Analysis Tools

Hsuan-Tien Lin

Dept. of CSIE, NTU

March 14, 2013

Asymptotic Notations: Symbols

<=

- $f(n)$ grows slower than or similar to $g(n)$: $f(n) = O(g(n))$
- $f(n)$ grows faster than or similar to $g(n)$: $f(n) = \Omega(g(n))$ >=
- $f(n)$ grows similar to $g(n)$: $f(n) = \Theta(g(n))$ \sim

- $n = O(n)$; $n = O(10n)$; $n = O(0.3n)$; $n = O(n^2)$; $n = O(n^5)$; $\cdots$ (note: $=$ more like "$\in$")
- $n = \Omega(n)$; $n = \Omega(0.2n)$; $n = \Omega(5n)$; $n = \Omega(\log n)$; $n = \Omega(\sqrt{n})$; $\cdots$
- $n = \Theta(n)$; $n = \Theta(0.1n + 4)$; $n = \Theta(7n)$; $n \neq \Theta(5^n)$

Asymptotic Notations: Definitions

$n = O(n^2)$

- $f(n)$ grows slower than or similar to $g(n)$: $n \leq 2n^2$

  > 0

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

  $2 \quad 1 \quad n \quad n^2$

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

  $f(n) = \Omega(g(n))$, 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 $g(n)$: c_1, c_2, n_0

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

$n^2 \neq O(n)$ $\qquad c_2 g(n) \leq f(n) \leq c_1 g(n)$

Basic Algorithms: Sequential and Binary Search

- Input: a **sorted** integer array *list* with size *n*, an integer *searchnum*
- Output: if *searchnum* is within *list*, its index; otherwise -1

SEQ-SEARCH (*list*, *n*, *searchnum*)

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

BIN-SEARCH (*list*, *n*, *searchnum*)

```
left ← 0, right ← n − 1
while left ≤ right do
    middle ← floor((left + right)/2)
    if list[middle] > searchnum
        right ← middle − 1
    else if list[middle] < searchnum
        left ← middle + 1
    else        /* list[middle] == searchnum */
        return middle
    end if
end while
return −1
```

SEQ-SEARCH(*list*, *n*, *searchnum*)

for $i \leftarrow 0$ to $n-1$ **do**
  if $list[i] == searchnum$
    return i
  end if
end for
return -1

$l[0]$	$l[1]$	$l[2]$	$l[3]$	$l[4]$	$l[5]$	$l[6]$
1	3	4	9	9	10	13

- search for 9

3

- search for 15 (worst case?)

-1

Binary Search: Eliminate at Least Half Each Time

BIN-SEARCH(*list*, *n*, *searchnum*)

```
left ← 0, right ← n − 1
while left ≤ right do
    middle ← floor((left + right)/2)
    if list[middle] > searchnum
        right ← middle − 1
    else if
    list[middle] < searchnum
        left ← middle + 1
    else
        return middle
    end if
end while
return −1
```

$l[0]$	$l[1]$	$l[2]$	$l[3]$	$l[4]$	$l[5]$	$l[6]$
1	3	4	9	9	10	13

- search for 9

$(l,r) = (0,6)$ $m = 3$

$l[m] == s.n.$

$\to 3$

- search for 15 (worst case?)

$(l,r) = (0,6)$ $m = 3$ ×

$(4,6)$ $= 5$ ×

$(6,6)$ $= 6$ ×

$(7,6)$ $\to -1$

Analysis of Sequential Search

Sequential Search

for $i \leftarrow 0$ to $n-1$ **do**
  if $list[i] == searchnum$
    return i
  end if
end for
return -1

- best case (e.g. *searchnum* at 0): time $\Theta(1)$
- worst case (e.g. *searchnum* at last or not found): time $\Theta(n)$
- in general: time $\Omega(1)$ and $O(n)$

Analysis of Binary Search

Binary Search

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

- best case (e.g. *searchnum* at *middle*): time $\Theta(1)$
- worst case (e.g. *searchnum* not found):
  because (*right* − *left*) is halved in each WHILE iteration, needs time $\Theta(\log n)$ iterations if not found
- in general:
  time $\Omega(1)$ and $O(\log n)$

often care about the worst case (and thus see $O(\cdot)$ often)

Sequential and Binary Search

- Input: any integer array *list* with size n, an integer *searchnum*
- Output: if *searchnum* is not within *list*, -1; otherwise, othernum

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

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

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

```
SEL-SORT(list, n)
return BIN-SEARCH(list, n, searchnum)
```

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

want: show asymptotic complexity of SORT-AND-BIN-SEARCH as its bottleneck

Some Properties of Big-Oh I

$n^3 = O(n^2)$ $\log n = O(n^2)$ → $O(n^2)$

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, n_0

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

n_0 c

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-Oh 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-Oh

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-Oh!

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	$1ms$	$19.92ms$	$16.67m$	$32y$	$3 \cdot 10^7y$	$3 \cdot 10^{43}y$	

10^9

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