Chapter 0 Prologue

Joe Song

August 31, 2019

Further reading: *The notes are based on Chapter 0 of Dasgupta, Papadimitriou and Vazirani. Algorithms. 2008. McGraw-Hill. New York.*

1 Data structure and algorithms

1.1 Numbers and additions

Solution 1:

Data structure: Roman numerals (none positional)

Algorithm: addition

Example: $MCDXLVIII + DCCCXII =?$

- I = 1
- IV = 4
- V = 5
- X = 10
- XL = 40
- L = 50
- C = 100
- CD = 400
- D = 500
- M = 1000

Solution 2:

Data structure: Arabic numerals (positional) invented in Indian

Algorithm: addition

Example: 1448 + 812 = ?

The word "algorithm" is coined after *Al Khwarizmi*, the author who wrote an Arabic textbook to promote the use of these numbers.

1.2 Maps and routing

Data structure: a highway map

Algorithm: routing—find the shortest path from one city to another on the map.

2 Fibonacci

$$F_n = \begin{cases} 0 & n = 0 \\ 1 & n = 1 \\ F_{n-2} + F_{n-1} & n > 1 \end{cases} \tag{1}$$

An algorithm to compute the n-th Fibonacci number:

```
function fib1(n)
  if n=0: return 0
  if n=1: return 1
  return fib1(n-1)+fib1(n-2)
```

Questions:

1. Is it correct?

   It is correct as it follows the definition of the Fibonacci number.

2. How much time does it take, as a function of n?

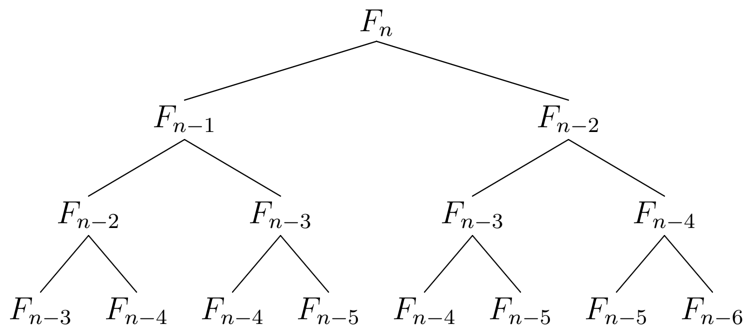
   Let $T(n)$ be the total number of operations needed to compute F_n.

   Evidently,

   $$T(n) \leq 2, \quad \text{for } n \leq 1$$

   $$T(n) = T(n-1) + T(n-2) + 3, \quad \text{for } n > 1$$

   Here 3 comes from 2 comparisons with the base cases and one addition of the F_{n-1} and F_{n-2}.



The runtime is about exponential to n—very slow if n is large.

If n is even:

$$T(n) > 2T(n-2) > 2^2T(n-4) > \cdots > 2^{n/2}T(0) = 2^{n/2} \approx 1.414^n$$

If n is odd:

$$T(n) > 2T(n-2) > 2^2T(n-4) > \cdots > 2^{(n-1)/2}T(1) = 2^{(n-1)/2} \approx 1.414^{n-1}$$

3. Can we do better? Yes.

```
function fib2(n)
  if n=0: return 0
  create an array f[0...n]
  f[0] = 0, f[1] = 1
  for i = 2 ... n:
    f[i] = f[i-1] + f[i-2]
  return f[n]
```

Run time: loop will be done $n-1$ times, each time only addition will be done. Hence the total time is linear in n.

Note: the above analysis assumed that numbers can be added in constant time. However, addition depends on the width of the numbers to be added, proportional to the value of n generally.

3 Asymptotic notations

Asymptotic notation: computer-independent characterization of an algorithm's efficiency as the input size increases.

Compare:

- Exact runtime $T(n)$

  - What is the exact time taken on an input size of n?

  - Relevant at all n

  - Difficult to predict on paper before a program is coded

  - Computer dependent

  - Improve by code optimization dependent on the programming language:

  E.g., argument passing by reference is more efficient than passing by value in C++.

- Asymptotic runtime $T(n)$ (change of $T(n)$ as n increases)

  - If n doubles, will runtime double, quadruple, or exponentially increase?

