Sort Procedures and Quicker Sorting

One-Slide Summary

- **g is in O(f)** iff there exist positive constants c and n_0 such that $g(n) \le cf(n)$ for all $n \ge n_0$.
- If g is in O(f) we say that f is an **upper bound** for g.
- We use **Omega Ω** for lower bounds and **Theta Θ** for tight bounds.
- Knowing a running time is in O(f) tells you that the running time is *not worse than f*. This can only be good news.
- We can add two numbers with electricity.

Outline

- Review Big Oh
- Adding Two Numbers With Electricity
- Sorting: timing and costs
- Insertion Sort

Recall: Asymptotic Complexity

g is in $O(f)$ iff: There are positive constants c and n_0 such that $g(n) \le cf(n)$ for all $n \ge n_0$.

g is in $\Omega(f)$ iff: There are positive constants c and n_0 such that $g(n) \ge cf(n)$ for all $n \ge n_0$.

g is in $\Theta(f)$ iff: g is in $O(f)$ and g is in $\Omega(f)$.

Θ Examples

- Is $10n$ in $\Theta(n)$?
- Is n^2 in $\Theta(n)$?
- Is n in $\Theta(n^2)$?

Θ Examples

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

Example

- Is n in $\Omega(n^2)$?

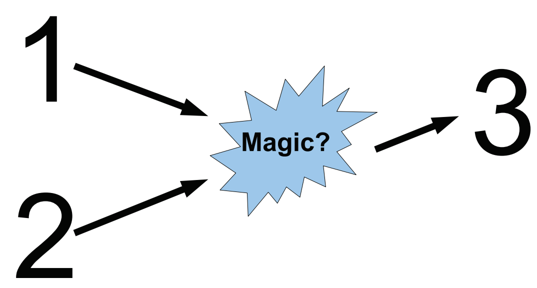
$n \geq cn^2$ for all $n \geq n_0$

$1 \geq cn$ for all $n \geq n_0$

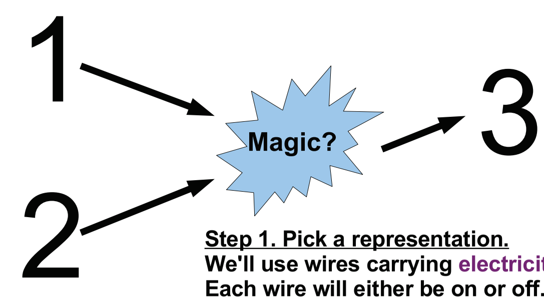
No matter what c is, I can make this false by using $n = (1/c + 1)$

> g is in $\Omega(f)$ iff there are positive constants c and n_0 such that $g(n) \geq cf(n)$ for all $n \geq n_0$.

How To Add Two Numbers With Electricity

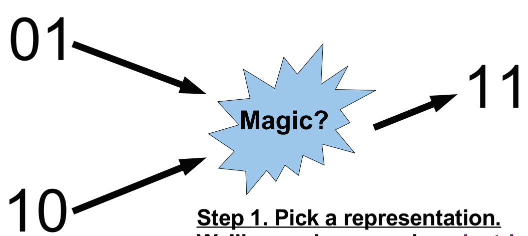


How To Add Two Numbers With Electricity



<u>Step 1. Pick a representation.</u>
We'll use wires carrying electricity.
Each wire will either be on or off.
Each wire will thus encode one bit.
We'll represent our numbers in binary.

How To Add Two Numbers With Electricity



<u>Step 1. Pick a representation.</u>
We'll use wires carrying electricity.
Each wire will either be on or off.
Each wire will thus encode one bit.
We'll represent our numbers in binary.

How To Add Two Numbers With Electricity

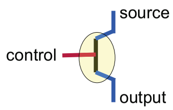
<u>Step 1. Pick a representation.</u>
We'll use wires carrying electricity.
Each wire will either be on or off.
Each wire will thus encode one bit.
We'll represent our numbers in binary.

Still Adding Numbers

- What does it mean for a wire to be "on"?
  - We'll use voltage.
  - Ex: bit 0 is 0V to 0.8V and bit 1 is 2V to 5V
- Great. So how do I combine and manipulate voltages?
  - Example: 0+0 = 0
  - Example: 0+1 = 1
  - Example: 1+0 = 1
  - Somehow I need the output to be "on" if either of the inputs is "on". How do I do it?

The Transistor

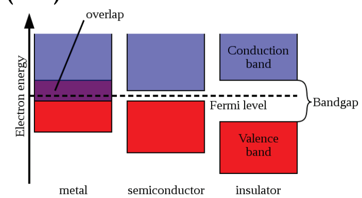
- A **transistor** is a device used to amplify or switch electronic signals.
- A transistor used as a switch has three connections to the outside world:
  - source
  - control
  - output



- If the control is "on", the source flows to the output. Otherwise, the output is "off".
  - A transistor is like a faucet.

The Transistor Continued

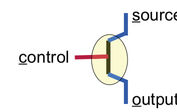
- A **transistor** is made of a solid piece of semiconductor material.
- A **semiconductor** is a material that has electrical conductivity that varies dynamically between that of a conductor (on) or an insulator (off). **Silicon** is a semiconductor.



