CS140 Pragmatism

What is the asymptotic, worst-case running time of the algorithm?

Types of Algorithms

- Recursive Algorithm: one that calls itself
- Purely Iterative Algorithm: one that doesn’t

Run Time Analysis

- **Iterative algorithm ® Loop counting**
- Recursive algorithm → Recurrence relations

Iterative Sorting Algorithms

- Insertion-sort
- Bubble-sort
- Modified Bubble-sort

Insertion-sort(S)

(in pseudo-code)

S is an array of n integers: S(1), S(2), …, S(n)

```
For j = 2 to n
   key = S(j)
   i = j-1
   While i > 0 and S(i) > key
      S(i+1) = S(i--)
   S(i+1) = key
```

C decrement operator

Insertion Sort – Status after each change

j	key	i	S(1)	S(2)	S(3)	S(4)	S(5)
2	1	1	3	1	5	2	4
2	1	**0**	3	**3**	5	2	1
2	1	0	**1**	3	5	2	1

Correctness

- Is Insertion-sort correct?
- Proof by loop invariant:

  When the **for loop** executes for the k^{th} time, S(1), S(2), …, S(k) are sorted in ascending order.

  (Prove claim by induction on k.)

Loop Counting: Insertion-sort(S)

$\Sigma_{j=2..n}$ (1	For j = 2 to n
+1	key = S(j)
+1	i = j-1
+ $\Sigma_{i=0..j-1}$ (1	While i>0 & S(i)>key
+1)	S(i+1) = S(i--)
+1)	S(i+1) = key

$$\boxed{\Sigma_{j=2..n}(4+\Sigma_{i=0..j-1} 2) = O(n^2)}$$

Bubble-sort(S)

```
Bubble-sort(S)
    For i=n down to 2
        For j=1 to i-1
            If S(j) >S(j+1) then swap(S(j), S(j+1) )
    Return
```


BubbleSort – Status after each change

i	j	S(1)	S(2)	S(3)	S(4)	S(5)
5	1	3	1	5	2	4
5	1	**1**	**3**	5	2	4
5	**2**	1	3	5	2	4

Correctness

- Is Bubble-sort correct?
- Proof by loop invariant:

  When the i-loop completes its k^{th} execution,
  - S(n-k+1), S(n-k+2), …, S(n) is sorted in ascending order, and
  - the max(S(1), …, S(n-k)) ≤ S(n-k+1).

Does Bubble-sort do too much work?

1, 3, 2, 4, 5
1, 2, 3, 4, 5
1, 2, 3, 4, 5
1, 2, 3, 4, 5
1, 2, 3, 4, 4

- Repeat on smaller list

Modified Bubble-sort

```
Modified-Bubble-sort(S)
    SWAP=T
    For i=n down to 2
        If SWAP=F then return
        SWAP=F
        For j=1 to i-1
            If S(j)>S(j+1) then swap(S(j),S(j+1)) and set
              SWAP=T
    Return
```


Example

1, 3, 2, 4, 5

1, 2, 3, 4, 5

1, 2, 3, 4, 5

- Repeat on smaller list unless no swaps are made

Loop counting: M-Bubble-sort

	Modified-Bubble-sort(S)
1 + →	SWAP=T
$\Sigma_{i=2..n}$(1+ →	For i=n down to 2
1+ →	If SWAP=F then return
1+ →	Flag=F
$\Sigma_{j=1..i-1}$(1+ →	For j=1 to i-1
3) →	If $s_j>s_{j+1}$ then swap(s_j, s_{j+1}) and set SWAP=T
	Return

$$\boxed{1+\Sigma_{i=2\ldots n}(3+\Sigma_{j=1\ldots i-1}\,4)) = O(n^2)}$$

Loop Counting: Bubble-sort

- Also $O(n^2)$

Summation

$$\Sigma_{i=2\ldots n}\,\Sigma_{j=1\ldots i-1}c = c\Sigma_{i=2\ldots n}(i-1)$$

$$= c(\Sigma_{i=1\ldots n}(i-1)) - c$$

$$= c(\Sigma_{i=1\ldots n}i - \Sigma_{i=1\ldots n}1) - c$$

$$= c(n(n+1)/2 - n) - c$$

$$= \mathbf{O(n^2)}$$

Series

- A series is a summation of terms
- Common series
  - Arithmetic series: $1+2+\ldots+n$
  - Geometric series: $1+a+a^2+\ldots+a^n$

Series
Things we want to do:

- **Solve exactly**
- Bound above or below
- Prove that a solution (or bound) is correct

Closed form solutions to some common series

- $f(n) = 1+2+ \ldots + n = n(n+1)/2$
- $f(n) = 1^2 + 2^2 + \ldots + n^2 = (2n^3 + 3n^2 + n)/6$
- $f(n) = 1+a+a^2+\ldots+a^n = n$ if $a=1$
  $= (a^{n+1}-1)/(a-1)$ else
- $f(n) = 1+a+a^2+\ldots = 1/(1-a)$ if $0 \le a<1$

Series
Things we want to do:

- Solve exactly
- **Bound above or below**
- Prove that a solution (or bound) is correct

Upper Bounds on series

For any constant k:

$$\begin{aligned} \Sigma_{i=1..n} i^k &\le \Sigma_{i=1..n} n^k \\ &= (n^{k+1}) \\ &= O(n^{k+1}) \end{aligned}$$

Is this a good upper bound?

