How Fast Is It?: Algorithmic Complexity

- Three aspects of a program
  - Functionality: What does it do?
  - Design: How does it do it?
  - Performance: How fast is it?
- Performance depends on
  - Speed and capacity of hardware (CPU, bus, network...)
  - Software environment (OS, other programs)
  - Design of program

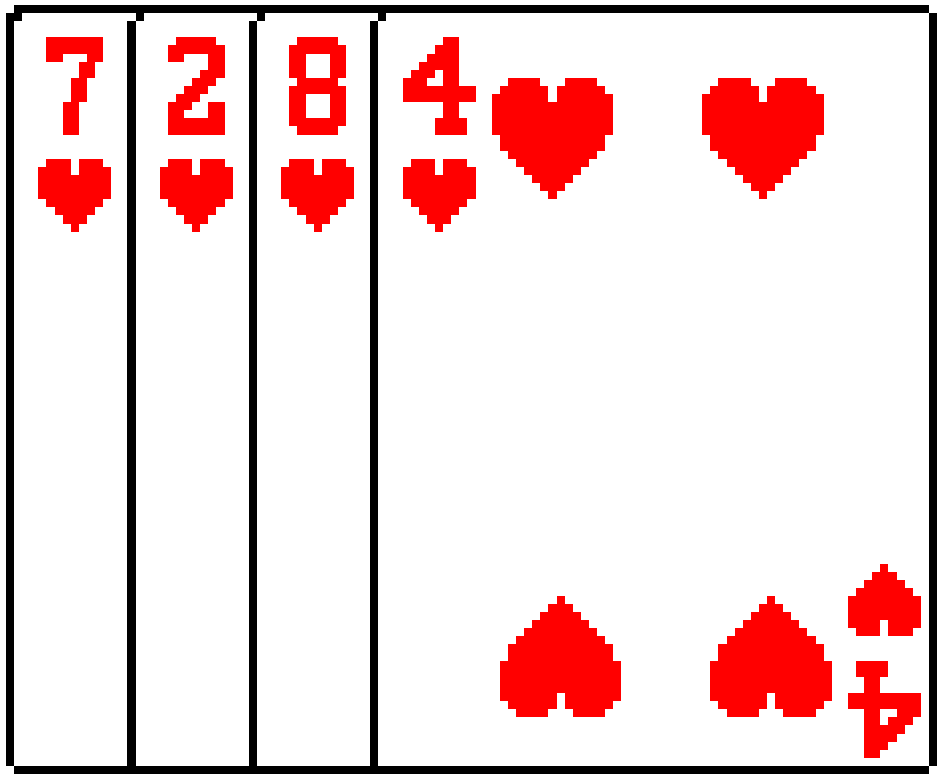
The Impact of Design on Performance

- Tradeoff of resources
  - RAM vs. disk
  - Speed vs. space
  - Remote vs. local
- Squeezing every ounce
  - Virtual machine vs. native code
  - Compiler optimizations
  - Folk wisdom
    - Avoid function calls
    - Don't launch an external process

Algorithmic Complexity

- An algorithm is a formulaic method of producing an output based on input
  - Sorting students by height
  - Finding the shortest path from your house to Starbucks
  - Compressing a song into a smaller file
  - Go hand-in-hand with data structures
- Some ways of doing it are just plum faster
  - And some take less space

Insertion Sort

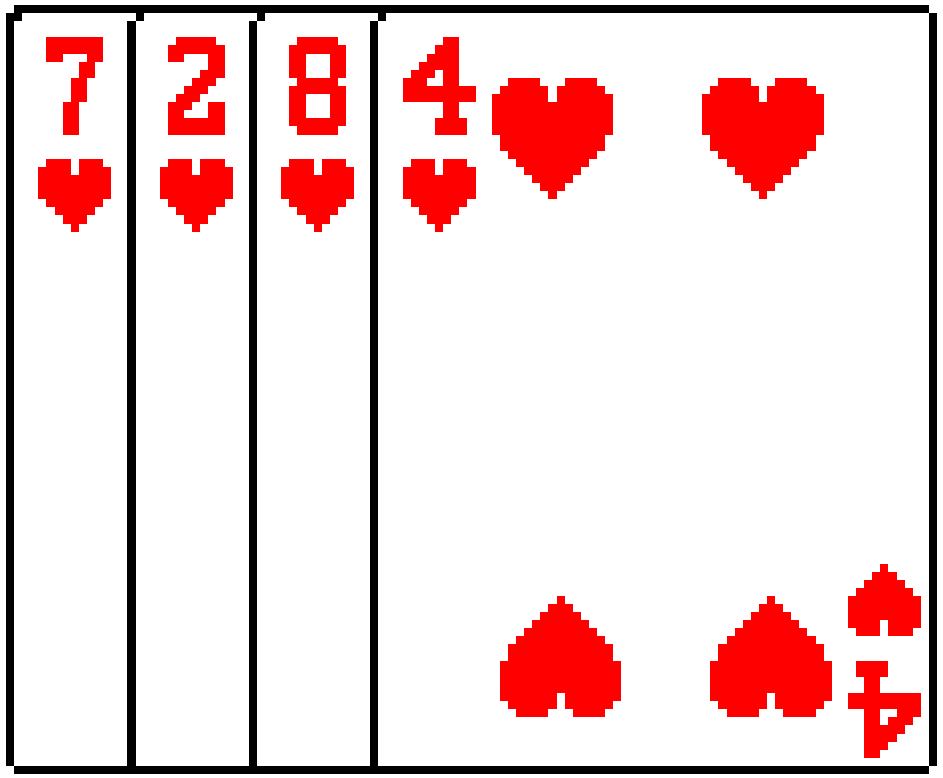


- The typical way people sort cards
- Start with all the cards in the left pile
- One by one, put them into the right pile, finding the correct spot in the order