The Transistor

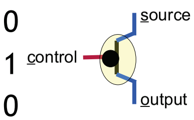
- With transistors it is possible to make two switches: normal control, and inverted control.
  - The black dot means inverted.
- Exhaustive Listing:

S	C	O
1	1	1
1	0	0
0	1	0
0	0	0



What logical operation is this?

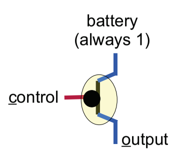
S	C	O
1	1	0
1	0	1
0	1	0
0	0	0



The Notty Transistor

- One Trick: what if we wire the source of an inverted control switch up to a battery that is always on?
- Exhaustive Listing:

C	O
1	0
0	1



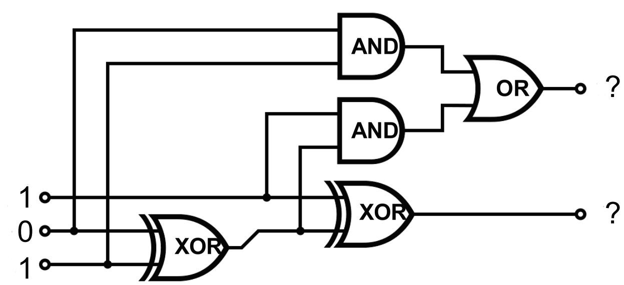
What logical operation is this?

Boolean Logic

- So we have (**and** X Y) and (**not** X) for bits.
- Also (**or** X Y) = (not (and (not X) (not Y)))
- Also (**xor** X Y) = (and (or x y) (not (and x y)))
- An electronic circuit that operates on bits and implements basic boolean logic is called a **gate**.
- So far we have and, or, xor and not gates.
- That's all we need to add numbers!

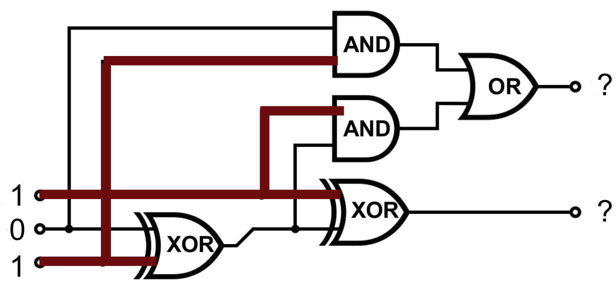
Adding Numbers!

- 1 + 0 + 1 = 10



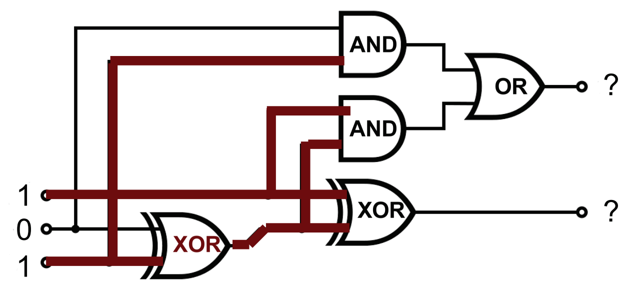
Adding Numbers!

- 1 + 0 + 1 = 10



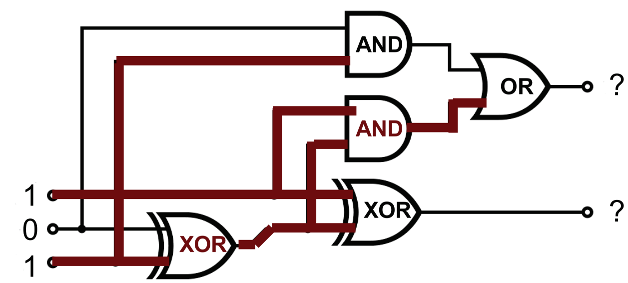
Adding Numbers!

- 1 + 0 + 1 = 10



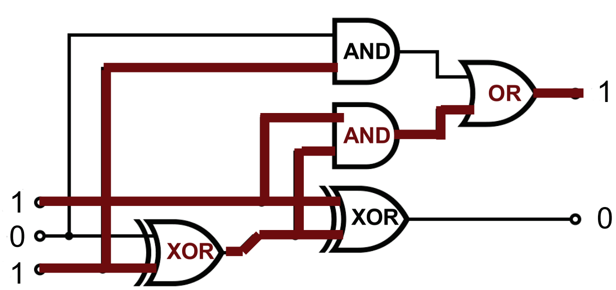
Adding Numbers!

- 1 + 0 + 1 = 10



Adding Numbers!

- 1 + 0 + 1 = 10



Electronic Computers

- By using *semiconductors*
  - which work using physical properties of silicon
- We can build *transistors*
  - which are like switches or faucets
- To manipulate electrical *voltages*
  - which represent bits
- Through logical *gates*
  - which encode and, or, not, etc.
- To add (and subtract, etc.) numbers!
  - In O(1) time. This is the **basis** of our cost model.