Lower Bounds on series

For any constant k:

$$\begin{aligned} \Sigma_{i=1..n} i^k &\ge \Sigma_{i=\lfloor n/2 \rfloor \ldots n}\, i^k \\ &\ge \Sigma_{i=\lfloor n/2 \rfloor \ldots n} \lfloor n/2 \rfloor^k \\ &\ge \lfloor n/2 \rfloor^{k+1} \\ &= \Omega(n^{k+1}) \end{aligned}$$

So $\Sigma_{i=1..n} i^k = \Theta(n^{k+1})$

Series
Things we want to do:

- Solve exactly
- Bound above or below
- **Prove that a solution (or bound) is correct**

Proving correctness

- Claim: $\Sigma_{i=1..n}\, i^2 = (2n^3 + 3n^2 + n)/6$
- Claim holds for n=1.
- If the claim holds for n then it holds for n+1:

$$\Sigma_{i=1..n+1}\, i^2 = (n+1)^2 + \Sigma_{i=1..n}\, i^2$$
$$= (n+1)^2 + (2n^3 + 3n^2 + n)/6$$
$$= (2(n+1)^3 + 3(n+1)^2 + (n+1))/6$$

Run Time Analysis

- Iterative algorithm $\rightarrow$ Loop counting
- **Recursive algorithm ® Recurrence relations**

Sort3: A Recursive Algorithm for SIAO

```
Sort3(S)
    If ||S||≤1
        Return:   S
    Else
        Return:   Sort3(S\max-element(S)),max-element(S)
```


Example: Sort3(3,1,5,2,4)

Sort3(3,1,5,2,4) = Sort3(3,1,2,4),5
= Sort3(3,1,2),4,5
= Sort3(1,2),3,4,5
= Sort3(1),2,3,4,5
= 1,2,3,4,5

Recursive Algorithms

What about Sort3?

```
Sort3(S)
    If ||S||≤1
        Return:   S
    Else
        Return:   Sort3(S\max-element(S)),max-element(S)
```

Let T(n) be the running time of Sort3:

$T(1) = c_2$
$T(n) = c_1 n + T(n-1)$, n>1

Recurrence Relations

Methods to solve or bound:

- Guess and prove
- Unwinding
- Master method
- **WORK TREES**

Guess and Prove (bound)

- Guess: $T(n) = O(n^2)$
- Proof: We need to show that there exists constants c and M such that $T(n) \le cn^2$ for all $n \ge M$

9/6/00 CS140(3) 31

Guess and Prove cont.

- $T(1) \le c$, $\boxed{\text{provided } c \ge c_1}$
- Suppose $T(n-1) \le c(n-1)^2$.
  - $T(n) = c_2 n + T(n-1)$
  - $\le c_2 n + c(n-1)^2$
  - $= c_2 n + c(n^2 - 2n + 1)$
  - $= cn^2 - (2c - c_2)n + c$
  - $\le cn^2$, $\boxed{\text{provided } c \ge c_2 \text{ and } n \ge 1}$

9/6/00 CS140(3) 32

Guess and Prove cont.

- $T(n) \le cn^2$ for all $n \ge 1$, where $c = \max(c_1, c_2)$

$\Downarrow$

- $T(n) = O(n^2)$

9/6/00 CS140(3) 33

Unwinding

$$
\begin{aligned}
T(n) &\le c_2 n + T(n-1) \\
&\le c_2 n + c_2(n-1) + T(n-2) \\
&\le c_2 n + c_2(n-1) + c_2(n-2) + T(n-3)
\end{aligned}
$$

.
.
.

$$
\begin{aligned}
T(n) &\le c_1 + \Sigma_{i=2..n} c_2 i = (c_1 - c_2) + c_2 n(n-1)/2 \\
&= O(n^2)
\end{aligned}
$$

9/6/00 CS140(3) 34

Master Theorem

- Read the book

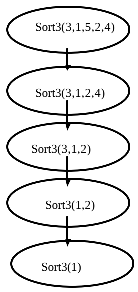
9/6/00 CS140(3) 35

Work Tree

- A rooted tree for algorithm A on input size n:
  - Each node corresponds to a (recursive) call of A
  - An edge from u to v represents the fact that the recursive call v is made from within u.

