Colorado State University

CS 320 Fall 2021
Solving recurrences for divide & conquer

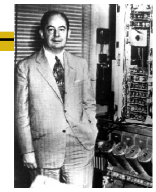
Sanjay Rajopadhye
Colorado State University

Divide & Conquer

- Break up the problem into (multiple, smaller) parts
- Solve each of the parts recursively
- Combine the solution of each of the parts into a solution of the original problem

Colorado State University 2

First example: Merge sort



John von Neumann (1945)

- Divide the array into two halves
- Recursively sort each half
- Merge the two sorted halves

Analysis

Divide $O(1)$

Merge $O(n)$

What about the recursive calls?

$2\,T\left(\frac{n}{2}\right)$

A L G O R I T H M S

A L G O R   I T H M S

A G L O R   H I M S T

A G H I L M O R S T

Colorado State University 3

Complexity of merge

- Time: $O(n)$
- Space: $O(n)$
  - Often with two arrays of length n
  - Can you do (a constant factor) better?

Colorado State University 4

Recurrence relations

- A recurrence relation for a sequence, $\{a_n\}$ is and equation that expresses a_n in terms of one or more of the previous elements of the sequence, $a_1, a_2, \ldots\ a_{n-1}$
- There may be base cases, and the equation hold for $n \geq n_0$ for some constant n_0
  - Example: $a_n = 2a_{n-1} + 1$ and $a_1 = 1$
  - After setting up the recurrence relation, we solve it

Colorado State University 5

Recurrence relation for Merge-sort

- Define the number of comparisons to sort an input of length *n* as: $T(n)$
- Use the structure of the D&C algorithm to define an equation/relation for $T(n)$

$$T(n) \leq \begin{cases} c & \text{if } n = 1 \\ T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + T\left(\left\lceil \frac{n}{2} \right\rceil\right) + cn & \text{otherwise} \end{cases}$$

Colorado State University 6

Solving the Recurrence

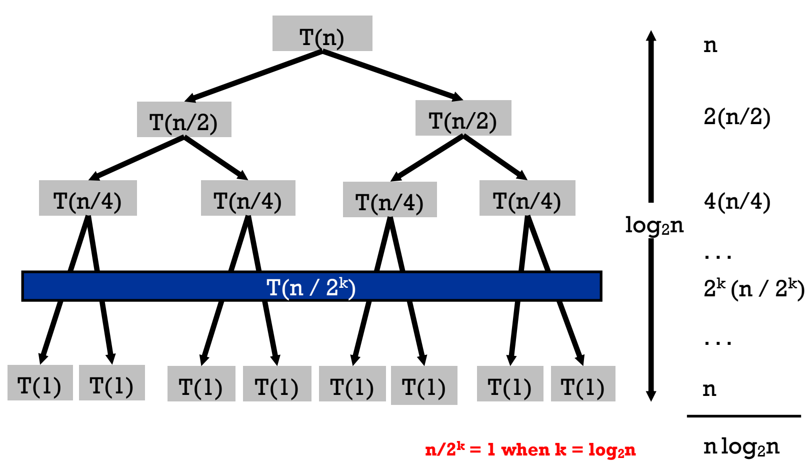
$$T(n) = \begin{cases} c & \text{if } n = 1 \\ 2T\left(\frac{n}{2}\right) + cn & \text{otherwise} \end{cases}$$

- Solution:

$$T(n) = \Theta(n \log n)$$

- Number of techniques
  - Unrolling the recurrence
  - Repeated substitution
  - See a pattern, guess and then prove by induction

Unrolling $T(n) = \begin{cases} c & \text{if } n = 1 \\ 2T\left(\frac{n}{2}\right) + cn & \text{otherwise} \end{cases}$



Seeing the pattern

- What is the "label" of each node?
- When does the label become "small enough" (base case)
- How many levels in the tree? [Hint: use the above two]
- How many nodes at each level?
- What is the "contribution" of each node?
- What is the contribution of each level?
- How many leaves?
- Contribution of the leaves (different from contribution of other levels)

Colorado State University 9

Repeated substitution for $T(n) = \begin{cases} c & \text{if } n = 1 \\ 2T\left(\frac{n}{2}\right) + cn & \text{otherwise} \end{cases}$

- Claim: $T(n) = cn \log_2 n)$

$$
\begin{aligned}
T(n) &= 2T(n/2) \quad + cn \\
&= 4T(n/4) \quad + cn + 2cn/2 \\
&= 8T(n/8) \quad + cn + cn + 4cn/4 \\
&\dots \\
&= 2^{\log_2 n} T(1) \quad + \underbrace{cn + \cdots + cn}_{\log_2 n} \\
&= O(n \log_2 n)
\end{aligned}
$$

