

CSCI 303 Introduction to Algorithms
Spring 2007
January 8, 2007 class notes

Class website: <http://www-scf.usc.edu/~csci303>

Grade breakup:

40% final exam
30% midterm exam
20% two quizzes
10% class participation

The homework will not be graded, but the quiz questions will come straight and unaltered from the homework. Class participation will be assigned twice during the semester, by the professor, during the midterm and the final exams. Never being seen earns 0 points out of a possible 5. Being seen in class earns 1 point. Having ever spoken, just once, during class earns 2 points. Further participation earns more points.

For the quizzes, the midterm, and the final exams, please bring a blue book. The midterm and final exam questions will come from the material discussed in lecture and the assigned sections of the textbook.

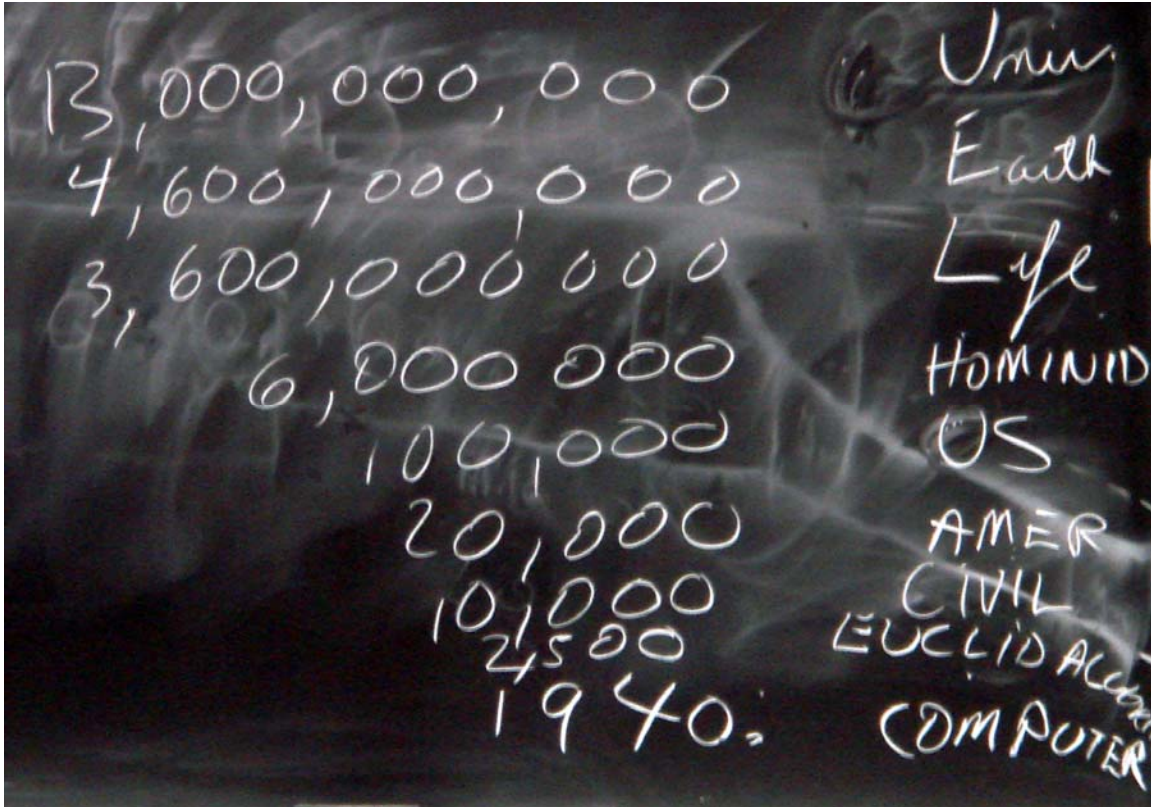
Textbook: We will be using the 2nd edition of [Introduction to Algorithms](#) by Thomas H Cormen, Charles E. Leiserson, Ronald L. Rivest and Cliff Stein.

Introduction to Algorithms.

Algorithm: a computer program (for our purposes).

First modern computer was built in 1946. This computer had a property that other “computers” before it did not: universality. A universal computer is one that can compute all that is computable.

