CLRS Notes

MIT 6.046J

Author: Haopeng Li

Date: July 27, 2022

ElegantLaTeX Program

Contents

Chapter 1 Analysis of Algorithms

Introduction

- ❑ *Insertion sort*
- ❑ *Merge sort*
- ❑ *Asymptotic analysis*
- ❑ *Recurrences*

Definition 1.1 (Algorithms)

The theoretical study of computer-program performance and resource usage. ♣

Note *Why study algorithms and performance?*

- *Algorithms help us to understand scalability.*
- *Performance often draws the line between what is feasible and what is impossible.*
- *Algorithmic mathematics provides a language for talking about program behavior.*
- *Performance is the currency of computing.*
- *The lessons of program performance generalize to other computing resources.*
- *Speed is fun!*

1.1 The problem of sorting

Problem 1.1(The problem of sorting)

- Input:sequence $< a_1, a_2, \cdots, a_n >$ of numbers.
- Output: permutation $< a'_1, a'_2, \cdots, a'_n >$ such that $a'_1 \leq a'_2 \leq \cdots \leq a'_n$

1.2 Insertion Sort

```
Insertion-Sort(A,n)
 for j <- 2 to n
   do key <- A[j]
     i <- j - 1
     while i > 0 and A[i] > key
       do A[i+1] <- A[i]
       i<- i-1
     A[i+1] = key
```

1.3 Running time

- The running time depends on the input:an already sorted sequence is easier to sort.
- Parameterize the running time by the size of the input,since short sequences are easier to sort than long ones.
- Generally,we seek upper bounds on the running time,because everybody likes a guarantee.

1.3.1 Kinds of Analysis

Definition 1.2 (Worst-Case(usually))

$T(n)$ = maximum time of algorithm on any input of size n. ♣

Definition 1.3 (Average-Case(Sometimes))

- *$T(n)$ = expected time of algorithm over all inputs of size n.*
- *Need assumption of statistical distribution of inputs.* ♣

Definition 1.4 (Best-case: (bogus))

Cheat with a slow algorithm that works fast on some input. ♣

Note *What is insertion sort's worst-case time?*

It depends on the speed of our computer:

- relative speed (on the same machine),
- absolute speed (on different machines).

Note *BIG IDEA:*

1. Ignore machine-dependent constants.
2. look at the growth of $T(n)$ as $n \to \infty$
3. "Asymptotic Analysis"

1.3.2 Θ-Notation

Definition 1.5 (Θ-Notation)

$\Theta(g(n)) = \{f(n)$: there exist positive constants c_1, c_2, and n_0 such that $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$ for all $n \geq n_0\}$ ♣

Note *Engineering:Drop low-order terms; Ignore leading constants.*

Example 1.1

$$3n^3 + 90n^2 - 5n + 6046 = \Theta\left(n^3\right)$$

1.3.3 Asymptotic performance

Note *When n gets large enough, a $\Theta\left(n^2\right)$ algorithm always beats a $\Theta\left(n^3\right)$ algorithm.*

Note

- *We shouldn't ignore asymptotically slower algorithms, however.*
- *Real-world design situations often call for a careful balancing of engineering objectives.*
- *Asymptotic analysis is a useful tool to help to structure our thinking.*

1.3.4 Insertion sort analysis

Worst case

Input reverse sorted

$$T(n) = \sum_{j=2}^{n} \Theta(j) = \Theta\left(n^2\right)$$

Average case

All permutations equally likely.

$$T(n) = \sum_{j=2}^{n} \Theta(j/2) = \Theta\left(n^2\right)$$

Note *Is insertion sort a fast sorting algorithm?*

- *Moderately so, for small n*
- *Not at all, for large n*

1.4 Merge Sort

```
Merge-Sort A[1..n]
  1. If n = 1,done
  2. Recurisively sort A[1...⌈n/2⌉] and A[⌈n/2⌉+1...n]
  3. Merge the 2 sorted lists.
```

Note *Key subroutine: MERGE*

1.4.1 Analyzing Merge Sort

Time $= \Theta(n)$ to merge a total of n elements (linear time).

