

Analysis of Algorithms CS 3343 Lecture Two

Prof. Jianhua Ruan Substituting
for Prof. William Winsborough
September 4, 2008

Business

- Homework 1 due Thursday September 11
 - Exercises 2.1-3, 2.2-1, 2.2-2, 2.3-1, 2.3-3, 2.3-5

4 September 2008

Winsborough CS 3343 Lecture 2
Delivered by Prof. Ruan

2

Lecture outline

- Correctness of insertion sort
- Time complexity of insertion sort
- If time permits, merge sort

4 September 2008

Winsborough CS 3343 Lecture 2
Delivered by Prof. Ruan

3

Correctness

- For any algorithm, we must prove that it *always* returns the desired output for *all* legal instances of the problem.

4 September 2008

Winsborough CS 3343 Lecture 2
Delivered by Prof. Ruan

4

How to prove correctness?

- Given a **concrete** input, eg. $\langle 4, 2, 6, 1, 7 \rangle$ trace it and prove that it works. **This is not a proof.**
- Given an **abstract** input, eg. $\langle a_1, \dots, a_n \rangle$ trace it and prove that it works.
- If you cannot prove it correct, then prove it is incorrect. Otherwise your algorithm is useless.
 - It might be easier to find a counterexample to show that an algorithm does **NOT** work.

4 September 2008

Winsborough CS 3343 Lecture 2
Delivered by Prof. Ruan

5

Use loop invariants to prove the correctness of loops

- A loop invariant is a formal statement about the variables in your program which holds true throughout the loop
- **Proof** by induction
 - Initialization: the LI is true prior to the 1st iteration
 - Maintenance: if the LI is true before the j^{th} iteration, it remains true before the $(j+1)^{\text{th}}$ iteration
 - Termination: when the loop terminates, the invariant shows the correctness of the algorithm

4 September 2008

Winsborough CS 3343 Lecture 2
Delivered by Prof. Ruan

6

Prove correctness using loop invariants

```

InsertionSort(A, n) {
  for j = 2 to n {
    key = A[j];
    i = j - 1;
    ▶Insert A[j] into the sorted sequence A[1..j-1]
    while (i > 0) and (A[i] > key) {
      A[i+1] = A[i];
      i = i - 1;
    }
    A[i+1] = key;
  }
}

```

Loop invariant: at the start of each iteration of the for loop, the subarray consists of the elements originally in A[1..j-1] but in sorted order.

4 September 2008

Winsborough CS 3343 Lecture 2
Delivered by Prof. Ruan

7

Initialization

```

InsertionSort(A, n) {
  for j = 2 to n {
    key = A[j];
    i = j - 1;
    ▶Insert A[j] into the sorted sequence A[1..j-1]
    while (i > 0) and (A[i] > key) {
      A[i+1] = A[i];
      i = i - 1;
    }
    A[i+1] = key;
  }
}

```

Subarray A[1] is sorted. So loop invariant is true before the loop starts.

4 September 2008

Winsborough CS 3343 Lecture 2
Delivered by Prof. Ruan

8

Maintenance

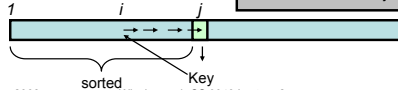
```

InsertionSort(A, n) {
  for j = 2 to n {
    key = A[j];
    i = j - 1;
    ▶Insert A[j] into the sorted sequence A[1..j-1]
    while (i > 0) and (A[i] > key) {
      A[i+1] = A[i];
      i = i - 1;
    }
    A[i+1] = key;
  }
}

```

Assume loop invariant is true prior to iteration j

Loop invariant will be true before iteration j+1



4 September 2008

Winsborough CS 3343 Lecture 2
Delivered by Prof. Ruan

9

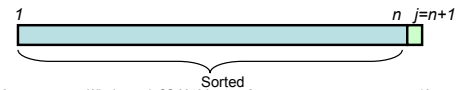
Termination

```

InsertionSort(A, n) {
  for j = 2 to n {
    key = A[j];
    i = j - 1;
    ▶Insert A[j] into the sorted sequence A[1..j-1]
    while (i > 0) and (A[i] > key) {
      A[i+1] = A[i];
      i = i - 1;
    }
    A[i+1] = key;
  }
}

```

Upon termination, A[1..n] contains the original elements in sorted order.



