CS 577 Introduction to Algorithms: Introduction

Jin-Yi Cai

University of Wisconsin–Madison

- Computational Problem: A well-defined input/output relationship. E.g., sorting, connected components, greatest common divisor (GCD), matrix multiplication.
- Algorithm: A well-defined procedure that takes something (as input) and produces something (as output).
  - Existed before computers: e.g., the Euclidean algorithm for GCD. [Section 31.2 of the textbook if interested]
- An algorithm correctly solves a problem if, for every input instance, it halts with the correct output.

- Correctness: Provably correct in this course.
- Performance: (mostly) time complexity, and space complexity (or other computational resources).
- How to measure the running time of an algorithm?
  - the random-access machine (RAM) model
    [Section 2.2 of the textbook for more details]
  - cells storing integers and rational numbers
  - basic operations: arithmetic/data movement/control
  - count the number of basic operations

A generic form of InsertionSort.

InsertionSort(A), where $A = \langle a_1, \ldots, a_n \rangle$ is a sequence of integers:

1. Create an empty list B
2. For i from 1 to n

   Enumerate the list B backwards to find the first integer in B smaller than a_i; insert a_i right after that integer.

This "backwards" is not essential. But is more convenient in the actual implementation using the array of A itself.

A generic form of InsertionSort.
InsertionSort(A), where $A = \langle a_1, \ldots, a_n \rangle$ is a sequence of integers:

1. Create an empty list B
2. For i from 1 to n

   Enumerate the list B backwards to find the first integer in B smaller than a_i; insert a_i right after that integer.

This "backwards" is not essential. But is more convenient in the actual implementation using the array of A itself.

Initially, $A[1:1]$ is sorted.
Start with $j = 2$ to n, insert $A[j]$ to its rightful place among $A[1:j-1]$. (*Inductively assume* $A[1:j-1]$ *is currently sorted*.)
Example: $A[1:6] = \langle 5, 2, 4, 6, 1, 3 \rangle$.
This is an "in place" sorting algorithm: Use $A[1:n]$ to hold the data at all time, with only a constant amount of extra space.

We use $T(A)$ to denote the number of basic operations it uses when the input is A, and we are interested in its worst-case time complexity: For $n \geq 1$, let

$$T(n) = \max_{\text{all } A \text{ of length } n} T(A).$$

Deriving exactly what $T(n)$ is can be very tedious, e.g., it depends on how we implement a list using a RAM.

In any reasonable implementation, other than at most a constant number of steps, the main cost of the algorithm is the loop.
The loop variable j goes through from 2 to n. for each chosen j, searching backwards until the "right place" is found for $A[j]$ takes k_j steps, a variable number of steps.
However, note that $k_j \leq j$.
Therefore, the total number of steps is at most

$$c_1 + c_2 \sum_{j=2}^{n} j < c_1 + c_2 \frac{n(n+1)}{2},$$

where c_1 and c_2 are some constants independent of A.

Different input instances yield different k_j's. If $A = \langle 1, 2, \ldots, n \rangle$ is already ordered increasingly, then $k_j = 1$ for all j. But when $A' = \langle n, n-1, \ldots, 1 \rangle$, we have $k_j = j$ for all j.

You should know how to prove an equality like this

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}.$$

In the homework you will be asked to do some slightly more involved induction proofs.

So in any case

$$T(A) \le c + c' \frac{n^2}{2} < Cn^2.$$

Usually we make the following two simplifications in analysis:

- focus on the dominant term: keep $c_2 n^2/2$ only
- suppress the constant coefficient: keep n^2 only

More formally, we use the asymptotic notation: $T(n) = \Theta(n^2)$ (to be defined next).

We focus on the asymptotic performance to

- avoid the tedious analysis of the constants;
- understand the intrinsic (and machine-independent) complexity of an algorithm;
- concentrate on the dominant term when designing an algorithm because this decides its performance when the inputs are large.