$T(n)$	MERGE-SORT $A[1 \dots n]$
$\Theta(1)$	1. If $n = 1,$ done.
$2T(n/2)$	2. Recursively sort $A[1 \dots \lceil n/2 \rceil]$ and $A[\lceil n/2 \rceil + 1 \dots n]$.
$\Theta(n)$	3. "Merge" the 2 sorted lists

Note ***Sloppiness**: $2T(n/2)$ should be $T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor)$, but it turns out not to matter asymptotically.*

1.4.2 Recurrence for merge sort

$$T(n) = \begin{cases} \Theta(1) \text{ if } n = 1 \\ 2T(n/2) + \Theta(n) \text{ if } n > 1 \end{cases}$$

Note

- *We shall usually omit stating the base case when $T(n) = \Theta(1)$ for sufficiently small n, but only when it has no effect on the asymptotic solution to the recurrence.*
- *CLRS and Lecture 2 provide several ways to find a good upper bound on $T(n)$.*

1.4.3 Recursion tree

Example 1.2 Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.

1.4.4 Conclusions

- $\Theta(n \lg n)$ grows more slowly than $\Theta\left(n^2\right)$
- Therefore, merge sort asymptotically beats insertion sort in the worst case.
- In practice, merge sort beats insertion sort for $n > 30$ or so.

Chapter 2 Asymptotic Notation & Recurrences

Introduction

- $O-, \Omega-$, *and* $\Theta-$ *notation*
- *Substitution method*
- *Iterating the recurrence*
- *Recursion tree*
- *Master method*

2.1 Asymptotic notation

2.1.1 $O-$notation (upper bounds)

Definition 2.1 ($O-$**notation (upper bounds)**)

We write $f(n) = O(g(n))$ *if there exist constants* $c > 0, n_0 > 0$ *such that* $0 \leq f(n) \leq \mathrm{cg}(n)$ *for all* $n \geq n_0$ ♣

Example 2.1

$$2n^2 = O\left(n^3\right) \quad (c = 1, n_0 = 2)$$

Note *Notice that in this equation,* $2n^2$ *are functions not values and the eqal sign is just "one-way" equality. Actually, it can be denoted more precisely:*

$$2n^2 \in O\left(n^3\right)$$

Note *Convention: A set in a formula representsan anonymous function in the set.*

Example 2.2

$$n^2 + O(n) = O\left(n^2\right)$$

means for any $f(n) \in O(n)$,$n^2 + f(n) = h(n)$ for some $h(n) \in O(n^2)$

2.1.2 $\Omega-$ notation(lower bounds)

O-notation is an upper-bound notation. It makes no sense to say $f(n)$ is at least $O(n^2)$.

Definition 2.2 ($\Omega-$ **notation(lower bounds)**)

$\Omega(g(n)) = \{f(n)$: *there exist constants* $c > 0, n_0 > 0$ *such that* $0 \leq \mathrm{cg}(n) \leq f(n)$ *for all* $n \geq n_0\}$ ♣

Example 2.3

$$\sqrt{n} = \Omega(\lg n) \quad (c = 1, n_0 = 16)$$

2.1.3 $\Theta-$notation(tight bounds)

Definition 2.3 ($\Theta-$**notation(tight bounds)**)

$$\Theta(g(n)) = \mathrm{O}(g(n)) \cap \Omega(g(n))$$

♣

Example 2.4

$$\frac{1}{2}n^2 - 2n = \Theta\left(n^2\right)$$

2.1.4 $o-$notation and ω-notation

O-notation and Ω-notation are like $\leq$ and $\geq$. o-notation and ω-notation are like $<$ and $>$.

Definition 2.4 ($o-$notation)

$o(g(n)) = \{f(n)$: *for any constant* $c > 0$ *,there is a constant* $n_0 > 0$ *such that* $0 \leq f(n) < cg(n)$ *for all* $n \geq n_0\}$

♣

Example 2.5

$$2n^2 = o\left(n^3\right) \quad (n_0 = 2/c)$$

Definition 2.5 (ω-notation)

$o(g(n)) = \{f(n)$: *for any constant* $c > 0$ *, there is a constant* $n_0 > 0$ *Such that* $0 \leq f(n) < \mathrm{cg}(n)$ *for all* $n \geq n_0\}$

♣

Example 2.6

$$\sqrt{n} = \omega(\lg n) \quad (n_0 = 1 + 1/c)$$

2.2 Solving recurrences

The analysis of merge sort from Lecture 1 required us to solve a recurrence.Recurrences are like solving integrals, differential equations, etc. Learn a few tricks.

2.2.1 Substitution method

Definition 2.6 (Substitution method)

The most general method:

1. *Guess the form of the solution.*
2. *Verify by induction.*
3. *Solve for constants.*

♣

Example 2.7

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

- Assume that $T(1) = \Theta(1)$.
- Guess $O\left(n^3\right)$. (Prove O and Ω separately.)
- Assume that $T(k) \leq ck^3$ for $k < n$
- Prove $T(n) \leq cn^3$ by induction.