4 September 2008

Winsborough CS 3343 Lecture 2
Delivered by Prof. Ruan

10

Efficiency

- Correctness alone is not enough
- For any problem, there is always a brute-force algorithm
 - Enumerate all possibilities
 - Verify which one is correct
 - Tooooo slow
 - Takes too much space if you have to compare all of them at the same time
- Time efficiency
- Space efficiency

4 September 2008

Winsborough CS 3343 Lecture 2
Delivered by Prof. Ruan

11

Asymptotic Analysis

- Running time depends on the size of the input
 - Larger array takes more time to sort
 - $T(n)$: the time taken on input with size n
 - Look at **growth** of $T(n)$ as $n \rightarrow \infty$.
- “Asymptotic Analysis”
- Size of input is generally defined as the number of input elements
 - In some cases may be tricky

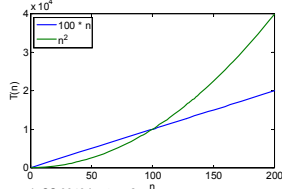
4 September 2008

Winsborough CS 3343 Lecture 2
Delivered by Prof. Ruan

12

Asymptotic Analysis

- Ignore actual and abstract statement costs
- Order of growth** is the interesting measure:
 - Highest-order term is what counts
 - As the input size grows larger it is the high order term that dominates



4 September 2008 Winsborough CS 3343 Lecture 2 Delivered by Prof. Ruan 13

Comparison of functions

	$\log_2 n$	n	$n \log_2 n$	n^2	n^3	2^n	$n!$
10	3.3	10	33	10^2	10^3	10^3	10^6
10^2	6.6	10^2	660	10^4	10^6	10^{30}	10^{158}
10^3	10	10^3	10^4	10^6	10^9		
10^4	13	10^4	10^5	10^8	10^{12}		
10^5	17	10^5	10^6	10^{10}	10^{15}		
10^6	20	10^6	10^7	10^{12}	10^{18}		

For a super computer that does 1 trillion operations per second, it will take longer than 1 billion years

4 September 2008 Winsborough CS 3343 Lecture 2 Delivered by Prof. Ruan 14

Order of growth

$$1 \ll \log_2 n \ll n \ll n \log_2 n \ll n^2 \ll n^3 \ll 2^n \ll n!$$

(We are slightly abusing of the " \ll " sign. It means a smaller order of growth).

4 September 2008 Winsborough CS 3343 Lecture 2 Delivered by Prof. Ruan 15

Asymptotic notations

- O : Big-Oh
- Ω : Big-Omega
- Θ : Theta

4 September 2008 Winsborough CS 3343 Lecture 2 Delivered by Prof. Ruan 16

Big O

- Informally, $O(g(n))$ is the set of all functions with a smaller or same order of growth as $g(n)$, within a constant multiple
- What is $O(n^2)$?
 - The set of all functions that grow slower than or in the same order as n^2

4 September 2008 Winsborough CS 3343 Lecture 2 Delivered by Prof. Ruan 17

So:

- $n \in O(n^2)$
- $n^2 \in O(n^2)$
- $1000n \in O(n^2)$
- $n^2 + n \in O(n^2)$
- $100n^2 + n \in O(n^2)$

Intuitively, O is like \leq

But:

$$1/1000 n^3 \notin O(n^2)$$

4 September 2008 Winsborough CS 3343 Lecture 2 Delivered by Prof. Ruan 18

Big Ω

- Informally, $\Omega(g(n))$ is the set of all functions with a larger or same order of growth as $g(n)$, within a constant multiple

So:

$$n^2 \in \Omega(n)$$

$$1/1000 n^2 \in \Omega(n)$$

Intuitively, Ω is like \geq

But:

$$1000 n \notin \Omega(n^2)$$

4 September 2008

Winsborough CS 3343 Lecture 2
Delivered by Prof. Ruan

19

Theta (Θ)

- Informally, $\Theta(g(n))$ is the set of all functions with the same order of growth as $g(n)$, within a constant multiple
- What is $\Theta(n^2)$?
 - The set of all functions that grow in same order as n^2