Insertion Sort

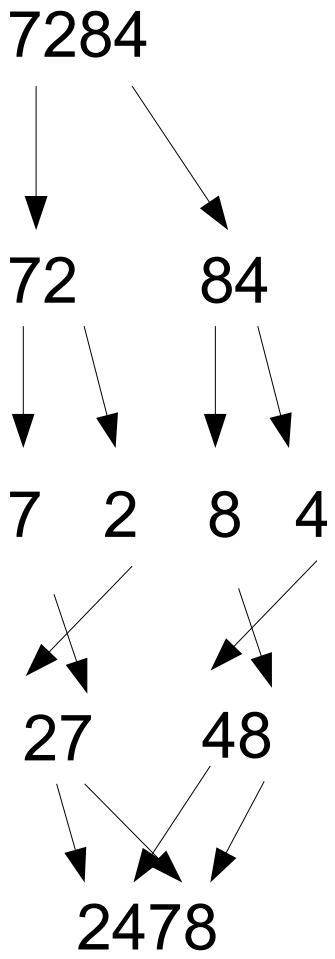
Left pile	Right pile	Comparisons	Moves
7284			
728	4	0	1
72	48	1	1
7	248	1	1
	2478	3	1
		5	4

Merge Sort



- Divide the pile into two halves
- Sort each half
- Combine the two resulting piles into a third pile, by drawing the two top cards and comparing them

Merge Sort



split into two piles

split each of those piles in half

to merge each pile, 1 comparison + 2 moves

to merge, 3 comparisons + 4 moves

But Which Is Faster in General?

- It depends how much a split or move costs, for n cards
- We need a way to describe the speed of an algorithm in simple terms, based on the input size
- We mainly care about big input sizes, and the worst-case time

Running Time Formulas

- For insertion sort, worst case is
  - 1+2+3+...+n-1 comparisons
  - Plus n insertions
- For merge sort, worst case is
  - One split
  - Plus twice however long it takes to sort each half
  - Plus n-1 comparisons for merge
- Hmm... how long is that?
- Let's look just at the cost of comparisons, for now