Solution

$$
\begin{aligned}
T(n) &= 4T(n/2) + n \\
&\leq 4c(n/2)^3 + n \\
&= (c/2)n^3 + n \\
&= cn^3 - \left((c/2)n^3 - n\right) \leftarrow \textit{ desired - residual} \\
&\leq cn^3 \longleftarrow \textit{ desired}
\end{aligned}
$$

whenever $(c/2)n^3 - n \geq 0$ *, for example,if* $c \geq 2$ *and* $n \geq 1$

Note *We must also handle the initial conditions, that is, ground the induction with base cases.*

1. **Base**: $T(n) = \Theta(1)$ for all $n < n_0$, where n_0 is a suitable constant.
2. For $1 \leq n < n_0$, we have "$\Theta(1)$" $\leq cn^3$, if we pick c big enough.

This bound is not tight!

Note *A tighter upper bound? We shall prove that* $T(n) = O\left(n^2\right)$

Assume that $T(k) \leq ck^2$ for $k < n$:

$$
\begin{aligned}
T(n) &= 4T(n/2) + n \\
&\leq 4c(n/2)^2 + n \\
&= cn^2 + n \\
&= O\left(n^2\right)
\end{aligned}
$$

Wrong! We must prove the I.H.($T(k) \leq ck^2$)

$$= cn^2 - (-n)$$

for no choice of $c > 0$. Lose!

Note *IDEA: Strengthen the inductive hypothesis.(Subtract a low-order term.)*

Inductive hypothesis: $T(k) \leq c_1k^2 - c_2k$ *for* $k < n$.

Solution

$$
\begin{aligned}
T(n) &= 4T(n/2) + n \\
&= 4\left(c_1(n/2)^2 - c_2(n/2)\right) + n \\
&= c_1n^2 - 2c_2n + n \\
&= c_1n^2 - c_2n - (c_2n - n) \\
&\leq c_1n^2 - c_2n \textit{ if } c_2 \geq 1.
\end{aligned}
$$

Pick c_1 *big enough to handle the initial conditions.*

2.2.2 Recursion-tree method

- A recursion tree models the costs(time)ofa recursive execution of an algorithm.
- The recursion-tree method can be unreliable, just like any method that uses ellipses (...)
- The recursion-tree method promotes intuition, however.
- The recursion tree method is good for generating guesses for the substitution method.

Example 2.8 Solve

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

2.2.3 The master method

The master method applies to recurrences of the form

$$T(n) = aT(n/b) + f(n)$$

where $a \geq 1, b > 1$ and f is asymptotically positive.

Theorem 2.1 (Three common cases)

Compare $f(n)$ with $n^{\log_b a}$:

1. $f(n) = O\left(n^{\log_b a-\varepsilon}\right)$ *for some constant* $\varepsilon > 0$
   - $f(n)$ *grows polynomially slower than* $n^{\log_b a}$ *(by an* n^{ε} *factor).*

   Solution:
   $$T(n) = \Theta\left(n^{\log_b a}\right)$$
2. $f(n) = \Theta\left(n^{\log_b a}\lg^k n\right)$ *for some constant* $k \geq 0$
   - $f(n)$ *and* $n^{\log_b a}$ *grow at similar rates.*

   Solution:
   $$T(n) = \Theta\left(n^{\log_b a}\lg^{k+1} n\right)$$
3. $f(n) = \Omega\left(n^{\log_b a+\varepsilon}\right)$ *for some constant* $\varepsilon > 0$
   - $f(n)$ *grows polynomially faster than* $n^{\log_b a}$ *(by an* n^{ε} *factor),and* $f(n)$ *satisfies the regularity condition that* $af(n/b) \leq cf(n)$ *for some constant* $c < 1$

   Solution:
   $$T(n) = \Theta(f(n))$$

Example 2.9

$$T(n) = 4T(n/2) + n^3$$

Solution

$$a = 4, b = 2 \Rightarrow n^{\log_b a} = n^2; f(n) = n^3$$

***CASE 3**: $f(n) = \Omega\left(n^{2+\varepsilon}\right)$ for $\varepsilon = 1$ and $4(n/2)^3 \leq cn^3$ (reg. cond.) for $c = 1/2$.*

$$\therefore T(n) = \Theta\left(n^3\right)$$

Example 2.10

$$T(n) = 4T(n/2) + n^2/\lg n$$

Solution

$$a = 4, b = 2 \Rightarrow n^{\log_b a} = n^2; f(n) = n^2/\lg n$$

***Master method does not apply.** In particular, for every constant $\varepsilon > 0$, we have $n^{\varepsilon} = \omega(\lg n)$.*

2.2.3.1 Idea of master theorem