← This reaches T(1) when $n = 2^{\log_2 n}$ by definition of $\log_2 n$

Colorado State University 10

Towers of Hanoi

- Move all disks to third peg, without ever placing a larger disk on a smaller one.
- What's the recurrence relation? $a_n = 2a_{n-1} + 1$ with the base case that $a_1 = 1$
- Let's solve by repeated substitution
  - Plug in the definition
  - Do the algebra to collect all the non-recursive expressions together
  - Identify a pattern
  - Determine how many times the pattern occurs until we hit the base case

Colorado State University 11

Hanoi by repeated substitution

$$T(n) = 2T(n-1) + 1$$

$$= 2(2T(n-2) + 1) + 1$$
$$= 4T(n-2) + 2 + 1$$
$$= 4(2T(n-3) + 1) + 2 + 1$$
$$= 8T(n-3) + 4 + 2 + 1$$

- What is the label and how is it changing?
- What about the other terms?
- When do we hit the base case?

Colorado State University 12

Hanoi by repeated substitution

$$T(n) = 2T(n-1) + 1$$

$$\begin{aligned} &= 2(2T(n-2)+1)+1 \\ &= 4T(n-2)+2+1 \\ &= 4(2T(n-3)+1)+2+1 \\ &= 8T(n-3)+4+2+1 \\ &\vdots \\ &= 2^i T(n-i) + \sum_{j=0}^{i-1} 2^j \end{aligned}$$

- When does the label become 1?
- When $i = n-1$ So our solution is

Hanoi by repeated substitution

$$T(n) = 2^{n-1}T(1) + \sum_{j=0}^{n-2} 2^j$$

$$= \sum_{j=0}^{n-1} 2^j = 2^n - 1 = \Theta(2^n)$$

- This is a geometric series

Binary search

```
function BS(x, start, end)
  if (end <= start)
    return A[start]
  mid = (end + start)/2
  if A[mid] < x
    return BS(x, mid, end)
  return BS(x, start, mid-1)
```

- What is the recurrence?
- Apply repeated substitution (on doc cam or exercise)

Find max in an unsorted array

Algorithm:

- Base case n=1
- Otherwise: find the max of the two halves, and return the max of that

```
function FM(start, end)
  if (end = start)
    return A[start]
  mid = (end + start)/2
  return max( FM(start, mid-1), FM(mid, end) )
```

Find max in an unsorted array

Recurrence: base case: $T(1) = 0$

Otherwise: $T(n) = 2T\left(\frac{n}{2}\right) + 1$

$$= 4T\left(\frac{n}{4}\right) + 2 + 1$$
$$= 8T\left(\frac{n}{8}\right) + 4 + 2 + 1$$
$$\vdots$$
$$= 2^k T\left(\frac{n}{2^k}\right) + 2^{k-1} + 2^{k-2} + \cdots 2^0$$
$$= 2^k T\left(\frac{n}{2^k}\right) + 2 \cdot 2^{k-1} - 1$$
$$= 2^k T\left(\frac{n}{2^k}\right) + 2^k - 1$$

Bae case is reached when $2^k = n$, i.e., $k = \log_2 n$, So

$T(n) = 0 + 2^{\log n} - 1 = n - 1$

Colorado State University 17

Another example

```
function foo(A, B) // the size of A is n
  if (n == 1):
  return fuzz(A, B) // base case, fuzz is
constant time

// Process A to build two parts, A0 and A1 of
size n/2 each

C0 = foo (A0, B)
C1 = foo (A0, B)
return buzz(C0, C1) // buzz is O(n^2)
```

Colorado State University 18

General D&C

```
function foo(parameters) // the size of A is n
  if (n <= b):  // base case
  return fuzz(A, B) // base case
// Divide input into a parts, each of size n/b
  divide()
// Make a calls to
  foo(new parameters) // size is n/b
  return combine(r1, ra)
// Complexity of divide and combine is O(n^d)
```

Colorado State University 19

Master Theorem

- Let $a \geq 1, b > 1, n = b^k$ and $T(n)$ be given by

  $T(n) = aT\left(\frac{n}{b}\right) + cn^d$
- The solution of the recurrence is

$$T(n) = \begin{cases} O(n^d) & \text{if } a < b^d \\ O(n^d \log n) & \text{if } a = b^d \\ O(n^{\log_b a}) & \text{if } a > b^d \end{cases}$$

Colorado State University 20

Merge-sort by master theorem

- $a = 2, b = 2, d = 1$
- So, $b^d = 2 = a$

… and the solution is

$$T(n) = O(n^d \log n) = O(n \log n)$$