Summations and Recurrence Equations

- Comparisons in insertion sort

$$T(n) = 0 + 1 + \cdots + (n-1) = \sum_{i=1}^{n} i - 1 = \frac{n(n-1)}{2}$$

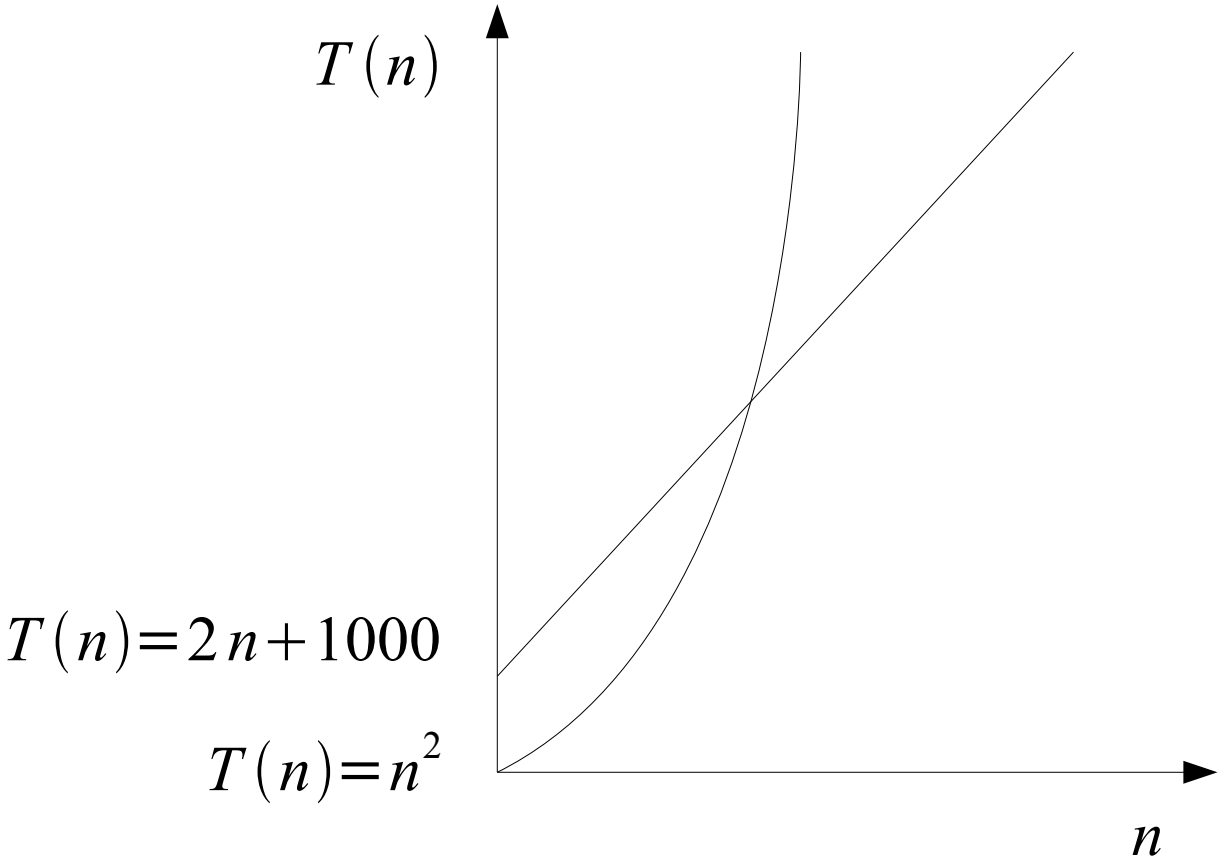
- Comparisons in merge sort

$$T(1) = 0$$
$$T(n) = 2T(n/2) + (n-1)$$

- It looks a little cleaner, but how does that help?

Asymptotic Behavior

- In the long run, which of these is faster?



Asymptotic Notation

- The little terms don't matter
- We don't even care about constant coefficients
- What matters is the term with the biggest exponent
- So, let's use a simple notation for expressing what matters

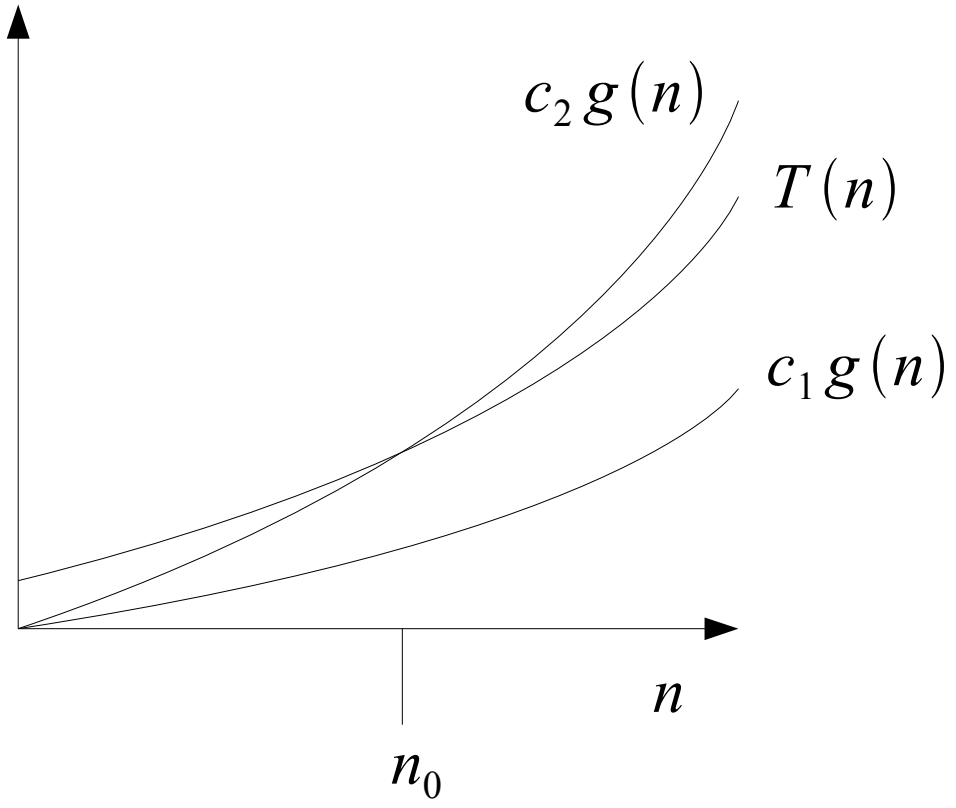
$$n^2 \in \Theta(n^2)$$

$$2n + 1000 \in \Theta(n)$$

$$124582393 \in \Theta(1)$$

Asymptotic Bounds

- When you say that a function $T(n)$ is $\Theta(g(n))$, it means you can pick constants c_1, c_2, and n_0 such that for all $n \geq n_0$, $0 \leq c_1 g(n) \leq T(n) \leq c_2 g(n)$.



Back to Our Sorts

- So for insertion sort, $T(n)=\frac{n(n-1)}{2}\in\Theta(n^2)$
- For merge sort, we have a recurrence equation

$$T(1)=0$$
$$T(n)=2T(n/2)+(n-1)$$

- Maybe we don't care about $T(1)$, but what do we do with the recurrence in the $T(n)$ case?
- For now, we'll make a guess

$$T(n)\in\Theta(n\log(n))$$