Analysis Tools for Data Structures and Algorithms

Hsuan-Tien Lin

Dept. of CSIE, NTU

March 24, 2020

Motivation

Properties of Good Programs

- meet requirements, correctness: basic
- clear usage document (external), readability (internal), etc.

Resource Usage (Performance)

- efficient use of computation resources (CPU, FPU, GPU, etc.)?
  time complexity
- efficient use of storage resources (memory, disk, etc.)?
  space complexity

need: “language” for describing the complexity

Space Complexity of List Summing

LIST-SUM(float array *list*, integer length *n*)

```
total ← 0
for i ← 0 to n − 1 do
    total ← total + list[i]
end for
return total
```

- array *list*: size of pointer, often 8
- integer *n*: often 4
- float *total*: 4
- integer *i*: commonly 4
- float return place: 4

total space 24 (constant) *within algorithm execution*
does not depend on *n*

Space Complexity of Recursive List Summing

RECURSIVE-LIST-SUM(float array *list*, integer length *n*)

if $n = 0$
  return 0
else
  return $list[n]+$ RECURSIVE-LIST-SUM$(list, n-1)$
end if

- array *list*: size of pointer, often 8
- integer *n*: often 4
- float return place: 4

only 16, better than previous one?

Time Complexity of Matrix Addition

MATRIX-ADD
(integer matrix a, b, result integer matrix c, integer $rows$, $cols$)

```
for i ← 0 to rows − 1 do
    for j ← 0 to cols − 1 do
        c[i][j] ← a[i][j] + b[i][j]
    end for
end for
```

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

total time needed: $P \cdot rows \cdot cols + (Q + S) \cdot rows + T$

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
(integer matrix a, b, result integer matrix c, integer $rows$, $cols$)

```
for i ← 0 to rows − 1 do
    for j ← 0 to cols − 1 do
        c[i][j] ← a[i][j] + b[i][j]
    end for
end for
```

- 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 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

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

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

intuitions (properties) to be proved later

Formal Definition of Big-O

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

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

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

next: prove that lim intuition $\Rightarrow$ formal definition

lim Intuition $\Rightarrow$ Formal Definition

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

- with definition of limit, there exists ϵ, n_0 such that for all $n \geq n_0$, $|\frac{f(n)}{g(n)} - c| < \epsilon$.
- That is, for all $n \geq n_0$, $\frac{f(n)}{g(n)} < c + \epsilon$.
- Let $c' = c + \epsilon$, $n'_0 = n_0$, big-O definition satisfied with (c', n'_0). QED.

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

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 $f(n) = \Omega(g(n))$

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?)

will often encounter them in future classes

Analysis of Sequential Search

Sequential Search

```
for i ← 0 to n − 1 do
    if list[i] == num
        return i
    end if
end for
return −1
```

- best case (e.g. *num* at 0): time $\Theta(1)$
- worst case (e.g. *num* at last or not found): time $\Theta(n)$

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

Analysis of Binary Search

Binary Search

$left \leftarrow 0$, $right \leftarrow n-1$
while $left \leq right$ **do**
  $mid \leftarrow \text{floor}((left + right)/2)$
  if $list[mid] > num$
    $left \leftarrow mid + 1$
  else if $list[mid] < num$
    $right \leftarrow mid - 1$
  else
    return mid
  end if
end while
return -1

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

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

Sequential and Binary Search

- Input: any integer array *list* with size n, an integer *num*
- Output: if *num* not within *list*, -1; otherwise, $+1126$

DIRECT-SEQ-SEARCH (*list*, n, *num*)

```
for i ← 0 to n − 1 do
    if list[i] == num
        return +1126
    end if
end for
return −1
```

SORT-AND-BIN-SEARCH (*list*, n, *num*)

```
SEL-SORT(list, n)
return
BIN-SEARCH(list, n, num) ≥ 0? + 1126 : −1
```

- 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* $f_1(n) + f_2(n) = O(g_2(n))$

- When $n \geq n_1$, $f_1(n) \leq c_1 g_2(n)$
- When $n \geq n_2$, $f_2(n) \leq c_2 g_2(n)$
- So, when $n \geq \max(n_1, n_2)$, $f_1(n) + f_2(n) \leq (c_1 + c_2) g_2(n)$

Theorem (遞移律)

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

- When $n \geq n_1$, $f_1(n) \leq c_1 g_1(n)$
- When $n \geq n_2$, $g_1(n) \leq c_2 g_2(n)$
- So, when $n \geq \max(n_1, n_2)$, $f_1(n) \leq c_1 c_2 g_2(n)$

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

RECURSIVE-BIN-SEARCH is $O(\log n)$ time and $O(\log n)$ space

- 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!

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 $N = 50000$?