4 September 2008

Winsborough CS 3343 Lecture 2
Delivered by Prof. Ruan

20

So:

$$n^2 \in \Theta(n^2)$$

$$n^2 + n \in \Theta(n^2)$$

$$100n^2 + n \in \Theta(n^2)$$

$$100n^2 + \log_2 n \in \Theta(n^2)$$

Intuitively, Θ is like $=$

But:

$$n \log_2 n \notin \Theta(n^2)$$

$$1000n \notin \Theta(n^2)$$

$$1/1000 n^3 \notin \Theta(n^2)$$

4 September 2008

Winsborough CS 3343 Lecture 2
Delivered by Prof. Ruan

21

Formal definitions

- $f(n) = O(g(n))$
 - \exists positive constants c and n_0 such that $0 \leq f(n) \leq cg(n) \forall n > n_0$
 - Or: $\lim_{n \rightarrow \infty} g(n)/f(n) > 0$ (if the limit exists.)
- $f(n) = \Omega(g(n))$
 - \exists positive constants c and n_0 such that $0 \leq cg(n) \leq f(n) \forall n > n_0$
 - Or: $\lim_{n \rightarrow \infty} f(n)/g(n) > 0$ (if the limit exists.)
- $f(n) = \Theta(g(n))$
 - \exists positive constants c_1, c_2 , and n_0 such that $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \forall n \geq n_0 \forall n > n_0$
 - Or: $\lim_{n \rightarrow \infty} f(n)/g(n) = c > 0$ and $c < \infty$
 - Or: $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$

4 September 2008

Winsborough CS 3343 Lecture 2
Delivered by Prof. Ruan

22

Prove Big-Oh

- Claim:** $f(n) = 3n^2 + 10n + 5 \in O(n^2)$

Proof:

$$f(n) = 3n^2 + 10n + 5$$

$$\leq 10n^2 + 10n + 10$$

$$\leq 10n^2 + 10n^2 + 10n^2, \forall n \geq 1$$

$$\leq 30n^2, \forall n \geq 1$$

If we let $c = 30$ and $n_0 = 1$, we have $f(n) \leq cn^2, \forall n \geq n_0$.
Therefore according to definition, $f(n) = O(n^2)$.

- Alternatively**, we can prove that

$$\lim_{n \rightarrow \infty} n^2 / (3n^2 + 10n + 5) = 1/3 > 0$$

4 September 2008

Winsborough CS 3343 Lecture 2
Delivered by Prof. Ruan

23

Prove Big-Omega

- Claim:** $f(n) = n^2 / 10 = \Omega(n)$

Proof:

- $f(n) = n^2 / 10, g(n) = n$
- We only need to find a c and a n_0 to satisfy the definition $f(n) \in \Omega(g(n))$
- $n \leq n^2 / 10$ when $n \geq 10$
- Therefore, $c = 1$ and $n_0 = 10$

Alternatively:

$$\lim_{n \rightarrow \infty} f(n)/g(n) = \lim_{n \rightarrow \infty} (n/10) = \infty$$

4 September 2008

Winsborough CS 3343 Lecture 2
Delivered by Prof. Ruan

24

Prove Theta

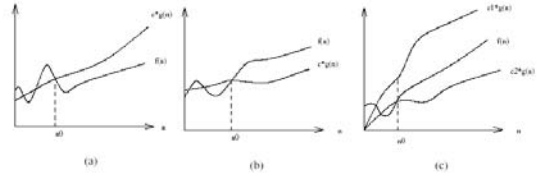
- **Claim:** $f(n) = 2n^2 + n = \Theta(n^2)$
- **Proof:**
 - We just need to find the three constants c_1 , c_2 , and n_0 such that
 - $c_1 n^2 \leq 2n^2 + n \leq c_2 n^2$ for all $n > n_0$
 - A simple solution is $c_1 = 2$, $c_2 = 3$, and $n_0 = 1$
- **Alternatively,** $\lim_{n \rightarrow \infty} (2n^2 + n)/n^2 = 2$

4 September 2008

Winsborough CS 3343 Lecture 2
Delivered by Prof. Ruan

25

O, Ω, and Θ



The definitions imply a constant n_0 *beyond which* they are satisfied. We do not care about small values of n .