But what if the hidden constant is really really large: E.g., for an algorithm with $T(n) = 10^{100}n$ to perform better than an algorithm with $T(n) = n^2$, n needs to be 10^{100}.

- Fortunately the algorithms we cover in the course are well polished and have low hidden constants.

Let $f(n)$ and $g(n)$ are functions that map $n = 1, 2, \ldots$ to real numbers, then we let

$$O(g(n)) = \Big\{ f(n) : \exists \text{ constants } c > 0 \text{ and } n_0 > 0 \\ \text{s.t. } 0 \le f(n) \le c \cdot g(n) \text{ for all } n \ge n_0 \Big\}$$

Check Figure 3.1 (b) of the textbook. Usually we use

$$f(n) = O(g(n)) \quad \text{to denote} \quad f(n) \in O(g(n))$$

Let $f(n)$ and $g(n)$ are functions that map $n = 1, 2, \ldots$ to real numbers, then we let

$$\Omega(g(n)) = \Big\{ f(n) : \exists \text{ constants } c > 0 \text{ and } n_0 > 0 \\ \text{s.t. } 0 \leq g(n) \leq c \cdot f(n) \text{ for all } n \geq n_0 \Big\}$$

Check Figure 3.1 (c) of the textbook. Usually we use

$$f(n) = \Omega(g(n)) \quad \text{to denote} \quad f(n) \in \Omega(g(n)).$$

Let $f(n)$ and $g(n)$ are functions that map $n = 1, 2, \ldots$ to real numbers, then we let

$$\begin{aligned}\Theta(g(n)) &= \Big\{ f(n) : \exists \text{ constants } c_1, c_2 > 0 \text{ and } n_0 > 0 \\ &\qquad \text{s.t. } 0 \le c_1 \cdot g(n) \le f(n) \le c_2 \cdot g(n) \text{ for all } n \ge n_0 \Big\}\end{aligned}$$

Check Figure 3.1 (a) of the textbook. Usually we use

$$f(n) = \Theta(g(n)) \quad \text{to denote} \quad f(n) \in \Theta(g(n)).$$

- Read Section 3.1 of the textbook to get comfortable about the asymptotic notation. Will be used in almost every lecture.
- Back to the InsertionSort, we have $T(n) = \Theta(n^2)$. To formally prove this, use limit from calculus:

$$\lim_{n \to \infty} \frac{T(n)}{n^2} = \frac{c_2}{2}$$

Let $\epsilon > 0$ be any constant. By the definition of limit, there exists a large enough n_0 such that

$$\left| \frac{T(n)}{n^2} - \frac{c_2}{2} \right| < \epsilon, \quad \text{for all } n \geq n_0.$$

We introduce two more asymptotic notations: $o(\cdot)$ and $\omega(\cdot)$. Let $f(n)$ and $g(n)$ be functions that map $n = 1, 2, \ldots$ to real numbers, then we let

$$o(g(n)) \;=\; \Big\{ f(n) : \text{for any constant } c > 0, \text{ there exists an } n_0 > 0 \text{ s.t. } 0 \le f(n) < c \cdot g(n) \text{ for all } n \ge n_0 \Big\}$$

Usually we use

$$f(n) = o(g(n)) \quad \text{to denote} \quad f(n) \in o(g(n)).$$

It means asymptotically $f(n)$ is dominated by $g(n)$, or the order of $f(n)$ is strictly less than that of $g(n)$.

Note the crucial difference between $O(g(n))$ and $o(g(n))$: "there exists a constant $c > 0$" versus "for any constant $c > 0$". Usually $f(n) = o(g(n))$ can be proved using

$$\lim_{n \to \infty} \frac{f(n)}{g(n)}$$

when the limit exists.

