Analysis of Algorithms I: Asymptotic Notation, Induction, and MergeSort

Xi Chen

Columbia University

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

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

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

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

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 $a, b : 0 < a < b$.

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.

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". 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 $n = 1, 2, \ldots$ to real numbers, then

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

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.

In the last class, we 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. Here is an example:

Theorem

For any $n \geq 1$, we have $1^2 + 2^2 + \cdots + n^2 = n(n+1)(2n+1)/6$.

Proof.

1. **Basis**: Easy to check that the equation holds for $n = 1$.
2. **Induction Step**: Let $k \geq 2$ be any integer, and assume the equation holds for $1, 2, \ldots, k-1$ (the Inductive Hypothesis). In particular, this implies that it holds for $k-1$. Thus,

$$\begin{aligned} 1^2 + \cdots + k^2 &= (1^2 + \cdots + (k-1)^2) + k^2 \\ &= (k-1)k(2k-1)/6 + k^2 \\ &= k(k+1)(2k+1)/6 \end{aligned}$$

3. Conclude that the equation holds for any $n = 1, 2, \ldots$

Now back to 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 list B is of length i and is a nondecreasing permutation of the first i integers of 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.

Proof.

1. **Basis**: The statement holds for $i = 1$ because in the first loop, we simply insert the first integer a_1 into the empty list B.
2. **Induction Step**: Assume the statement holds for all $i : 1 \leq i \leq k-1$, for some $k : 2 \leq k \leq n$. In particular, after round $k-1$, B is of length $k-1$ and is a nondecreasing permutation of $a_1, \ldots, a_{k-1}$. Now in loop k, InsertionSort finds the first integer in B smaller than a_k and inserts a_k after that integer. As a result, the list B after round k is of length $k-1+1 = k$ and is a nondecreasing permutation of $a_1, \ldots, a_{k-1}, a_k$.
3. Conclude that the statement holds for $1, 2, \ldots, n$.

Note that the induction step in the last proof only works for $k : 2 \leq k \leq n$ because the algorithm only has n rounds. As a result, we conclude that the statement holds for $1, 2, \ldots, n$ (exactly what we need), instead of $1, 2, \ldots$

Next we describe MergeSort, a sorting algorithm that is asymptotically faster than InsertionSort. It is an example of the Divide-and-Conquer technique:

1. Divide the problem into smaller subproblems
2. Conquer (or solve) each of the subproblems recursively
3. Combine the solutions to the subproblems to get a solution to the original problem

MergeSort (A), where $A = (a_1, \ldots, a_n)$ is a sequence of n integers

1. If $n = 1$, return
2. MergeSort $(a_1, a_2, \ldots, a_{\lceil n/2 \rceil})$
3. MergeSort $(a_{\lceil n/2 \rceil + 1}, \ldots, a_{n-1}, a_n)$
4. Merge the two sequences obtained to produce a sorted permutation of the n integers in A

An implementation of line 4 can be found on page 31 of the textbook. It takes time $\Theta(n)$ (or in other words, cn steps for some positive constant c). And one can use induction (page 32 and 33) to show that, if the two sequences we obtain from the two recursive calls are sorted sequences of $a_1, \ldots, a_{\lceil n/2 \rceil}$ and $a_{\lceil n/2 \rceil + 1}, \ldots, a_n$, respectively, then the Merge subroutine outputs a sorted permutation of $a_1, \ldots, a_n$.

Practice the use of induction to show that

Theorem

MergeSort outputs correctly for sequences of length $n = 1, 2, \ldots$

To understand the performance of MergeSort, we use $T(n)$ to denote the number of steps it takes over sequences of length n. We get the following recurrence:

$$T(1) = \Theta(1) \qquad \text{usually let } \Theta(1) \text{ denote a positive constant}$$

$$T(n) = \Theta(1) + T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + \Theta(n) \qquad \text{for all } n \geq 2$$

For now, we focus the analysis on powers of 2: $n = 2^k$ for some integer $k \geq 0$. We will see later that this suffices to understand the asymptotic complexity of $T(n)$. For powers of 2, we have

$$T(1) = \Theta(1)$$

$$T(n) = 2T(n/2) + \Theta(n)$$

Putting the constants back gives us

$$T(1) = c_1 \tag{3}$$

$$T(n) = 2T(n/2) + c_2 n \tag{4}$$

for some positive constants c_1 and c_2.

Let $n = 2^k$. Then start with $T(n)$ and substitute it using (4):

$$T(n) = 2T(n/2) + c_2 n$$

Next, expand it further by substituting $T(n/2)$ using (4): $T(n/2) = 2T(n/4) + c_2 n/2$. We get

$$T(n) = 2(2T(n/4) + c_2 n/2) + c_2 n = 4T(n/4) + c_2 n + c_2 n$$

Repeat it for k rounds, and we get the so-called recursion tree in Fig 2.5 on page 38. $T(n)$ is the sum of all numbers on the nodes:

$$T(n) = c_2 nk + c_1 n = c_2 n \lg n + c_1 n$$

To see why this gives us $T(n) = \Theta(n \lg n)$, we use the fact that $T(n)$ is monotonically nondecreasing over $n = 1, 2, \ldots$ (Try to prove it using induction). Given any integer $n \geq 1$, we let n' denote the largest power of 2 that is smaller than n. This implies that $n' < n \leq 2n'$ and by the monotonicity of T, we have

$$T(n') \leq T(n) \leq T(2n')$$

Because both n' and $2n'$ are powers of 2, from the last slide,

$$T(n) \leq T(2n') = 2c_2 n' \lg(2n') + 2c_1 n' < 2c_2 n \lg(2n) + 2c_1 n$$

$$T(n) \geq T(n') = c_2 n' \lg n' + c_1 n' \geq c_2 (n/2) \lg(n/2) + c_1 (n/2)$$

From these two inequalities, it is easy to show $T(n) = \Theta(n \lg n)$.

From the analysis, we see that c_1 and c_2 do not affect the asymptotic complexity of $T(n)$. This is why we can suppress them and denote the running time of line 1 and 4 by $\Theta(1)$ and $\Theta(n)$, respectively, and we do not care what they are exactly in the analysis.

However, the coefficient 2 of $T(n/2)$ cannot be suppressed. Changing it would change the order of $T(n)$. In particular, if we change it to 3, then the recursion tree would look different: every internal node would have 3 children instead of 2. This would change the order of $T(n)$ significantly. We will use an example: Strassen's algorithm for matrix multiplication, to demonstrate the importance of this constant in the next class.