CS140: Algorithms

Z Sweedyk
Lecture 2
1/23/01

Today

- **Recurrence relations**
- Work trees
- Divide and conquer

Run Time Analysis

- Iterative algorithm $\rightarrow$ Loop counting

- **Recursive algorithm ® Recurrence relations**
  1. **Write the recurrence relation**
  2. **Convert the recurrence relation to a series**
  3. **Solve the series**

Sort3: A Recursive Algorithm for SIAO

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


1. Write the recurrence relation

```
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

2. Convert to series

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

$\Downarrow$

$c_2 + \Sigma_{i=2 \ldots n} c_1 n$

3. Solve

$c_2 + \Sigma_{i=2 \ldots n} c_1 n \Rightarrow c_2 - c_1 + c_1 n(n+1)/2 = O(n^2)$

Recurrence Relations

Shortcuts and other tools:

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

Steps 2-3: Guess and Prove

- $T(n)=c_1 n + T(n-1)$, $T(1)=c_2$
- Guess: $T(n) = O(n^2)$
- Prove: We need to show that there exists constants c and M such that $T(n) \leq cn^2$ for all $n \geq M$

Guess and Prove cont.

- $T(1) \leq c$, $\boxed{\text{provided } c \geq c_1}$
- Suppose $T(n-1) \leq c(n-1)^2$.

$$
\begin{aligned}
T(n) &= c_2 n + T(n-1) \\
&\leq c_2 n + c(n-1)^2 \\
&= c_2 n + c(n^2 - 2n + 1) \\
&= cn^2 - (2c - c_2)n + c \\
&\leq cn^2, \ \boxed{\text{provided } c \geq c_2 \text{ and } n \geq 1}
\end{aligned}
$$

Guess and Prove cont.

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

$\Downarrow$

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

Guess and Prove cont.

- What if you guess is wrong?
- You'll reach a contradiction in the proof step

Recurrence Relations

Shortcuts and other tools:

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

1/23/01 CS140 - Lec 2 13

Steps 2-3: Master Theorem

- Read the book

Warning – only works for certain types of recurrence relations

1/23/01 CS140 - Lec 2 14

Recurrence Relations

Shortcuts and other tools:

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

1/23/01 CS140 - Lec 2 15

Step 2: Unwinding

$T(n) \leq c_2 n + T(n-1)$

$\leq c_2 n + c_2(n-1) + T(n-2)$

$\leq c_2 n + c_2(n-1) + c_2(n-2) + T(n-3)$

.

.

.

$T(n) \leq c_1 + \Sigma_{i=2..n} c_2 i$

1/23/01 CS140 - Lec 2 16

Recurrence Relations

Shortcuts and other tools:

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

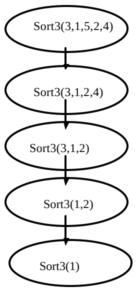
1/23/01 CS140 - Lec 2 17

Step 2 (and a little 3): Work Tree

- Topology: 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.
- $T(n) = cn + T(n-1)$

1/23/01 CS140 - Lec 2 18

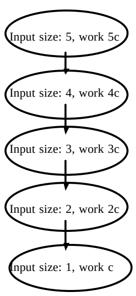
Example: Sort3(3,1,5,2,4)



Work Tree

- The “work” done at a node is the number of steps performed by the algorithm within the recursive call.
- $T(n) = \boxed{cn} + T(n-1)$

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?
  - What is the work done?
- How many nodes are there at level i?
- What is the total work done at level i?
- How many levels are there in the tree?
- What is the total work done?

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? $\Sigma_{i=0..(n-1)} c(n-i)$

Merge-sort

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

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

(k>0 and j>0)

- If $s_1 \le t_1$ then output

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

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

1/23/01 CS140 - Lec 2 25

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

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

↓

Input: 4,1
Merge-sort(4)

↓

Input: 4

1/23/01 CS140 - Lec 2 26

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

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

↓

