Data Structures and Algorithms
(資料結構與演算法)

Lecture 3: Analysis Tools

Hsuan-Tien Lin (林軒田)

htlin@csie.ntu.edu.tw

Department of Computer Science
& Information Engineering

National Taiwan University
(國立台灣大學資訊工程系)

Roadmap

1. the one where it all began

Lecture 2: Data Structures

scheme of purposefully **organizing** data with **access/maintenance** algorithms, such as **ordered array** for **faster search**

Lecture 3: Analysis Tools

- motivation
- cases of complexity analysis
- asymptotic notation
- usage of asymptotic notation

2. the data structures awaken
3. fantastic trees and where to find them
4. the search revolutions
5. sorting: the final frontier

motivation

Recall: Properties of Good Program

good program: proper use of resources

Space Resources

- memory
- disk(s)
- transmission bandwidth

—space complexity

Computation Resources

- CPU(s)
- GPU(s)
- computation power

—time complexity

need: language for describing complexity

Space Complexity of GET-MIN

```
GET-MIN(A)
m = 1 // store current min. index
for i = 2 to A.length
    // update if i-th element smaller
    if A[m] > A[i]
        m = i
return A[m]
```

- array A: pointer size s_1 (not counting the actual input elements)
- integer m: size s_2
- integer i: size s_2

total space $s_1 + 2s_2$:
constant to $n = A.length$ within algorithm

Space Complexity of GET-MIN-WASTE

GET-MIN-WASTE(A)

```
B = COPY(A, 1, A.length)
INSERTION-SORT(B)
return B[1]
```

- array A: pointer size s_1 (not counting the actual input elements)
- array B:
  - pointer size s_1
  - n integers with total size $s_2 \cdot n$, where $n = A.length$
- any space that INSERTION-SORT uses: $\square$

total space $2s_1 + s_2 n + \square$:
(at least) linear to n within algorithm

Time Complexity of Insertion Sort

INSERTION-SORT(A)	cost	times
for $m = 2$ **to** $A.length$	d_1	n
$key = A[m]$	d_2	$n-1$
// insert $A[m]$ into the sorted sequence $A[1\,..\,m-1]$	0	$n-1$
$i = m-1$	d_4	$n-1$
while $i > 0$ and $A[i] > key$	d_5	$\sum_{m=2}^{n} t_m$
$A[i+1] = A[i]$	d_6	$\sum_{m=2}^{n}(t_m - 1)$
$i = i - 1$	d_7	$\sum_{m=2}^{n}(t_m - 1)$
$A[i+1] = key$	d_8	$n-1$

(from Introduction to Algorithms Third Edition, Cormen at al.)

total time $T(n)$

$$= \quad d_1 n + d_2(n-1) + d_4(n-1) + d_5 \sum_{m=2}^{n} t_m + d_6 \sum_{m=2}^{n}(t_m - 1) + d_7 \sum_{m=2}^{n}(t_m - 1) + d_8(n-1)$$

actual time $d_\bullet$ depends on machine type;
total $T(n)$ depends on n and t_m, number of **while** checks

Fun Time

Consider running GET-MIN on an array A of length n. If line i takes a time cost of d_i, and the inequality in line 4 is TRUE for t times, what is the time complexity of GET-MIN?

```
GET-MIN(A)
1  m = 1 // store current min. index
2  for i = 2 to A.length
3      // update if i-th element smaller
4      if A[m] > A[i]
5          m = i
6  return A[m]
```

1. $d_1 + d_2 + d_4 + d_5 + d_6$
2. $d_1 + td_2 + td_4 + td_5 + d_6$
3. $d_1 + nd_2 + td_4 + td_5 + d_6$
4. $d_1 + nd_2 + (n-1)d_4 + td_5 + d_6$