4 September 2008

Winsborough CS 3343 Lecture 2
Delivered by Prof. Ruan

26

Using limits to compare orders of growth

$$\lim_{n \rightarrow \infty} f(n) / g(n) = \begin{cases} 0 & \longrightarrow f(n) \in O(g(n)) \\ c > 0 & \longrightarrow f(n) \in \Theta(g(n)) \\ \infty & \longrightarrow f(n) \in \Omega(g(n)) \end{cases}$$

4 September 2008

Winsborough CS 3343 Lecture 2
Delivered by Prof. Ruan

27

- Compare 2^n and 3^n
- $\lim_{n \rightarrow \infty} 2^n / 3^n = \lim_{n \rightarrow \infty} (2/3)^n = 0$
- Therefore, $2^n \in O(3^n)$, and $3^n \in \Omega(2^n)$

4 September 2008

Winsborough CS 3343 Lecture 2
Delivered by Prof. Ruan

28

L' Hopital's rule

$$\lim_{n \rightarrow \infty} f(n) / g(n) = \lim_{n \rightarrow \infty} f'(n) / g'(n) \quad \text{If both } \lim f(n) \text{ and } \lim g(n) \text{ goes to } \infty$$

4 September 2008

Winsborough CS 3343 Lecture 2
Delivered by Prof. Ruan

29

- Compare $n^{0.5}$ and $\log n$
- $\lim_{n \rightarrow \infty} n^{0.5} / \log n = ?$
- $(n^{0.5})' = 0.5 n^{-0.5}$
- $(\log n)' = 1 / n$
- $\lim_{n \rightarrow \infty} (n^{0.5} / 1/n) = \lim_{n \rightarrow \infty} (n^{1.5}) = \infty$
- Therefore, $\log n \in O(n^{0.5})$

4 September 2008

Winsborough CS 3343 Lecture 2
Delivered by Prof. Ruan

30

Stirling's formula

$$n! \approx \sqrt{2\pi n} \left(\frac{n}{e}\right)^n = \sqrt{2\pi n} n^{n+1/2} e^{-n}$$

$$n! \approx (\text{constant}) n^{n+1/2} e^{-n}$$

4 September 2008

Winsborough CS 3343 Lecture 2
Delivered by Prof. Ruan

31

- Compare 2^n and $n!$

$$\lim_{n \rightarrow \infty} \frac{n!}{2^n} = \lim_{n \rightarrow \infty} \frac{c\sqrt{nn^n}}{2^n e^n} = \lim_{n \rightarrow \infty} c\sqrt{n} \left(\frac{n}{2e}\right)^n = \infty$$

- Therefore, $2^n = O(n!)$

4 September 2008

Winsborough CS 3343 Lecture 2
Delivered by Prof. Ruan

32

Kinds of analyses

- Running time depends on the input: an already sorted sequence is easier to sort
- **Worst case**
 - Provides an upper bound on running time
 - An absolute guarantee
- **Best case**
 - Provides a lower bound on running time
- **Average case**
 - Provides the expected running time
 - Very useful, but treat with care: what is "average"?
 - Random (equally likely) inputs
 - Real-life inputs

4 September 2008

Winsborough CS 3343 Lecture 2
Delivered by Prof. Ruan

33

Analysis of insertion Sort

```

InsertionSort(A, n) {
  for j = 2 to n {
    key = A[j]
    i = j - 1;
    while (i > 0) and (A[i] > key) {
      A[i+1] = A[i]
      i = i - 1
    }
    A[i+1] = key
  }
}
    
```

How many times will this line execute?

4 September 2008

Winsborough CS 3343 Lecture 2
Delivered by Prof. Ruan

34

Analysis of insertion Sort

```

InsertionSort(A, n) {
  for j = 2 to n {
    key = A[j]
    i = j - 1;
    while (i > 0) and (A[i] > key) {
      A[i+1] = A[i]
      i = i - 1
    }
    A[i+1] = key
  }
}
    
```

How many times will this line execute?