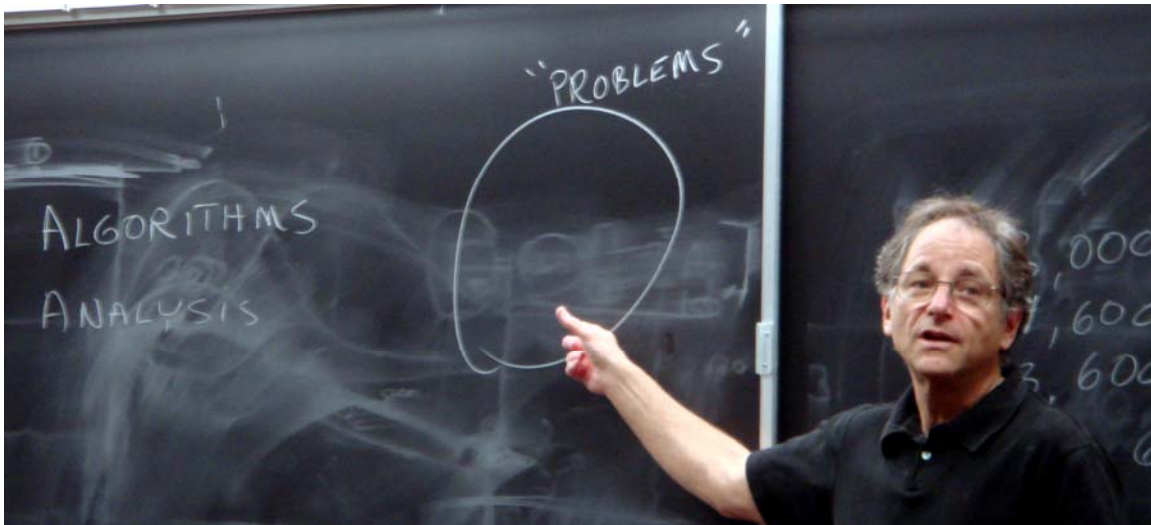
How old are various things:



Computer programs, or algorithms, can compute various things. For example, a computer can compute the area of a rectangle, multiply numbers, find the shortest path in a graph, etc.

In the real world, we have to deal with memory and energy constraints, but this class is largely mathematical, so we don't worry about such constraints.

What if computers can't solve every problem out there? We think there is no more powerful tool than a computer. Turns out, most problems cannot be solved on a computer! And of those that can be solved, most can't be solved nearly fast enough to matter.



When we say *number*, we mean a non-negative integer, or an element of $\{0, 1, 2, \dots\}$.
 Some problems computers can solve:

GCD problem:

Input: a, b : positive integers

Output: g : positive integer such that g is the greatest common divisor of a and b , or $\text{gcd}(a,b)$.

Primality problem:

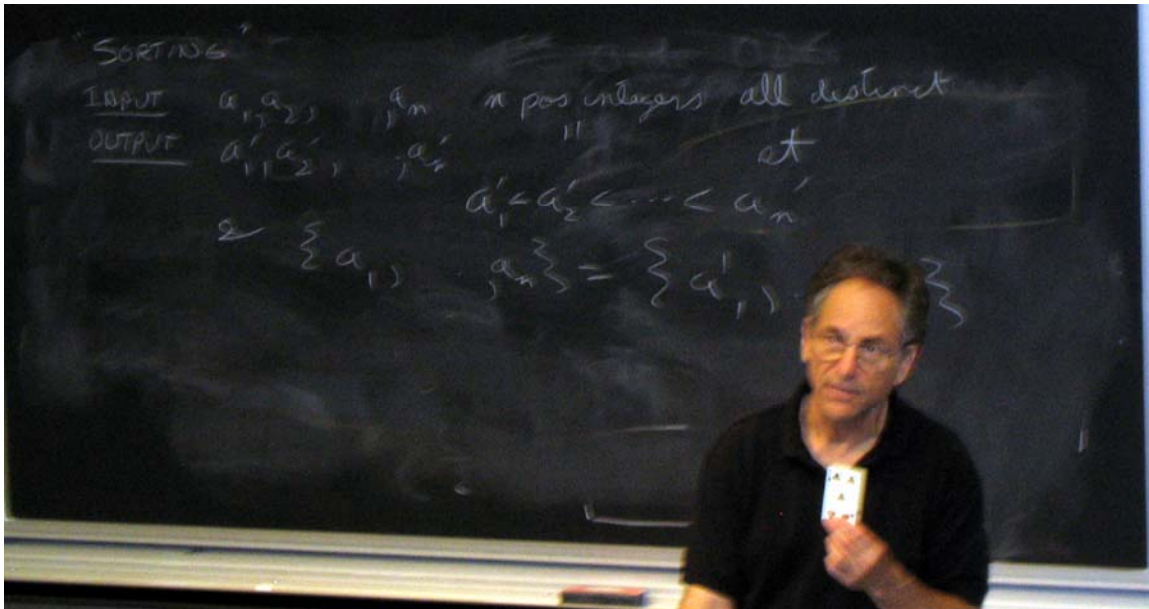
Input: a : positive integer

Output: 1 if a is prime and 0 otherwise

Sorting problem:

Input: a_1, a_2, \dots, a_n n distinct positive integers

Output: a'_1, a'_2, \dots, a'_n n positive integers such that $a'_1 < a'_2 < \dots < a'_n$ and $\{a_1, a_2, \dots, a_n\} = \{a'_1, a'_2, \dots, a'_n\}$



Two algorithms: insertion sort and merge sort.

Demonstrations on cards:

Insertions sort: pick up each card one at a time and insert it into its correct spot relative to the previously picked up cards.

Merge sort: split all cards into piles of 2 and sort those piles, then merge the piles into piles of 4, 8, 16, etc. by comparing the top cards.

Ways to compare algorithms:

- how long the algorithm takes
- how much memory the algorithm uses
- how easy is it to implement

We usually look at only the running time. For sorting, we count comparisons.

