Chapter 1
The Role of Algorithms in Computing

The slides for this course are based on the course textbook: Cormen, Leiserson, Rivest, and Stein, *Introduction to Algorithms*, 3rd edition, The MIT Press, McGraw-Hill, 2001.

Chapter 1 Topics

- Algorithms
- Algorithms as a Technology

Algorithms

Goals:

Learn techniques of algorithm design and analysis so that you can:

- develop algorithms,
- show that they give the correct answer, and
- understand their efficiency

Algorithms

What is an algorithm?

- An *algorithm* is any well-defined computational procedure that takes some value, or set of values, as input and produces some value, or set of values, as output. An algorithm is thus a sequence of computational steps that transform the *input* into the *output*. (CLRS, p. 5)

Algorithms

What is an algorithm?

- An *algorithm* is a tool for solving a well-specified computational problem. The statement of the problem specifies in general terms the desired input/output relationship . The algorithm describes a specific computational procedure for achieving that input/output relationship. (CLRS, p. 5)

Algorithms

Formal definition of the *sorting problem*:

Input: A sequence of numbers $\langle a_1, a_2, \ldots, a_n \rangle$

Output: A permutation (reordering) $\langle a_1', a_2', \ldots, a_n' \rangle$ of the input sequence such that $a_1' \leq a_2' \leq \ldots \leq a_n'$

Algorithms

Instance: The input sequence <14, 2, 9, 6, 3> is an *instance* of the sorting problem.

An *instance* of a problem consists of the input (satisfying whatever constraints are imposed in the problem statement) needed to compute a solution to the problem.

Algorithms

Correctness: An algorithm is said to be *correct* if, for every instance, it halts with the correct output. We say that a correct algorithm *solves* the given computational problem.

Algorithms as a Technology

Efficiency: Algorithms that solve the same problem can differ enormously in their efficiency. Generally speaking, we would like to select the most efficient algorithm for solving a given problem.

Algorithms as a Technology

Space efficiency: Space efficiency is usually an all-or-nothing proposition; either we have enough space in our computer’s memory to run a program implementing a specific algorithm, or we do not. If we have enough, we’re OK; if not, we can’t run the program at all. Consequently, analysis of the space requirements of a program tend to be pretty simple and straightforward.

Algorithms as a Technology

Time efficiency: When we talk about the efficiency of an algorithm, we usually mean the *time* requirements of the algorithm: how long would it take a program executing this algorithm to solve the problem? If we could afford to wait around forever (and could rely on the power company not to lose the power), it wouldn’t make any difference how efficient our algorithm was in terms of time. But we can’t wait forever; we need solutions in a reasonable amount of time.

Algorithms as a Technology

Space efficiency:

Note that space requirements set a minimum lower bound on the time efficiency of the problem.

Suppose that our data structure is a single-dimensioned array with n = 100 elements in it. Let's say that the first step in our algorithm is to execute a loop that just copies values into each of the 100 elements. Then our algorithm must take at least 100 iterations of the loop. So the running time of our algorithm is at least O(n), just from setting up (initializing) our data structure!

Algorithms as a Technology

Time efficiency of two sorts:

Suppose we use insertion sort to sort a list of numbers. Insertion sort has a time efficiency roughly equivalent to $c_1 \cdot n^2$. The value n is the number of items to be sorted. The value c_1 is a constant which represents the overhead involved in running this algorithm; it is independent of n.

Compare this to merge sort. Merge sort has a time efficiency of $c_2 \cdot n \cdot \lg n$ (where lg n is the same as $\log_2 n$).

Algorithms as a Technology

n	*Insertion sort - O(n^2)^*	*Merge sort – O(n lg n)*
4	16	8
8	64	24
16	256	64
32	1024	160
64	4096	384
128	16,394	896
256	65,536	2048
512	262,144	4608
1024	1,048,576	10,240
1,048,576	~1,000,000,000,000	20,971,520

Algorithms as a Technology

Time efficiency of two sorts:

Do the two constants, c_1 and c_2, affect the result?

Yes, but only with low values of n.

Suppose that insertion sort is hand-coded in machine language for optimal performance and its overhead is very low, so that $c_1 = 2$.

Now suppose that merge sort is written in Ada by an average programmer and the compiler doesn't do a good job of optimization, so that $c_2 = 50$.

Algorithms as a Technology

Time efficiency of two sorts:

To make things worse, suppose that insertion sort is run on a machine that executes 1 billion instructions per second, while merge sort is run on a slow machine that executes only 10 million instructions per second.