For example,

$$\lim_{n\to\infty} \frac{n^{1.9}}{n^2} = 0$$

By the definition of limits, this implies that for any constant $c > 0$, there exists an $n_0 > 0$ such that

$$\frac{n^{1.9}}{n^2} < c, \quad \text{for all } n \geq n_0.$$

It follows from the definition of $o(n^2)$ that $n^{1.9} = o(n^2)$, and in general, $n^a = o(n^b)$ for all constants $0 < a < b$.

Similarly, we have for any positive constants $a > 1$ and $b, c > 0$,

$$\lim_{n\to\infty} \frac{n^b}{a^n} = 0 \quad \Rightarrow \quad n^b = o(a^n) \tag{1}$$

$$\lim_{n\to\infty} \frac{\lg^c n}{n^b} = 0 \quad \Rightarrow \quad \lg^c n = o(n^b) \tag{2}$$

Here the limit in (2) follows from the one in (1), while (1) can be proved using the l'Hopital's rule.

Note that what base in the notation $\lg n$ is unimportant in this statement, since $\log_a n = \frac{\log_b n}{\log_b a}$.

Also $\lg^c n$ denotes $(\lg n)^c$.

As a result, we have

$$\lg n < \lg^2 n < \cdots < n < n \lg n < n^2 < n^3 < \cdots < 2^n < 3^n < n!$$

where $<$ means little-o or "is asymptotically dominated by".
A useful asymptotic formula called Stirling's approximation is that

$$n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n.$$

Also read Section 3.2 if you are not familiar with common functions like the floors $\lfloor\cdot\rfloor$, ceilings $\lceil\cdot\rceil$, polynomials, exponentials or logarithms.

Finally, let $f(n)$ and $g(n)$ are functions that map $n = 1, 2, \dots$ to real numbers, then

$$\begin{aligned}\omega(g(n)) \;=\; & \Big\{ f(n) : \text{for any constant } c > 0, \text{ there exists an } n_0 > 0 \\ & \text{s.t. } 0 \le c \cdot g(n) < f(n) \text{ for all } n \ge n_0 \Big\}\end{aligned}$$

Usually we use

$$f(n) = \omega(g(n)) \quad \text{to denote} \quad f(n) \in \omega(g(n)).$$

Check that $f(n) = \omega(g(n))$ if and only if $g(n) = o(f(n))$. It means that $f(n)$ dominates $g(n)$ asymptotically.

We just described InsertionSort and showed that its worst-case running time is $\Theta(n^2)$. However, we did not prove its correctness. Check Figure 2.2 for the intuition why it is correct. To give a formal proof, we use (mathematical) induction.

Induction is usually used to prove that a mathematical statement, involving a parameter n, holds for all $n = 1, 2, \ldots$ It has 3 steps:

1. **Basis**: Check that the statement holds for $n = 1$.
2. **Induction Step**: Prove that for any $k \geq 2$, if the statement holds for $1, 2, \ldots, k-1$, then it also holds for k.
3. Conclude that, by induction, the statement holds for $1, 2, \ldots$

Here is how to get the conclusion from the Basis and Induction Step: Let $n \geq 1$ be any positive integer. Then from the Basis, the statement holds for 1. Next by applying the Induction Step for $k = 2$, we know that the statement holds for 1 and 2. Keep applying the Induction Step for $n - 1$ times, we know that the statement holds for $1, 2, \ldots, n$, and this finishes the proof.

In the Induction Step, we assume that the statement holds for $1, 2, \ldots, k-1$. This assumption is usually referred to as the Inductive Hypothesis. The goal of the Induction Step is then to use the Inductive Hypothesis to prove the statement for k. For InsertionSort, we prove the following theorem:

Theorem

Let $n \geq 1$ be a positive integer. If $A = (a_1, \ldots, a_n)$ is the input sequence of InsertionSort, then after the ith loop, where $i = 1, 2, \ldots, n$, the sublist $A[1:i]$ of length i and is a nondecreasing permutation of the first i integers of the original A.

The intuition of this statement comes from the example of Figure 2.2 in the textbook. The correctness of InsertionSort clearly follows from this theorem.