9/6/00 CS140(3) 36

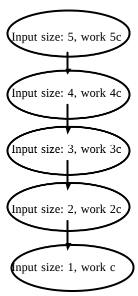
Example: Sort3(3,1,5,2,4)



Work Tree

- The “work” done at a node is the number of steps performed by the algorithm within the recursive call

Example: Sort3(3,1,5,2,4)



Sort3 Work Tree

- Consider a node at level i, where the root is at level 0:
  - What is the input size? n-i
  - What is the work done? c(n-i)
- How many nodes are there at level i? 1
- What is the total work done at level i? c(n-i)
- How many levels are there in the tree? n
- What is the total work done? $\Sigma_{i=0..(n-1)}c(n-i) = O(n^2)$

Merge-sort

Merge-sort(S={s_1, s_2, ... , s_n})
    If n=1 return(S)
    Else
        S_1 = Merge-sort(s_1, ... , $s_{\lfloor n/2 \rfloor}$)
        S_2 = Merge-sort($s_{\lfloor n/2 \rfloor+1}$, ... , s_n)
        Return Merge(S_1 , S_2)

Merge($s_1,s_2,\ldots,s_k$; $t_1,t_2,\ldots,t_j$)

(k>0 and j>0)

- If $s_1 \leq t_1$ then output

  s_1, Merge($s_2,\ldots,s_k$; $t_1,t_2,\ldots,t_j$)
- Else output

  t_1, Merge(s_1, $s_2,\ldots,s_k$; $t_2,\ldots,t_j$)

Merge-sort(4,1,3,2)

Input: 4,1,3,2
Merge-sort(4,1)

Input: 4,1
Merge-sort(4)

Input: 4

Merge-sort(4,1,3,2)

Input: 4,1,3,2
Merge-sort(4,1)

Input: 4,1
Merge-sort(4) =4

Merge-sort(4,1,3,2)

Input: 4,1,3,2
Merge-sort(4,1)

Input: 4,1
Merge-sort(4) =4
Merge-sort(1)

Input: 1

Merge-sort(4,1,3,2)

Input: 4,1,3,2
Merge-sort(4,1)

Input: 4,1
Merge-sort(4) =4
Merge-sort(1) =1
Merge(4;1)=1,4

Merge-sort(4,1,3,2)

Input: 4,1,3,2
Merge-sort(4,1)=1,4
Merge-sort(3,2)

Input: 3,2
Merge-sort(3) =3
Merge-sort(2) =2
Merge(3;2)=2,3

Merge-sort(4,1,3,2)

Input: 4,1,3,2
Merge-sort(4,1)=1,4
Merge-sort(3,2)=3,4
Merge(1,4; 2,3) = 1,2,3,4

Is Merge-sort correct?

- If n=1 then yes
- If n>1 then
  - We can assume Merge-sort(S(1),…,S($\lfloor n/2 \rfloor$)) and Merge-sort(S($\lfloor n/2 \rfloor$+1), … , S(n)) return correctly sorted lists.
  - So the merge of these lists is a correctly sorted list.

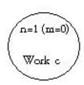
9/6/00 CS140(3) 49

How fast is Merge-sort? (Assume $n=2^m$)

- m=0: T(1) = c
- m>0: $T(2^m) = 2T(2^{m-1}) + c2^m$

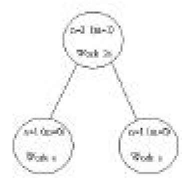
9/6/00 CS140(3) 50

Work Tree for Merge-sort
Input Size: 1 (m=0)



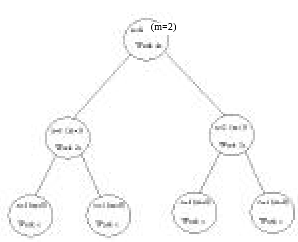
9/6/00 CS140(3) 51

Work Tree for Merge-sort
Input Size: 2 (m=1)



9/6/00 CS140(3) 52

Work Tree for Merge-sort
Input Size: 4 (m=2)



9/6/00 53

Work Tree for Merge-sort
Input Size: $n=2^m$

A root with two sub-trees
- Root
  - Input Size: n
  - Work: cn
- Each child
  - Roots a work tree with Input Size 2^{m-1}

9/6/00 CS140(3) 54

Work Tree for Merge-sort

Input Size: $n=2^m$

Properties of nodes at level i (root is at level 0):

- Input size:
- Work:

Properties of level i:

- Number of nodes at level i:
- Total work of nodes at level i:

Property of tree:

- Number of levels:
- Total work:

Work Tree for Merge-sort

Input Size: $n=2^m$

Properties of nodes at level i (root is at level 0):

- Input size: 2^{m-i}
- Work: $c2^{m-i}$

Properties of level i:

- Number of nodes at level i: 2^i
- Total work of nodes at level i: $c2^m$

Property of tree:

- Number of levels: $m+1$
- Total work: $c(m+1)2^m$ or $O(n \lg n)$

What if $n \neq \lceil\lceil n \rceil\rceil$?

- Claim 1: $T(n) = O(T(\lceil\lceil n \rceil\rceil))$
- Claim 2: $\lceil\lceil n \rceil\rceil \lg \lceil\lceil n \rceil\rceil = O(n \lg n)$