4 September 2008

Winsborough CS 3343 Lecture 2
Delivered by Prof. Ruan

35

Analysis of insertion Sort

Statement	cost time	
InsertionSort(A, n) {		
for j = 2 to n {	c_1	n
key = A[j]	c_2	$(n-1)$
i = j - 1;	c_3	$(n-1)$
while (i > 0) and (A[i] > key) {	c_4	S
A[i+1] = A[i]	c_5	$(S-(n-1))$
i = i - 1	c_6	$(S-(n-1))$
}	0	
A[i+1] = key	c_7	$(n-1)$
}	0	
}		

$S = t_2 + t_3 + \dots + t_n$, where t_j is number of while expression evaluations for the j^{th} for loop iteration

4 September 2008

Winsborough CS 3343 Lecture 2
Delivered by Prof. Ruan

36

Analysis of insertion Sort

Statement	cost	time
<code>InsertionSort(A, n) {</code>		
<code>for j = 2 to n {</code>	C_1	n
<code>key = A[j]</code>	C_2	$(n-1)$
<code>i = j - 1;</code>	C_3	$(n-1)$
<code>while (i > 0) and (A[i] > key) {</code>	C_4	S
<code>A[i+1] = A[i]</code>	C_5	$(S-(n-1))$
<code>i = i - 1</code>	C_6	$(S-(n-1))$
<code>}</code>	0	
<code>A[i+1] = key</code>	C_7	$(n-1)$
<code>}</code>	0	
<code>}</code>		

4 September 2008

Winsborough CS 3343 Lecture 2
Delivered by Prof. Ruan

37

Analysis of insertion Sort

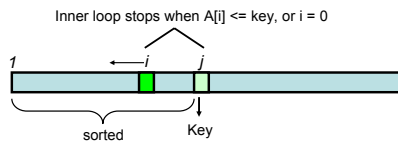
Statement	cost	time
<code>InsertionSort(A, n) {</code>		
<code>for j = 2 to n {</code>	C_1	n
<code>key = A[j]</code>	C_2	$(n-1)$
<code>i = j - 1;</code>	C_3	$(n-1)$
<code>while (i > 0) and (A[i] > key) {</code>	C_4	S
<code>A[i+1] = A[i]</code>	C_5	$(S-(n-1))$
<code>i = i - 1</code>	C_6	$(S-(n-1))$
<code>}</code>	0	
<code>A[i+1] = key</code>	C_7	$(n-1)$
<code>}</code>	0	
<code>}</code>		

4 September 2008

Winsborough CS 3343 Lecture 2
Delivered by Prof. Ruan

38

What can S be?



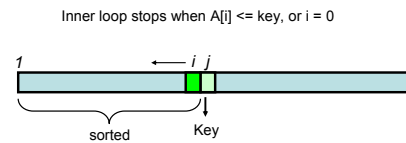
- $S = \sum_{j=1..n} t_j$
- Best case:
- Worst case:
- Average case:

4 September 2008

Winsborough CS 3343 Lecture 2
Delivered by Prof. Ruan

39

Best case



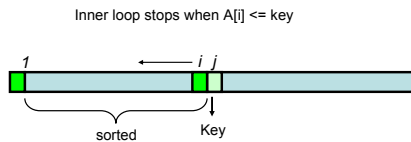
- Array already sorted
- $S = \sum_{j=1..n} t_j$
- $t_j = 1$ for all j
- $S = n$. $T(n) = \Theta(n)$

4 September 2008

Winsborough CS 3343 Lecture 2
Delivered by Prof. Ruan

40

Worst case



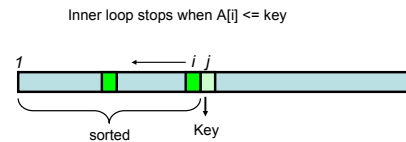
- Array originally in reverse order
- $S = \sum_{j=1..n} t_j$
- $t_j = j$
- $S = \sum_{j=1..n} j = 1 + 2 + 3 + \dots + n = n(n+1)/2 = \Theta(n^2)$

4 September 2008

Winsborough CS 3343 Lecture 2
Delivered by Prof. Ruan

41

Average case



- Array in random order
- $S = \sum_{j=1..n} t_j$
- $t_j = j/2$ in average
- $S = \sum_{j=1..n} j/2 = \frac{1}{2} \sum_{j=1..n} j = n(n+1)/4 = \Theta(n^2)$

4 September 2008

Winsborough CS 3343 Lecture 2
Delivered by Prof. Ruan

42

Divide and Conquer

- General idea:
 - **Divide** a large problem into **smaller** ones
 - By a constant ratio
 - By a constant or some variable
 - **Solve each smaller one** *recursively* or *explicitly*
 - **Combine** the solutions of smaller ones to form a solution for the original problem

4 September 2008

Winsborough CS 3343 Lecture 2
Delivered by Prof. Ruan

43

Merge sort

MERGE-SORT $A[1 \dots n]$

1. If $n = 1$, done.
2. Recursively sort $A[1 \dots \lceil n/2 \rceil]$ and $A[\lceil n/2 \rceil + 1 \dots n]$.
3. “**Merge**” the 2 sorted lists.