Now let’s sort 1 million numbers:

insertion sort: $\dfrac{2 \bullet \left(10^6\right)^2 \mathit{instructions}}{10^9 \mathit{instructions} / \sec} = 2000 \sec.$

merge sort: $\dfrac{50 \bullet 10^6 \lg 10^6 \mathit{instructions}}{10^7 \mathit{instructions} / \sec} \approx 100 \sec.$

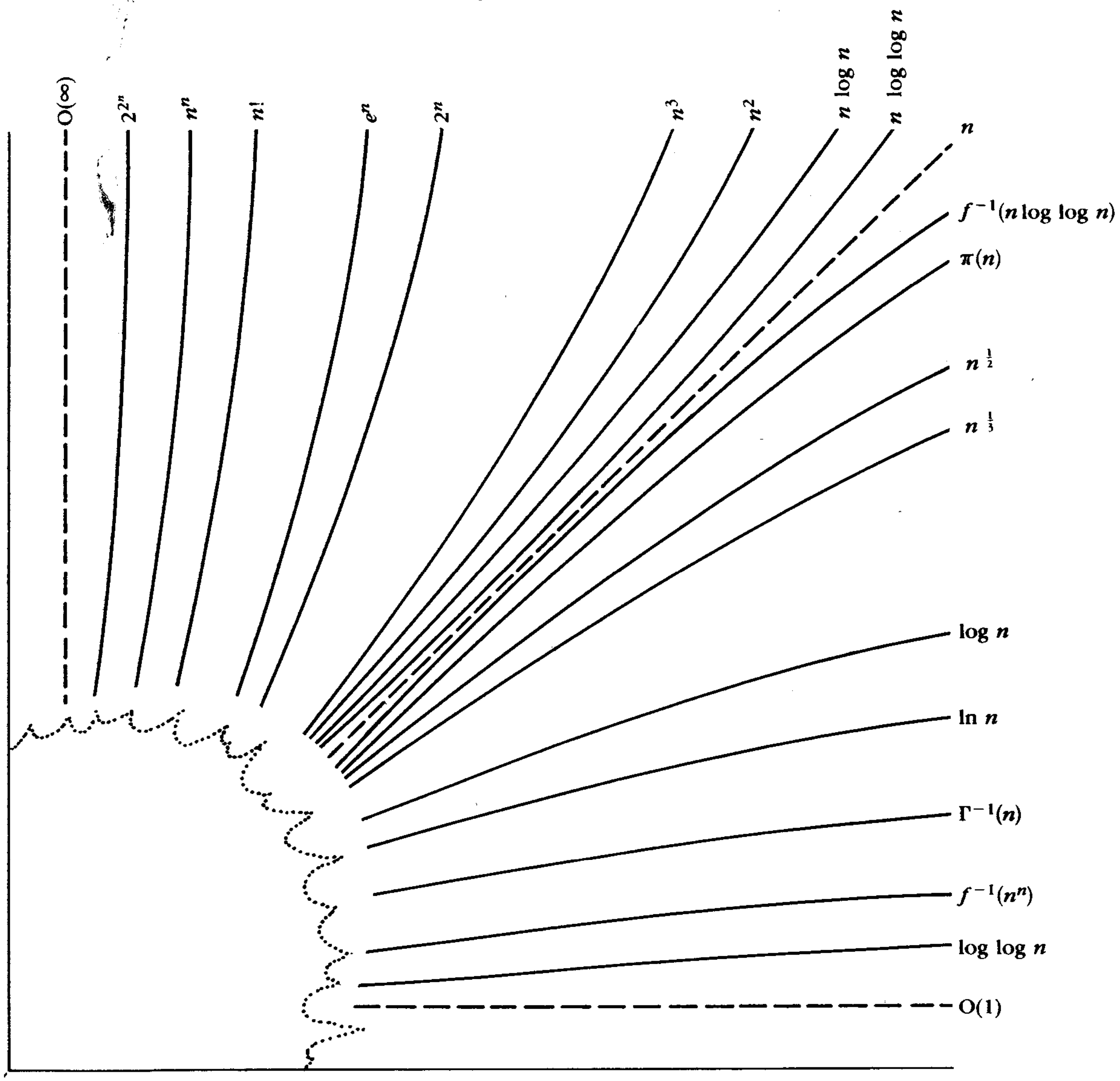

Figure 14.3. Relative asymptotic behavior of typical $f(n)$ (not drawn to scale)

From E. D. Reilly and F. D. Federighi, *Pascalgorithms*, Boston: Houghton Mifflin, 1989, p. 501.

Conclusion

What does this mean for computer science?

It means that using efficient algorithms can be even more important that building faster computers: more efficient thinking beats more efficient hardware!

And this means that algorithms are definitely worth studying.