
(define (permute-sort cf lst)
  (car
   (filter (lambda (lst) (is-sorted? cf lst))
           (all-permutations lst))))
```

- We evaluated is-sorted? once for each permutation of lst.
- How much work is is-sorted??

  $\Theta(n)$
- How many permutations of the list are there?

Number of permutations

```
(map
  (lambda (n)
    (if (= (length lst) 1) (list lst)
        (map (lambda (oneperm) (cons (nth lst n) oneperm))
             (all-permutations (exceptnth lst n)))))
 (intsto (length lst)))
```

- There are n = (length lst) values in the first map, for each possible first element
- Then, we call all-permutations on the list without that element (length = $n - 1$)
- There are $n * n - 1 * \ldots * 1$ permutations
- Hence, there are $n!$ lists to check: $\Theta(n!)$

Are there any procedures more complex than simulating the universe ($\Theta(n^3)$) ?

Maybe this is the wrong question…

Procedures and Problems

- So far we have been talking about **procedures** (how much work is permute-sort?)
- We can also talk about **problems**: how much work is *sorting*?
- A **problem** defines a desired output for a given input. A **solution** to a problem is a procedure for finding the correct output for all possible inputs.

The Sorting Problem

- Input: a list and a comparison function
- Output: a list such that the elements are the same elements as the input list, but in order so that the comparison function evaluates to true for any adjacent pair of elements

Problems and Procedures

- If we know a *procedure* that is that is $\Theta(f(n))$ that solves a *problem* then we know the problem is $O(f(n))$.
- The sorting **problem** is $O(n!)$ since we know a **procedure** (permute-sort) that solves it in $\Theta(n!)$
- Is the sorting **problem** is $\Theta(n!)$?

  No, we would need to **prove there is no better procedure**.

Problems and Procedures

- Sorting problem is $O(n \log n)$
  - We know a procedure (quicksort) that solves sorting in $\Theta(n \log n)$
- Is the sorting problem $\Theta(n \log n)$?
  - To know this, we need to prove there is no procedure that solves the sorting problem with time complexity better than $\Theta(n \log n)$

Sorting problem is $\Omega(n \log n)$

- There are $n!$ possible orderings
- Each comparison can eliminate at best ½ of them
- So, best possible sorting procedure is $\Omega(\log_2 n!)$
- Sterling's approximation: $n! = \Omega(n^n)$
  - So, best possible sorting procedure is

    $\Omega(\log (n^n)) = \Omega(n \log n)$

Recall log multiplication is normal addition: $\log mn = \log m + \log n$

Problems and Procedures

- Sorting problem is $\Theta(n \log n)$
  - We know a procedure (quicksort) that solves sorting in $\Theta(n \log n)$
  - We know there is no faster procedure since best sorting procedure is $\Omega(n \log n)$
- This is unusual: there are very few problems for which we know Θ
  - It is "easy" to get O for a problem: just find a procedure that solves it
  - It is **extraordinarily difficult** to get Ω for most problems: need to reason about all possible procedures

Charge

- Next class:
  - Some *problems* that are "hard"
    - No procedure is known that is less complex than simulating the universe
  - Introduce the most famous and important open problem in Computer Science
    - Are these really hard problems?
- Will return PS3 Friday
- PS4: Due Monday