Analysis Tools for Data Structures and Algorithms

Hsuan-Tien Lin

Dept. of CSIE, NTU

March 16, 2021

What We Have Done

reminder: use forum more; tag your emails properly

- linked list: 'clue mission game' in memory
- trading space (next, prev pointers) for flexibility (memory allocation, operations)
- application: polynomial, sparse vector
- rough notation for complexity

Motivation

Rough Time Complexity of Matrix Addition

$$P \cdot rows \cdot cols + (Q + S) \cdot rows + T$$

P, Q, R, S, T hard to keep track and not matter much

MATRIX-ADD($A, B, rows, cols$)

```
C = CONSTRUCT-MATRIX(row, col)
for i = 1 to rows
        for j = 1 to cols
                C[i, j] = A[i, j] + B[i, j]
return C
```

- inner for: $R = P \cdot cols + Q = \text{rough}(cols)$
- total: $(S + R) \cdot rows + T = \text{rough}(\text{rough}(cols) \cdot rows)$

rough time needed: $\text{rough}(rows \cdot cols)$

Asymptotic Notation

Representing "Rough" by Asymptotic Notation

- goal: rough rather than exact steps
- why rough? constant not matter much

—when input size n large

compare two complexity functions $f(n)$ and $g(n)$

growth of functions matters
—when n large, n^3 eventually bigger than $1126n$

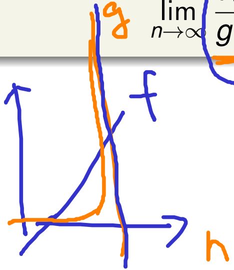
rough $\Leftrightarrow$ asymptotic behavior

Asymptotic Notations: Rough Upper Bound

big-O: rough upper bound

- $f(n)$ grows slower than or similar to $g(n)$: $f(n) = O(g(n))$
  - n grows slower than n^2: $n = O(n^2)$
  - $3n$ grows similar to n: $3n = O(n)$
- asymptotic intuition (rigorous math later):

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



big-O: arguably the most used "language" for complexity

More Intuitions on Big-O

$$f(n) = O(g(n)) \Leftarrow \lim_{n\to\infty} \frac{f(n)}{g(n)} \le c \quad \text{(not rigorously, yet)}$$

- "$= O(\cdot)$" more like "$\in$"
  - $n = O(n)$
  - $n = O(10n)$
  - $n = O(0.3n)$
  - $n = O(n^5)$
- "$= O(\cdot)$" also like "$\le$"
  - $n = O(n^2)$
  - $n^2 = O(n^{2.5})$
  - $n = O(n^{2.5})$
- $1126n = O(n)$: coefficient not matter
- $n + \sqrt{n} + \log n = O(n)$: lower-order term not matter

$n^3 = O(n^3)$

$n^2 = O(n^3)$

$n^{2.5} = O(n^3)$

intuitions (properties) to be proved later

Formal Definition of Big-O

positive

Consider positive functions $f(n)$ and $g(n)$ on integers n,

$f(n) = O(g(n))$, iff exist *positive* c, $\boxed{n_0}$ such that $\boxed{f(n) \le c \cdot g(n)}$ for all $n \ge n_0$

夠大

- covers the lim intuition if limit exists
- covers other situations without "limit"
  e.g. $|\sin(n)| = O(1)$

next: prove that lim intuition ⇒ formal definition

$f(n) \le c \cdot g(n)$

lim Intuition $\Rightarrow$ Formal Definition

性質

(c', n_0')

For positive functions f and g, if $\lim_{n\to\infty} \frac{f(n)}{g(n)} \le c$, then $f(n) = O(g(n))$.

定義

- with definition of limit, there exists ϵ, n_0 such that for all $n \ge n_0$, $|\frac{f(n)}{g(n)} - c| < \epsilon$.
- That is, for all $n \ge n_0$, $\frac{f(n)}{g(n)} < \boxed{c + \epsilon}$.
- Let $c' = c + \epsilon$, $\boxed{n_0' = n_0}$, big-O definition satisfied with (c', n_0'). QED.

important to not just have intuition (building),
but know definition (building block)

for all $n \ge n_0'$, $f(n) \le c' \cdot g(n)$

More on Asymptotic Notations

Asymptotic Notations: Definitions

- $f(n)$ grows slower than or similar to $g(n)$: ("$\leq$")

  $f(n) = O(g(n))$, iff exist c, n_0 such that $f(n) \leq c \cdot g(n)$ for all $n \geq n_0$

- $f(n)$ grows faster than or similar to $g(n)$: ("$\geq$")

  $f(n) = \Omega(g(n))$, iff exist c, n_0 such that $f(n) \geq c \cdot g(n)$ for all $n \geq n_0$

- $f(n)$ grows similar to $g(n)$: ("$\approx$")

  $f(n) = \Theta(g(n))$, iff $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$

- also $o(\cdot)$ and $\omega(\cdot)$

let's see how to use them

The Seven Functions as g

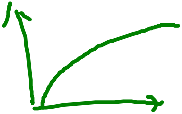
$g(n) = ?$

- 1: constant
- $\log n$: logarithmic (does base matter?)
- n: linear
- $n \log n$
- n^2: square
- n^3: cubic
- 2^n: exponential (does base matter?)