  - Relevant at large n

  - Often possible to predict on paper before a program is coded

  - Computer independent

  - Improve by efficient algorithm design independent of programming languages:

  E.g. Linear search versus binary search

3.1 Big-O notation: an asymptotic upper-bound on function growth

Example 3.1. *$5n^3 + 4n + 3$ can be reduced to $5n^3$ because the other two terms are less significant as n grows. Written as $O(n^3)$.*

Definition 3.2 (Big-O). *Let $f(n)$ and $g(n)$ be functions from positive integers to positive reals. We say $f = O(g)$ (f grows no faster than g) if there is a constant $c > 0$ such that*

$$f(n) \leq cg(n) \quad \textit{for all } n > n_0$$

Theorem 3.3. *Given $f(n), g(n) > 0$, $f(n) = O(g(n))$ if and only if*

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = c \quad (0 \leq c < \infty)$$

Note: c can be zero, but not infinity ∞ here.

Example 3.4.

$$k(n) = 2n + 20, \quad p(n) = n^2$$

By definition: *When $n \geq 6$, $k < p$. Therefore $c = 1$ and $n_0 = 6$, we have*

$$k(n) \leq cp(n) \quad n \geq n_0$$

which implies

$$k(n) = O(p(n))$$

By limit:

$$\lim_{n\to\infty} \frac{k(n)}{p(n)} = \lim_{n\to\infty} \frac{2n+20}{n^2} = 0$$

Therefore $k = O(p)$.

Example 3.5. *$k(n)$ versus*

$$h(n) = n + 1$$

By taking the limit, we have

$$\lim_{n\to\infty} \frac{k(n)}{h(n)} = \lim_{n\to\infty} \frac{2n+20}{n+1} = 2$$

implying that

$$k = O(h)$$

By taking the limit, we have

$$\lim_{n\to\infty} \frac{h(n)}{k(n)} = \lim_{n\to\infty} \frac{n+1}{2n+20} = \frac{1}{2}$$

implying that

$$h = O(k)$$

Example 3.6. $\ln n = O(n)$.

By limit

$$\lim_{n\to\infty} \frac{\ln n}{n} = \lim_{n\to\infty} \frac{1/n}{1} = \lim_{n\to\infty} \frac{1}{n} = 0$$

The second term is the consequence of applying the L'Hôpital's rule.

Therefore, we have $\ln n = O(n)$, *or*

Logarithm functions grow slower than linear functions.

3.2 Big-Ω defines a lower-bound on function growth

We define $g(n)$ to be a lower bound of $f(n)$, written as $f = \Omega(g)$, if and only if $g = O(f)$ (f is an upper bound of g).

Example 3.7. *As we have already established* $k = O(p)$, *it follows that* $p = \Omega(k)$, *or* $2n + 20$ *is a lower bound of* n^2.

Theorem 3.8. *Given* $f(n), g(n) > 0$, $f(n) = \Omega(g(n))$ *if and only if*

$$\lim_{n\to\infty} \frac{g(n)}{f(n)} = c \quad (0 \leq c < \infty)$$

Note: c can be zero, but not infinity here.

Example 3.9. $2^n = \Omega(n)$.

By limit

$$\lim_{n\to\infty} \frac{n}{2^n} = \lim_{n\to\infty} \frac{1}{2^n \ln 2} = 0$$

The second term is the consequence of applying the L'Hôpital's rule.

Therefore, we have $2^n = \Omega(n)$, *implying*

Exponential functions grow faster than linear functions.

3.3 Big-Θ defines the tight bound on function growth

We define $g(n)$ to be a tight bound of $f(n)$, written as $f = \Theta(g)$, if and only if $f = O(g)$ and $f = \Omega(g)$.

Example 3.10. *As we have already established $k = O(h)$ and $h = O(k)$, we have $k = \Theta(h)$ and also $h = \Theta(k)$.*

Commonsense rules: *(to quickly guess the bound or order of a function)*

1. multiplicative constant: $14n^2$ becomes n^2
2. n^a dominates n^b if $a > b$: n^2 dominates n
3. Exponential dominates polynomial: 3^n dominates n^{100}
4. Polynomial dominates logarithm: $n^{0.5}$ dominates $(\log n)^3$