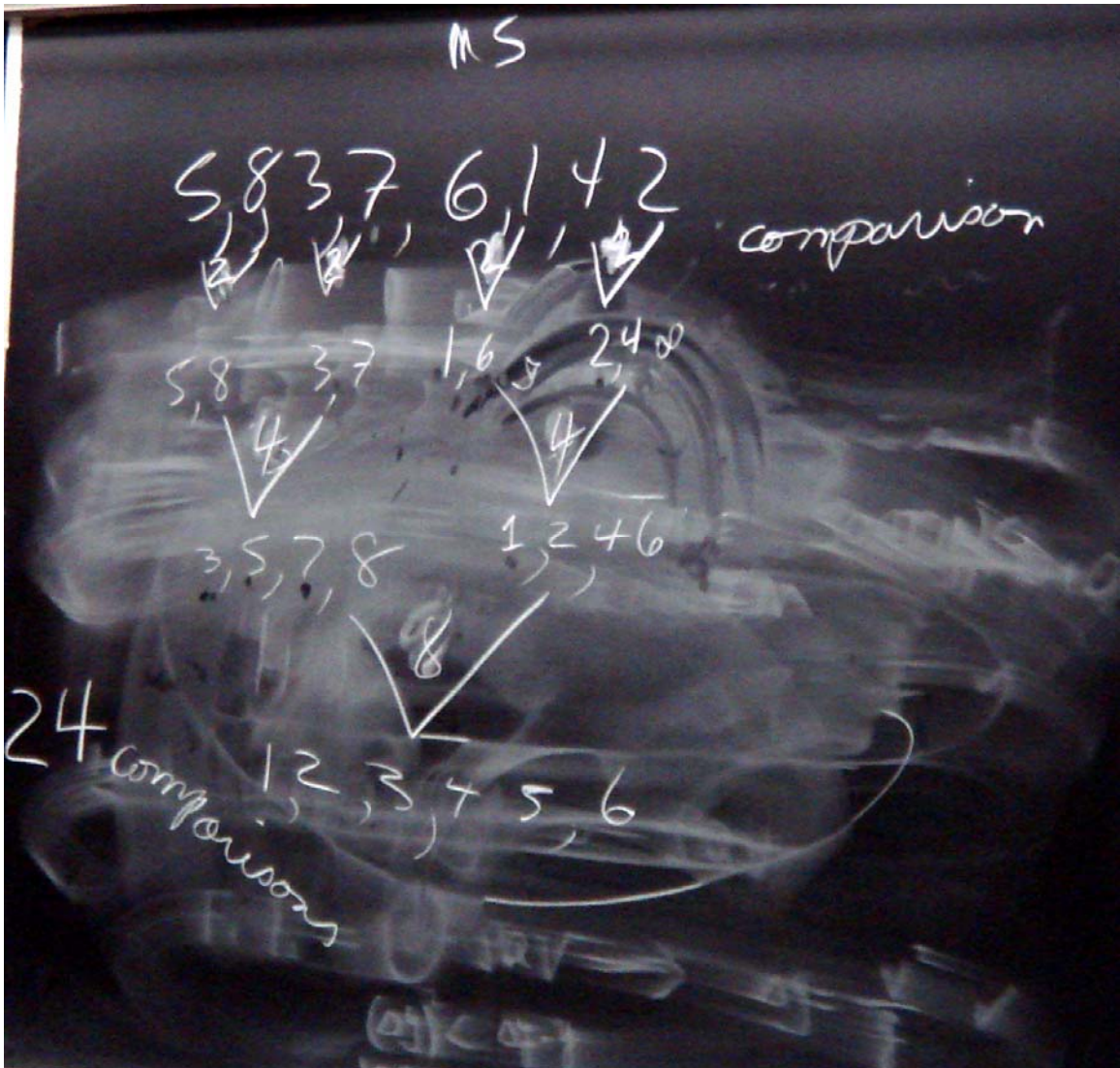
The number of comparisons insertion sort (left) performs to sort the list 5,8,3,7,6,1,4,2:

$$T_{IS}(5,8,3,7,6,1,4,2) = 25$$

The number of comparisons merge sort (right) performs to sort the list 5,8,3,7,6,1,4,2:

$$T_{MS}(5,8,3,7,6,1,4,2) = 24$$

In the worst case, insertion sort's input is in reverse order. The worst case running time of insertion sort for a list of size 8 is $T_{IS}^{WC}(8) = \sum_{i=1}^7 i = 28$. Note that $\sum_{i=1}^n i = \frac{n(n+1)}{2}$. The worst case running time of merge sort for a list of size 8 is $T_{MS}^{WC}(8) = 17$.



Asymptotic complexity: how many comparisons does it take to sort n -sized lists as n gets bigger and bigger? If for all n , algorithm A makes fewer comparisons than B, then we say that A is asymptotically better. Even if for some small n , B performs fewer comparisons, if for all large n A performs fewer, then we still say A is asymptotically better.

