
## Menu

- compose and n-times
- Measuring Work:
  What $\theta$ really means
- Quicker Sorting



## What does $\theta$ really mean?

- $O(x)$ – it is **no more than** $x$ work (upper bound)
- $\Theta(x)$ – work scales as $x$ (tight bound)
- $\Omega(x)$ – it is **at least** $x$ work (lower bound)

If $O(x)$ and $\Omega(x)$ are true, then $\Theta(x)$ is true.



## Meaning of $O$ ("big Oh")

$f(x)$ is $O(g(x))$ means:

There is a positive constant $c$ such that

$c * f(x) < g(x)$

for all but a finite number of $x$ values.



## $O$ Examples

$f(x)$ is $O(g(x))$ means:

There is a positive constant $c$ such that

$c * f(x) < g(x)$

for all but a finite number of $x$ values.

$x$ is $O(x^2)$? Yes, $c = 1$ works fine.

$10x$ is $O(x)$? Yes, $c = .09$ works fine.

~~$x^2$ is $O(x)$?~~ No, no matter what $c$ we pick, $cx^2 > x$ for big enough $x$



## Lower Bound: $\Omega$ (Omega)

$f(x)$ is $\Omega(g(x))$ means:

There is a positive constant $c$ such that

$c * f(x) > g(x)$

for all but a finite number of $x$ values.

Difference from $O$ – this was <

## Examples

$f(x)$ is $\Omega(g(x))$ means:
There is a positive constant $c$ such that
$c * f(x) > g(x)$
for all but a finite number of $x$ values.
$f(x)$ is $O(g(x))$ means:
There is a positive constant $c$ such that
$c * f(x) < g(x)$
for all but a finite number of $x$ values.

- $x$ is $\Omega(x)$
  - Yes, pick $c = 2$
- $10x$ is $\Omega(x)$
  - Yes, pick $c = 1$
- Is $x^2$ $\Omega(x)$?
  - Yes!

- $x$ is $O(x)$
  - Yes, pick $c = .5$
- $10x$ is $O(x)$
  - Yes, pick $c = .09$
- $x^2$ is **not** $O(x)$



## Tight Bound: $\theta$ (Theta)

$f(x)$ is $\theta(g(x))$ iff:
$f(x)$ is $O(g(x))$
and $f(x)$ is $\Omega(g(x))$



## $\theta$ Examples

- $10x$ is $\theta(x)$
  - Yes, since $10x$ is $\Omega(x)$ and $10x$ is $O(x)$
    - Doesn't matter that you choose different $c$ values for each part; they are independent
- $x^2$ is/is not $\theta(x)$?
  - No, since $x^2$ is not $O(x)$
- $x$ is/is not $\theta(x^2)$?
  - No, since $x^2$ is not $\Omega(x)$



## Sorting

```
(define (simple-sort cf lst)
  (if (null? lst) lst
      (let ((best (find-best cf lst)))
        (cons
          best
          (simple-sort cf
             (delete lst most))))))
```

```
(define (find-best cf lst)
  (insertl
   (lambda (c1 c2)
     (if (cf c1 c2) c1 c2))
   lst
   (car lst)))
```

simple-sort is $\Theta(n^2)$

If we double the length of the list, we amount of work sort does *approximately* quadruples.



## Is our sort good enough?

Takes over 1 second to sort 1000-length list. How long would it take to sort 1 million items?

1s = time to sort 1000
4s ~ time to sort 2000

$\Theta(n^2)$

1M is 1000 * 1000

Sorting time is $n^2$
so, sorting 1000 times as many items will take $1000^2$ times as long
= 1 million seconds ~ 11 days

Note: there are 800 Million VISA cards in circulation.
It would take 20,000 years to process a VISA transaction at this rate.



## Divide and Conquer sorting?

- simple-sort: find the lowest in the list, add it to the front of the result of sorting the list after deleting the lowest
- Insertion sort: insert the first element of the list in the right place in the sorted rest of the list

## insertsort

```
(define (insertsort cf lst)
  (if (null? lst)
      null
      (insertone cf
          (car lst)
          (insertsort cf (cdr lst)))))
```



## insertone

```
(define (insertone cf el lst)
  (if (null? lst)
      (list el)
      (if (cf el (car lst))
        (cons el lst)
        (cons (car lst)
            (insertone cf el (cdr lst))))))
```



## How much work is insertsort?

```
(define (insertsort cf lst)
  (if (null? lst)
      null
      (insertone cf
          (car lst)
          (insertsort cf
                  (cdr lst)))))
```

```
(define (insertone cf el lst)
  (if (null? lst)
      (list el)
      (if (cf el (car lst))
        (cons el lst)
        (cons (car lst)
            (insertone cf el
                  (cdr lst))))))
```

How many times does insertsort evaluate insertone?

$n$ times (once for each element)

insertsort is $\Theta(n^2)$

Worst case? Average case?

insertone is $\Theta(n)$



> (insertsort < (revintsto 20))
**(1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20)**
Requires 190 applications of <

> (insertsort < (intsto 20))
**(1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20)**
Requires 19 applications of <

> (insertsort < (rand-int-list 20))
**(0 11 16 19 23 26 31 32 32 34 42 45 53 63 64 81 82 84 84 92)**
Requires 104 applications of <



> (simplesort < (intsto 20))
**(1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20)**
Requires 210 applications of <

> (simplesort < (rand-int-list 20))
**(4 4 16 18 19 20 23 32 36 51 53 59 67 69 73 75 82 82 88 89)**
Requires 210 applications of <



## simplesort vs. insertsort

- Both are $\Theta(n^2)$ worst case (reverse list)
- Both are $\Theta(n^2)$ average case (random)
  - But insert-sort is about twice as fast
- insertsort is $\Theta(n)$ best case (ordered list)

## Can we do better?

(insertone < 88
(list 1 2 3 5 6 23 63 77 89 90))

Suppose we had procedures
(first-half lst)
(second-half lst)
that quickly divided the list in two halves?



## Charge

- Read Tyson's essay (before Friday)
  - How does it relate to $\theta(n^2)$
  - How does it relate to grade inflation
  - Don't misinterpret it as telling you to run out and get tattoos and piercings!