Key subroutine: MERGE

4 September 2008

Winsborough CS 3343 Lecture 2
Delivered by Prof. Ruan

44

Merging two sorted arrays

20 12
13 11
7 9
2 1

4 September 2008

Winsborough CS 3343 Lecture 2
Delivered by Prof. Ruan

45

Merging two sorted arrays

20 12
13 11
7 9
2 1

4 September 2008

Winsborough CS 3343 Lecture 2
Delivered by Prof. Ruan

46

Merging two sorted arrays

20 12
13 11
7 9
2 1

1

4 September 2008

Winsborough CS 3343 Lecture 2
Delivered by Prof. Ruan

47

Merging two sorted arrays

20 12 | 20 12
13 11 | 13 11
7 9 | 7 9
2 1 | 2

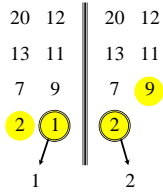
1

4 September 2008

Winsborough CS 3343 Lecture 2
Delivered by Prof. Ruan

48

Merging two sorted arrays

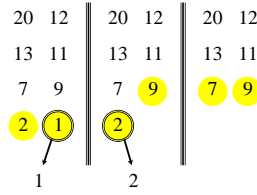


4 September 2008

Winsborough CS 3343 Lecture 2
Delivered by Prof. Ruan

49

Merging two sorted arrays

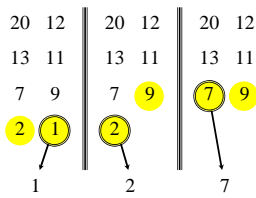


4 September 2008

Winsborough CS 3343 Lecture 2
Delivered by Prof. Ruan

50

Merging two sorted arrays

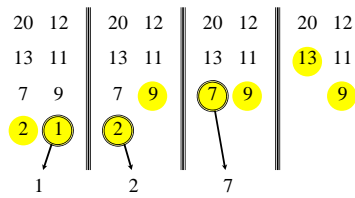


4 September 2008

Winsborough CS 3343 Lecture 2
Delivered by Prof. Ruan

51

Merging two sorted arrays

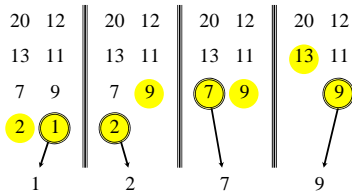


4 September 2008

Winsborough CS 3343 Lecture 2
Delivered by Prof. Ruan

52

Merging two sorted arrays

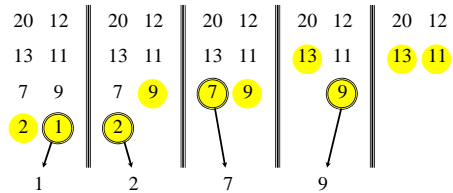


4 September 2008

Winsborough CS 3343 Lecture 2
Delivered by Prof. Ruan

53

Merging two sorted arrays

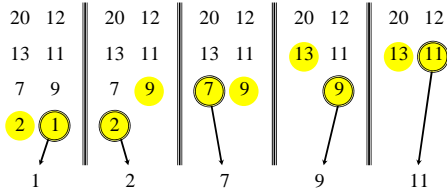


4 September 2008

Winsborough CS 3343 Lecture 2
Delivered by Prof. Ruan

54

Merging two sorted arrays

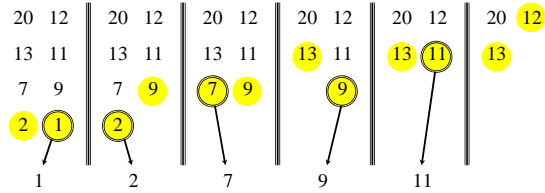


4 September 2008

Winsborough CS 3343 Lecture 2
Delivered by Prof. Ruan

55

Merging two sorted arrays

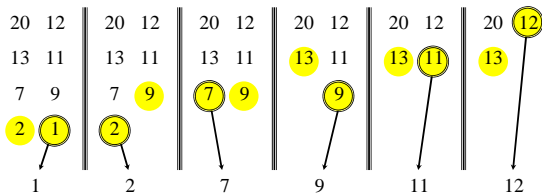


4 September 2008

Winsborough CS 3343 Lecture 2
Delivered by Prof. Ruan

56

Merging two sorted arrays



4 September 2008

Winsborough CS 3343 Lecture 2
Delivered by Prof. Ruan

57

How to show the correctness of a recursive algorithm?