Fun Time

Consider running GET-MIN on an array A of length n. If line i takes a time cost of d_i, and the inequality in line 4 is TRUE for t times, what is the time complexity of GET-MIN?

```
GET-MIN(A)
1  m = 1 // store current min. index
2  for i = 2 to A.length
3        // update if i-th element smaller
4        if A[m] > A[i]
5              m = i
6  return A[m]
```

1. $d_1 + d_2 + d_4 + d_5 + d_6$
2. $d_1 + td_2 + td_4 + td_5 + d_6$
3. $d_1 + nd_2 + td_4 + td_5 + d_6$
4. $d_1 + nd_2 + (n-1)d_4 + td_5 + d_6$

Reference Answer: 4

The loop (including ending check) in line 2 is run n times; the condition in line 4 is checked $n-1$ times, and t of those result in execution of line 5.

cases of complexity analysis

Best-case Time Complexity of Insertion Sort

INSERTION-SORT(A)	cost	times
for $m = 2$ **to** $A.length$	d_1	n
$\quad key = A[m]$	d_2	$n-1$
$\quad$ // insert $A[m]$ into the sorted sequence $A[1\,..\,m-1]$	0	$n-1$
$\quad i = m-1$	d_4	$n-1$
$\quad$ **while** $i > 0$ and $A[i] > key$	d_5	$\sum_{m=2}^{n} t_m$
$\quad\quad A[i+1] = A[i]$	d_6	$\sum_{m=2}^{n}(t_m - 1)$
$\quad\quad i = i-1$	d_7	$\sum_{m=2}^{n}(t_m - 1)$
$\quad A[i+1] = key$	d_8	$n-1$

(from Introduction to Algorithms Third Edition, Cormen at al.)

sorted $A \Longrightarrow t_m = 1$

$$
\begin{aligned}
& T(n) \\
= \quad & d_1 n + d_2(n-1) + d_4(n-1) + d_5 \sum_{m=2}^{n} t_m + d_6 \sum_{m=2}^{n}(t_m - 1) + d_7 \sum_{m=2}^{n}(t_m - 1) + d_8(n-1) \\
= \quad & d_1 n + d_2(n-1) + d_4(n-1) + d_5(n-1) + d_6(0) + d_7(0) + d_8(n-1)
\end{aligned}
$$

best case: $T(n) = \blacksquare \cdot n + \blacklozenge$ (linear to n)

Worst-case Time Complexity of Insertion Sort

INSERTION-SORT(A)	cost	times
for $m = 2$ **to** $A.length$	d_1	n
$key = A[m]$	d_2	$n-1$
// insert $A[m]$ into the sorted sequence $A[1\,..\,m-1]$	0	$n-1$
$i = m-1$	d_4	$n-1$
while $i > 0$ and $A[i] > key$	d_5	$\sum_{m=2}^{n} t_m$
$A[i+1] = A[i]$	d_6	$\sum_{m=2}^{n}(t_m-1)$
$i = i-1$	d_7	$\sum_{m=2}^{n}(t_m-1)$
$A[i+1] = key$	d_8	$n-1$

(from Introduction to Algorithms Third Edition, Cormen at al.)

reverse-sorted $A \Longrightarrow t_m = m$

$$
\begin{aligned}
& T(n) \\
= \quad & d_1 n + d_2(n-1) + d_4(n-1) + d_5 \sum_{m=2}^{n} t_m + d_6 \sum_{m=2}^{n}(t_m-1) + d_7 \sum_{m=2}^{n}(t_m-1) + d_8(n-1) \\
= \quad & d_1 n + d_2(n-1) + d_4(n-1) + d_5\left(\tfrac{(n+2)(n-1)}{2}\right) + d_6\left(\tfrac{n(n-1)}{2}\right) + d_7\left(\tfrac{n(n-1)}{2}\right) + d_8(n-1)
\end{aligned}
$$

worst case: $T(n) = \bigstar \cdot n^2 + \blacksquare \cdot n + \blacklozenge$ (quadratic to n)

