

Section 5.3 Divide and Conquer Relations

The form:

$$a_n = a_{n/m} + f(n)$$

The sequence:

$$\{a_{m^0}, a_{m^1}, a_{m^2}, \dots, a_{m^k}, \dots\}$$

- $n = m^k$ for some k .
 - Division of the problem in half gives $m = 2$
 - Division into thirds gives $m = 3$
 - etc.

Apply the *telescoping* technique described in Section 5.2.

Example:

- $m = 2$,

$$a_n = 2a_{n/2} + 1, a_1 = 3$$

n is a power of 2:

$$n = 2^k \text{ for some arbitrary } k.$$

Hence,

$$\log(n) = k.$$

and

$$n/2^k = 1$$

which is the value for the initial condition.

Using telescoping you are *dividing the previous index by 2* each time vs. *subtracting one from it*.

$$a_n = (2a_{n/2}) + 1$$

$$(2a_{n/2}) = [2^2 a_{n/2^2}] + 2$$

$$[2^2 a_{n/2^2}] = 2^3 a_{n/2^3} + 2^2$$

•
•
•

$$2^{k-1} a_{n/2^{k-1}} = 2^k a_{n/2^k} + 2^{k-1}$$

$$a_n = na_1 + \sum_{i=0}^{k-1} 2^i = 3n + n - 1 = 4n - 1 \quad O(n)$$

Divide and Conquer Algorithms

Note: most of the following algorithms are presented only to show how to analyze them using recurrence relations.

THEY DON'T NECESSARILY DO ANYTHING USEFUL!

We assume we have list processing capability in the pseudocode.

Example:

procedure DAC(*list*)

/ divide the list in half each time. Once the length of the list is one, square the number and return it. Multiply the two values returned from each half of the list. */*

if Length(*list*) = 1

return (Listhead(*list*)) • (Listhead(*list*))

a₁ = one mult

else

c = DAC(first half of *list*)

a_{n/2} mults

d = DAC(second half of *list*)

a_{n/2} mults

return (c • d)

one mult

Count multiplications

- list length $n = 2^k$.
- The initial condition:

If $n = 2^0 = 1$ - one multiplication is required so

$$a_1 = 1.$$

- Multiplications required to compute c

$$= a_{n/2}$$

plus

- Multiplications required to compute d

$$= a_{n/2}$$

plus

- Multiplications required to compute $a \cdot b$

$$1.$$

The recurrence equation becomes:

$$a_n = 2a_{n/2} + 1$$

with the initial condition $a_1 = 1$.

The only difference between this one and the previous one is the initial condition.

Hence, the solution is

$$a_n = 1(2^k) + 2^k - 1 = 2^{k+1} - 1 = 2n - 1 \quad O(n)$$

Example:

```
procedure TRIPLE(list)
    if Length(list) = 1 then return (Listhead(list) +
(Listhead(list))                                1 add
    else
        a = TRIPLE(first third of list)           $a_{n/3}$  adds
        b = TRIPLE(last third of list)            $a_{n/3}$  adds
    return (a + a + b)                          2 adds
```

Note: we only process the first and last third of the list.

We count the number of additions required to process a list of length $n = 3^k$. The recurrence relation becomes:

$$a_n = 2a_{n/3} + 2, a_1 = 1$$

Use the telescoping technique to get:

$$a_n = 2a_{n/3} + 2$$

$$2a_{n/3} = 2^2 a_{n/3^2} + 2^2$$

•
•

•

$$2^{k-1} a_{n/3^{k-1}} = 2^k a_{n/3^k} + 2^k$$

$$a_n = 1(2^k) + \sum_{i=1}^k 2^i$$

which we must put in terms of n vs. k .

But if $n = 3^k$, then

$$\log_3(n) = k$$

or

$$2^k = 2^{\log_3(n)} = \log_2 \sqrt[3]{n} = n^{1.58}$$

and so forth.

Merge-Sort

Merge-Sort is an asymptotically optimal worst case sorting algorithm.

We sort a list of $n = 2^k$ elements by divide and conquer.

- Divide the list in two halves
- Sort each half
- Merge the two lists to produce a sorted list

The length of the merged list is the sum of the lengths of the two lists.

We will do the worst case analysis.

Count comparisons of list elements.

MERGE:

We put two sorted lists, *list 1* and *list 2* (in this case of equal length) on separate stacks with the smallest element of each list on the top of the stack.

- Compare the elements on the top of each stack.
- Pop the smallest one and put it in the *new list*.
- Continue until at least one stack is empty.
- If the other stack is not empty, pop the remaining elements and put them in the *new list*.

Example:

<i>list 1</i>	<i>list 2</i>
1	2
2	4
5	6
6	8

- Pop list 1, *new list* = {1}

list 1

2
5
6

list 2

2
4
6
8

• Pop either list since the top elements are equal.
Assume list 1, *new list* = {1, 2}

list 1

5
6

list 2

2
4
6
8

• Pop list 2, *new list* = {1, 2, 2}

list 1

5
6

list 2

4
6
8

• Pop list 2, *new list* = {1, 2, 2, 4}

list 1

5
6

list 2

6
8

• Pop list 1, *new list* = {1, 2, 2, 4, 5}

list 1

list 2

6

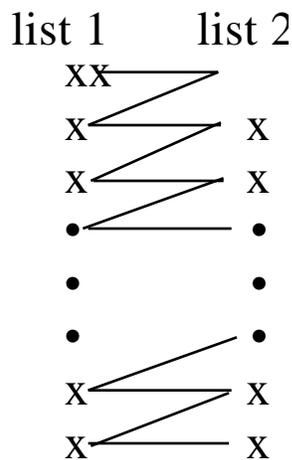
6

8

- Pop list 1, *new list* = {1, 2, 2, 4, 5, 6}

List 1 is empty. Put remaining elements of *list 2* on the end of the sorted *new list* = {1, 2, 2, 4, 5, 6, 6, 8}.

Worst case merge: k elements in each list.



Each line represents a comparison.

We pop an element off the opposite stack each time.

Maximum number of comparisons is defined by the recurrence relation

$$a_k = a_{k-1} + 2, a_1 = 1$$

Solution:

$$a_k = 2k - 1 = O(k) = \text{linear complexity.}$$

MERGESORT:

```
procedure MERGESORT(list)  
/* a famous sorting algorithm */  
  
if (Length (list) = 1) then return list  
  
else  
  
  list 1 = MERGESORT(first half of list)  
  list 2 = MERGESORT(second half of list)  
  
return MERGE(list 1, list 2)
```

The recurrence relation becomes

$$a_n = 2a_{n/2} + n - 1, a_1 = 0$$

You show that

$$a_n \text{ is } O(n \log_2 n)$$