- By induction:
 - **Base case:** prove it works for small examples
 - **Inductive hypothesis:** assume the solution is correct for all sub-problems
 - **Step:** show that, if the induction hypothesis is correct, then the algorithm is correct for the original problem.

4 September 2008

Winsborough CS 3343 Lecture 2
Delivered by Prof. Ruan

58

Correctness of merge sort

MERGE-SORT $A[1..n]$

1. If $n = 1$, done.
2. Recursively sort $A[1.. \lceil n/2 \rceil]$ and $A[\lceil n/2 \rceil + 1..n]$.
3. “Merge” the 2 sorted lists.

Proof:

1. **Base case:** if $n = 1$, the algorithm will return the correct answer because $A[1..1]$ is already sorted.
2. **Inductive hypothesis:** assume that the algorithm correctly sorts $A[1.. \lceil n/2 \rceil]$ and $A[\lceil n/2 \rceil + 1..n]$.
3. **Step:** if $A[1.. \lceil n/2 \rceil]$ and $A[\lceil n/2 \rceil + 1..n]$ are both correctly sorted, the whole array $A[1.. \lceil n/2 \rceil]$ and $A[\lceil n/2 \rceil + 1..n]$ is sorted after merging.

4 September 2008

Winsborough CS 3343 Lecture 2
Delivered by Prof. Ruan

59

How to analyze the time-efficiency of a recursive algorithm?

- Express the running time on input of size n as a function of the running time on smaller problems

4 September 2008

Winsborough CS 3343 Lecture 2
Delivered by Prof. Ruan

60

Analyzing merge sort

$T(n)$
 $\Theta(1)$
 $2T(n/2)$
 $f(n)$

MERGE-SORT $A[1 \dots n]$
 1. If $n = 1$, done.
 2. Recursively sort $A[1 \dots \lceil n/2 \rceil]$ and $A[\lceil n/2 \rceil + 1 \dots n]$.
 3. “Merge” the 2 sorted lists

Sloppiness: Should be $T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor)$, but it turns out not to matter asymptotically.

4 September 2008

Winsborough CS 3343 Lecture 2
Delivered by Prof. Ruan

61

Analyzing merge sort

1. **Divide:** Trivial.
2. **Conquer:** Recursively sort 2 subarrays.
3. **Combine:** Merge two sorted subarrays

$$T(n) = 2T(n/2) + f(n) + \Theta(1)$$

subproblems subproblem size Work dividing and Combining

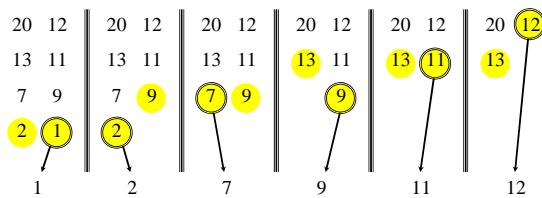
1. What is the time for the base case? **Constant**
2. What is $f(n)$?
3. What is the growth order of $T(n)$?

4 September

Delivered by Prof. Ruan

62

Merging two sorted arrays



$\Theta(n)$ time to merge a total of n elements (linear time).

4 September 2008

Winsborough CS 3343 Lecture 2
Delivered by Prof. Ruan

63

Recurrence for merge sort

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1; \\ 2T(n/2) + \Theta(n) & \text{if } n > 1. \end{cases}$$

- Later we shall often omit stating the base case when $T(n) = \Theta(1)$ for sufficiently small n , but only when it has no effect on the asymptotic solution to the recurrence.

• But what does $T(n)$ solve to? I.e., is it $O(n)$ or $O(n^2)$ or $O(n^3)$ or ...?

4 September 2008

Winsborough CS 3343 Lecture 2
Delivered by Prof. Ruan

64