Average-case Time Complexity of Insertion Sort

average case

other cases

$A = [1, 2, 4, 3]$

best cases

$A = [1, 2, 3, 4]$

other cases

$A = [1, 4, 2, 3]$

worst cases

$A = [4, 3, 2, 1]$

$\ldots$

other cases

$A = [4, 3, 1, 2]$

best case $\leq$ average case $\leq$ worst case

Time Complexity Analysis in Pratice

Common Focus

worst-case time complexity

- physically meaningful: waiting time/power consumption
- often ≈ average-case: when many near-worst-cases

Common Language

rough time needed

w.r.t. input size n

$$T(n) = \bigstar \cdot n^2 + \blacksquare \cdot n + \blacklozenge$$

- care more about
  - larger n
  - leading term of n
- care less about
  - constants
  - other terms of n

next: language of rough notation

Fun Time

Which of the following describes the best-case time complexity of GET-MIN on an array A of length n?

```
GET-MIN(A)
m = 1 // store current min. index
for i = 2 to A.length
    // update if i-th element smaller
    if A[m] > A[i]
        m = i
return A[m]
```

1. constant to n
2. linear to n
3. quadratic to n
4. none of the other choices

Fun Time

Which of the following describes the best-case time complexity of GET-MIN on an array A of length n?

```
GET-MIN(A)
1  m = 1 // store current min. index
2  for i = 2 to A.length
3      // update if i-th element smaller
4      if A[m] > A[i]
5          m = i
6  return A[m]
```

1. constant to n
2. linear to n
3. quadratic to n
4. none of the other choices

Reference Answer: 2

Even in the best case, where line 5 is executed 0 times, the loop (including ending check) in line 2 still needs to be run n times, and the condition in line 4 still needs to be checked $n - 1$ times.

asymptotic notation

‘Rough’ Notation

goal

$$\bigstar \cdot n^2 + \blacksquare \cdot n + \blacklozenge \overset{\text{roughly}}{\sim} n^2$$

- care more about
  - larger n
  - leading term of n
- care less about
  - constants
  - other terms of n

notation

$$\underbrace{\bigstar \cdot n^2 + \blacksquare \cdot n + \blacklozenge}_{f(n)} = \Theta(\underbrace{n^2}_{g(n)})$$

for positive $f(n)$ and $g(n)$ [when $n \in \mathbb{R}$ with $n \geq 1$]

extracting the similarity: consider $\frac{f(n)}{g(n)}$

Modeling Rough with Asymptotic Behavior

goal

$$\underbrace{\bigstar \cdot n^2 + \blacksquare \cdot n + \blacklozenge}_{f(n)} = \Theta(\underbrace{n^2}_{g(n)})$$

- growth of $\blacksquare \cdot n + \blacklozenge$ slower than $g(n) = n^2$: for large n, removable by dividing $g(n)$
- asymptotically, two functions only differ by $c > 0$

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

—why needing $c > 0$?

'rough' definition version 0 (to be changed):
for positive $f(n)$ and $g(n)$,
$f(n) = \Theta(g(n))$ if $\lim_{n\to\infty} \frac{f(n)}{g(n)} = c > 0$

Asymptotic Notation: Modeling Rough Growth

$$f(n) = \Theta(g(n)) \Longleftarrow \lim_{n\to\infty} \frac{f(n)}{g(n)} = c > 0$$

big-Θ: roughly the same

- definition meets criteria:
  - care about larger n: yes, $n \to \infty$
  - leading term more important: yes, $n + \sqrt{n} + \log n = \Theta(n)$
  - insensitive to constants: yes, $1126n = \Theta(n)$
- meaning: $f(n)$ grows roughly the same as $g(n)$
- "$= \Theta(\cdot)$" actually "$\in$"

	$\sqrt{n}$	$0.1126n$	n	$112.6n$	$n^{1.1}$	$\exp(n)$
$\Theta(n)$?	N	Y	Y	Y	N	N

asymptotic notation:
the most used 'language' for time/space complexity

Issue about the Convergence Definition

$$f(n) = \Theta(g(n)) \Longleftarrow \lim_{n\to\infty} \frac{f(n)}{g(n)} = c > 0$$

consider a hypothetical algorithm:

- $T(n) = n$ for even n
- $T(n) = 2n$ for odd n

— want: $T(n) = \Theta(n)$, but $\lim_{n\to\infty} \frac{T(n)}{n}$ does not exist!

fix (formal): for asymptotically non-negative f(n) & g(n)

$$f(n) = \Theta(g(n)) \quad \Longleftrightarrow \quad \text{exists positive } (n_0, c_1, c_2) \text{ such that } c_1 g(n) \le f(n) \le c_2 g(n) \text{ for } n \ge n_0$$

Convergence 'Definition' $\Rightarrow$ Formal Definition

For asymptotically non-negative functions $f(n)$ and $g(n)$, if $\lim_{n\to\infty} \frac{f(n)}{g(n)} = c$, then $f(n) = \Theta(g(n))$.

- with definition of limit, there exists $\epsilon > 0, n_0 > 0$ such that for all $n \geq n_0$, $|\frac{f(n)}{g(n)} - c| < \epsilon$.
- i.e. for all $n \geq n_0$, $c - \epsilon < \frac{f(n)}{g(n)} < c + \epsilon$.
- Let $c_1' = c - \epsilon$, $c_2' = c + \epsilon$, $n_0' = n_0$, formal definition satisfied with (c_1', c_2', n_0'). QED

often suffices to use convergence 'definition' in practice

usage of asymptotic notation

The Seven Functions as $g(n)$

$g(n) = ?$

- 1: constant
  —meaning $c_1 \le f(n) \le c_2$ for $n \ge n_0$
- $\log n$: 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

Logarithmic Function in Asymptotic Notation

Claim

For any $a > 1$, $b > 1$, if $f(n) = \Theta(\log_a n)$, then $f(n) = \Theta(\log_b n)$.

Proof

- $f(n) = \Theta(\log_a n) \iff \exists(c_1, c_2, n_0)$ such that $c_1 \log_a n \le f(n) \le c_2 \log_a n$ for $n \ge n_0$
- Then, $c_1 \log_a b \log_b n \le f(n) \le c_2 \log_a b \log_b n$ for $n \ge n_0$
- Let $c_1' = c_1 \log_a b$, $c_2' = c_2 \log_a b$, $n_0' = n_0$, we get $f(n) = \Theta(\log_b n)$

base does not matter in $\Theta(\log n)$

Analysis of Sequential Search

```
SEQ-SEARCH(A, key)
for i = 1 to A.length
        // return when found
        if A[i] equals key
                return i
return NIL
```

- best case (i.e. *key* at 1): $T(n) = \Theta(1)$
- worst case (i.e. return NIL): $T(n) = \Theta(n)$
- average case with respect to uniform *key* $\in A$: $\mathbb{E}(T(n)) = \Theta(n)$

\# iterations in loop: dominating often

Analysis of Binary Search

```
BIN-SEARCH(A, key, ℓ, r)
while ℓ ≤ r
    m = floor((ℓ + r)/2)
    if A[m] equals key
        return m
    elseif A[m] > key
        r = m − 1 // cut out end
    elseif A[m] < key
        ℓ = m + 1 // cut out begin
return NIL
```

- best case (i.e. *key* at first m): $T(n) = \Theta(1)$
- worst case (i.e. return NIL): because range $(r - \ell + 1)$ roughly halved in each **while** , # iterations roughly $\log_2 n$: $T(n) = \Theta(\log n)$

often care more about worst case, as mentioned

Summary

Lecture 3: Analysis Tools

- motivation

  roughly quantify time or space complexity to measure efficiency
- cases of complexity analysis

  often focus on worst-case with ‘rough’ notations
- asymptotic notation

  rough comparison of function for large n
- usage of asymptotic notation

  describe $f(n)$ (time, space) by simpler $g(n)$