$\log_2$

$\ln$

$\log_{10}$



will often encounter them in future classes

Analysis of Sequential Search

```
SEQUENTIAL-SEARCH(A, key)
1  for i = 1 to A.length
2      if A[i] = key
3          return i
4  return FAIL
```

- best case (e.g. *key* at 0): time $\Theta(1)$
- average case with respect to uniform $key \in A$: time $\Theta(n)$
- worst case (e.g. *key* at last or not found): time $\Theta(n)$

often just say $O(n)$-algorithm (linear complexity)

Analysis of Binary Search

BINARY-SEARCH(A, key)

```
left = 1, right = A.length
while left ≤ right
        mid = floor((left+right)/2)
        if A[mid] = key
                return mid
        elseif A[mid] < key
                left = mid + 1
        elseif A[mid] > key
                right = mid − 1
return FAIL
```

- best case (e.g. *key* at first *mid*): time $\Theta(1)$
- worst case (e.g. *key* not found):
  because range ($right - left$) halved in each WHILE, needs time $\Theta(\log n)$ iterations to decrease range to 0

lg n

often just say $O(\log n)$-algorithm (logarithmic complexity)

Sequential and Binary Search

- Input: any integer array A with size n, an integer *key*
- Output: if *key* not within *list*, SUCCEED; otherwise, FAIL

```
DIRECT-SEQ-SEARCH(A, key)

    if SEQ-SEARCH(A, key) = FAIL
        return FAIL
    else
        return SUCCEED
```

```
SORT-AND-BINSEARCH(A, key)
    B = SEL-SORT(A)
    if BIN-SEARCH(A, key) = FAIL
        return FAIL
    else
        return SUCCEED
```

- DIRECT-SEQ-SEARCH: $O(n)$ time
- SORT-AND-BIN-SEARCH: $O(n^2)$ time for SEL-SORT and $O(\log n)$ time for BIN-SEARCH

next: operations for "combining" asymptotic complexity

Properties of Asymptotic Notations

Some Properties of Big-O I

Theorem (封閉律)

if $f_1(n) = O(g_2(n))$, $f_2(n) = O(g_2(n))$ *then* $f_1(n) + f_2(n) = O(g_2(n))$

(c_1, n_1)

① $f_1 \le c_1 \cdot g_2$

$\forall n \geq n_1$

(c_2, n_2)

② $f_2 \le c_2 \cdot g_2$

$\forall n \geq n_2$

(c, n_0)

$f_1 + f_2 \le (c_1 + c_2) g_2$

$f_1 + f_2 \le c \cdot g_2$

$\forall n \geq n_0$

$\max(n_1, n_2)$

Some Properties of Big-O II

Theorem（ 併吞律 ）

if $f_1(n) = O(g_1(n))$, $f_2(n) = O(g_2(n))$ *and* $g_1(n) = O(g_2(n))$ *then* $f_1(n) + f_2(n) = O(g_2(n))$

~~*Proof: use two theorems above.*~~

Theorem

If $f(n) = a_m n^m + \cdots + a_1 n + a_0$, *then* $f(n) = O(n^m)$

Proof: use the theorem above.

similar proof for Ω and Θ

Some More on Big-O

BINARY-SEARCH is $O(\log n)$ time

- by 遞移律, time also $O(n)$
- time also $O(n \log n)$
- time also $O(n^2)$
- also $O(2^n)$
- ...

prefer the tightest Big-O!

Comparison of Complexity

	(consecutive) array	linked list
index access	$O(1)$	$O(n)$
head insertion	$O(n)$	$O(1)$
tail insertion	$O(1)$	$O(1)$ after getting `tail`
'middle' insertion	$O(n)$	$O(1)$ after getting `after`
wasted space	$O(1)$ for `len`	$O(n)$ for `next`

exercise: how about dynamic array (double after full)?

Practical Complexity

some input sizes are time-wise infeasible for some algorithms

when 1-billion-steps-per-second

n	n	$n\log_2 n$	n^2	n^3	n^4	n^{10}	2^n
10	$0.01\mu s$	$0.03\mu s$	$0.1\mu s$	$1\mu s$	$10\mu s$	$10s$	$1\mu s$
20	$0.02\mu s$	$0.09\mu s$	$0.4\mu s$	$8\mu s$	$160\mu s$	$2.84h$	$1ms$
30	$0.03\mu s$	$0.15\mu s$	$0.9\mu s$	$27\mu s$	$810\mu s$	$6.83d$	$1s$
40	$0.04\mu s$	$0.21\mu s$	$1.6\mu s$	$64\mu s$	$2.56ms$	$121d$	$18m$
50	$0.05\mu s$	$0.28\mu s$	$2.5\mu s$	$125\mu s$	$6.25ms$	$3.1y$	$13d$
100	$0.10\mu s$	$0.66\mu s$	$10\mu s$	$1ms$	$100ms$	$3171y$	$4 \cdot 10^{13}y$
10^3	$1\mu s$	$9.96\mu s$	$1ms$	$1s$	$16.67m$	$3 \cdot 10^{13}y$	$3 \cdot 10^{284}y$
10^4	$10\mu s$	$130\mu s$	$100ms$	$1000s$	$115.7d$	$3 \cdot 10^{23}y$	
10^5	$100\mu s$	$1.66ms$	$10s$	$11.57d$	$3171y$	$3 \cdot 10^{33}y$	
10^6	$1ms$	$19.92ms$	$16.67m$	$32y$	$3 \cdot 10^7y$	$3 \cdot 10^{43}y$	

note: similar for space complexity,
e.g. store an N by N double matrix when $N = 50000$?