Liberal Arts Trivia: Dance

- This four wall line dance was created in 1976 by American dancer Ric Silver. It was popularized by Marcia Griffiths and remains a perennial wedding favorite. Steps: 1-4 grapevine right (tap and clap on 4), 5-8 grapevine left (tap and clap on 8), 9-12 walk back (tap and clap on 12), etc. The lyrics include “I'll teach you the …”

Liberal Arts Trivia: Medieval Studies

- This son of Pippin the Short was King of the Franks from 768 to his death and is known as the "father of Europe": his empire united most of Western Europe for the first time since the Romans. His rule is associated with the Carolingian Renaissance, a revival of art, religion and culture. The word for *king* in various Slavic languages (e.g., Russian, Polish, Czech) was coined after his name.

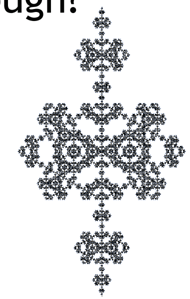
Is our sort good enough?

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

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



Eyes
by John Devor
and Eric Montgomery

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

Which of these is true?

- Our sort procedure is too slow for VISA because its running time is in $O(n^2)$
- Our sort procedure is too slow for VISA because its running time is in $\Omega(n^2)$
- Our sort procedure is too slow for VISA because its running time is in $\Theta(n^2)$

Which of these is true?

- ~~Our sort procedure is too slow for VISA because its running time is in $O(n^2)$~~
- Our sort procedure is too slow for VISA because its running time is in $\Omega(n^2)$
- Our sort procedure is too slow for VISA because its running time is in $\Theta(n^2)$

Knowing a running time is in $O(f)$ tells you the running time is not *worse* than f. This can *only* be good news. It doesn't tell you anything about how bad it is. *(Lots of people and books get this wrong.)*

Sorting Cost

```
(define (best-first-sort lst cf)
  (if (null? lst) lst
     (let ((best (find-best lst cf)))
       (cons best (best-first-sort (delete lst best) cf)))))
(define (find-best lst cf)
   (if (null? (cdr lst)) (car lst)
      (pick-better cf (car lst) (find-best (cdr lst) cf))))
```

The running time of best-first-sort is in $\Theta(n^2)$ where n is the number of elements in the input list.

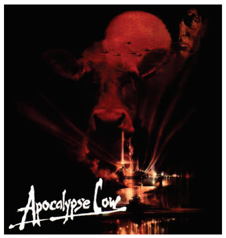
Assuming the comparison function passed as cf has constant running time.

Divide and Conquer sorting?

- **Best first 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.
  - Let's write this together!
  - I'll start on the next slide ...

insert-sort

```
(define (insert-sort lst cf)
  (if (null? lst) null
     (insert-one (car lst)
                 (insert-sort (cdr lst) cf) cf)))
```



Try writing insert-one.

(define (insert-one element lst cf) ...)

(insert-one 2 (list 1 3 5) <) --> (1 2 3 5)

insert-one

```
(define (insert-one el lst cf)
  (if (null? lst) (list el)
     (if (cf el (car lst)) (cons el lst)
        (cons (car lst)
              (insert-one el (cdr lst) cf)))))
```


How much work is insert-sort?

```
(define (insert-sort lst cf)
  (if (null? lst) null
     (insert-one (car lst) (insert-sort (cdr lst) cf) cf)))

(define (insert-one el lst cf)
  (if (null? lst) (list el)
     (if (cf el (car lst)) (cons el lst)
        (cons (car lst) (insert-one el (cdr lst) cf)))))
```

How many times does insert-sort evaluate insert-one?

How much work is insert-sort?

```
(define (insert-sort lst cf)
  (if (null? lst) null
     (insert-one (car lst) (insert-sort (cdr lst) cf) cf)))

(define (insert-one el lst cf)
  (if (null? lst) (list el)
     (if (cf el (car lst)) (cons el lst)
        (cons (car lst) (insert-one el (cdr lst) cf)))))
```

How many times does insert-sort evaluate insert-one?

n times (once for each element)

running time of insert-one is ?

How much work is insert-sort?

```
(define (insert-sort lst cf)
  (if (null? lst) null
     (insert-one (car lst) (insert-sort (cdr lst) cf) cf)))

(define (insert-one el lst cf)
  (if (null? lst) (list el)
     (if (cf el (car lst)) (cons el lst)
        (cons (car lst) (insert-one el (cdr lst) cf)))))
```

How many times does insert-sort evaluate insert-one?

n times (once for each element)

running time of insert-one is in $\Theta(n)$

How much work is insert-sort?

```
(define (insert-sort lst cf)
  (if (null? lst) null
     (insert-one (car lst) (insert-sort (cdr lst) cf) cf)))

(define (insert-one el lst cf)
  (if (null? lst) (list el)
     (if (cf el (car lst)) (cons el lst)
        (cons (car lst) (insert-one el (cdr lst) cf)))))
```

How many times does insert-sort evaluate insert-one?

n times (once for each element)

running time of insert-one is in $\Theta(n)$

insert-sort has running time in $\Theta(n^2)$ where n is the number of elements in the input list