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

1/23/01 CS140 - Lec 2 27

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

1/23/01 CS140 - Lec 2 28

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

1/23/01 CS140 - Lec 2 29

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

1/23/01 CS140 - Lec 2 30

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

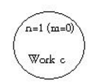
Is Merge-sort correct?

- 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.

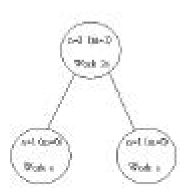
How fast is Merge-sort? (Assume $n=2^m$)

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

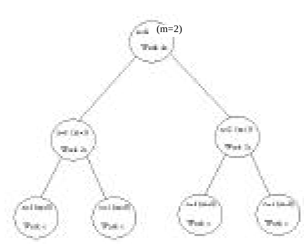
Work Tree for Merge-sort
Input Size: 1 (m=0)



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



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



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}

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:
  $T(\lceil\lceil n \rceil\rceil) = O(\lceil\lceil n \rceil\rceil \lg \lceil\lceil n \rceil\rceil)$
- Claim 3:
  $\lceil\lceil n \rceil\rceil \lg \lceil\lceil n \rceil\rceil = O(n \lg n)$

Today

- Recurrence relations
- Work trees
- **Divide and conquer**

Divide and Conquer

"Divide and conquer" is an algorithmic technique:

- Break the problems into ***a*** sub-problems of size ***n/b***
- Solve the sub-problems
- Combine the solutions to the sub-problems to create a solution for the original problem

Divide and Conquer

- "Divide and conquer" recurrence relations

  T(n)=a T(n/b) + f(n)

  T(1)=c=f(1)

Analysis

- Consider the case when $n=b^m$
- Then generalize to arbitrary n

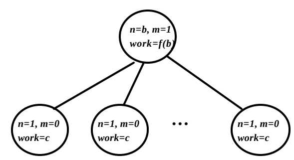
Work Tree for Divide and Conquer: $n=b^m$

m=0

Total work: c

Work Tree for Divide and Conquer: $n=b^m$

m=1



Total work: ***f(b) +ac***

Work Tree for Divide and Conquer: $n=b^m$

A root with a sub-trees

- Root
  - Input Size: $n=b^m$
  - Work: $f(n)$
- Each child
  - Roots a work tree with Input Size b^{m-1}

Work Tree for Divide and Conquer

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 Divide and Conquer

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

- Input size: n/b^i
- Work: $f(n/b^i)$

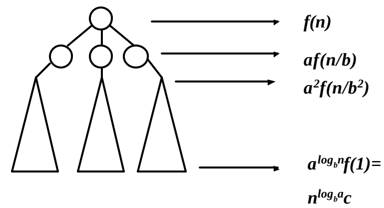
Properties of level i:

- Number of nodes at level i: a
- Total work of nodes at level i: $af(n/b^i)$

Property of tree:

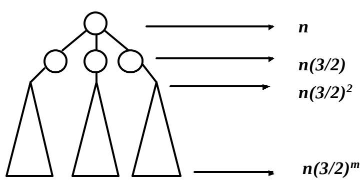
- Number of levels: $m+1$
- Total work: $\Sigma_{i=0,\ldots m} a^i f(n/b^i)$ =?

Work Tree for divide and conquer algorithm



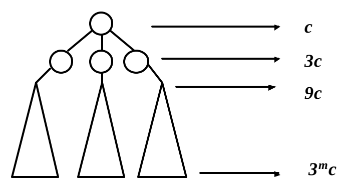
Total work: $\Sigma_{i=0\ldots m} a^i f(n/b^i)$

Work Tree for divide and conquer algorithm: a=3, b=2, f(n)=n



Total work: $\Sigma_{i=0\ldots m} n(3/2)^i$

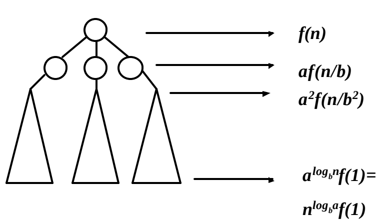
Work Tree for divide and conquer algorithm: a=3, b=2, f(n)=c



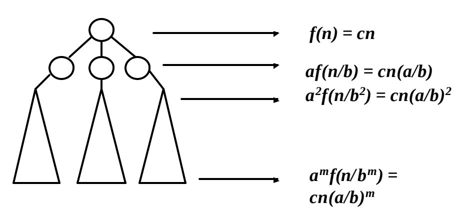
Total work: $\Sigma_{i=0\ldots m} c3^i$

Where is "most" of the work?

f(n) is slow-growing ↔ f(n) is fast-growing



$f(n)=cn,\ m=\log_b n$



$T(n) = cn\, \Sigma_{i=0\ldots m} (a/b)^i$

Total Work

$T(n) = cn\, \Sigma_{i=0\ldots m}\, (a/b)^i$

$a<b$:

$a=b$:

$a>b$:

Total Work

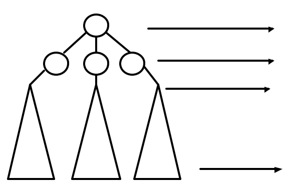
$T(n) = cn\, \Sigma_{i=0\ldots m}\, (a/b)^i$

$a<b$: $O(n)$

$a=b$: $O(n\lg(n))$

$a>b$: $O(n^{\log_b a})$

$f(n)=cn^k$



$f(n) = cn^k$

$af(n/b) = cn^k(a/b^k)$

$a^2f(n/b^2) = cn^k(a/b^k)^2$

$a^mf(n/b^m) =$
$cn^k(a/b^k)^m$

$T(n) = cn^k\, \Sigma_{i=0\ldots m}\, (a/b^k)^i$

Total Work

$T(n) = cn^k\, \Sigma_{i=0\ldots m}\, (a/b^k)^i$

$a<b^k$:

$a=b^k$:

$a>b^k$:

Total Work

$T(n) = cn^k\, \Sigma_{i=0\ldots m}\, (a/b^k)^i$

$a<b^k$: $\Theta(n^k)$

$a=b^k$: $\Theta(n^k\lg(n))$

$a>b^k$: $\Theta(n^{\log_b a})$