CS3383 Unit 1, Lecture 1: Divide and conquer intro

David Bremner

Divide and conquer

- Big Picture
- Merge Sort
- Recursion tree
- Integer Multiplication

unit prereqs

- mergesort
- geometric series (CLRS A.5)

/Subtype /Text/F 1/T (Video)/Contents (video/10.2-prereq.mkv)

Structure of divide and conquer

function SOLVE(P)
  if $|P|$ is small **then**
    SolveDirectly(P)
  else
    $P_1 \ldots P_k = \text{Partition}(P)$
    for $i = 1 \ldots k$ **do**
      $S_i = \text{Solve}(P_i)$
    end for
    Combine($S_1 \ldots S_k$)
  end if
end function

- Where is the actual work?
- How many subproblems?
- How big are the subproblems?

/Subtype /Text/F 1/T
(Video)/Contents
(video/10.3-structu[redacted]

Merge sort

```
MergeSort(A[1...n]):
    if (n == 1):
        return A
    left = MergeSort(A[1...⌈n/2⌉])
    right = MergeSort(A[⌈n/2⌉+1...n])
    return Merge(left, right)
```

- non-recursive cost is in merging (and splitting) arrays
- can be done in $\Theta(n)$ time

/Subtype /Text/F 1/T (Video)/Contents (video/10.4-merge-sort.mkv)

Recurrence for merge sort

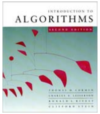
```
1 def MergeSort(A[1...n]):
2     if (n == 1):
3         return A
4     left = MergeSort(A[1...⌈n/2⌉])
5     right = MergeSort(A[⌈n/2⌉+1...n])
6     return Merge(left, right)
```

(line 4) $$T(n) = T(n/2)$$

(line 5) $$+\, T(n/2)$$

(line 6) $$+\, \Theta(n)$$

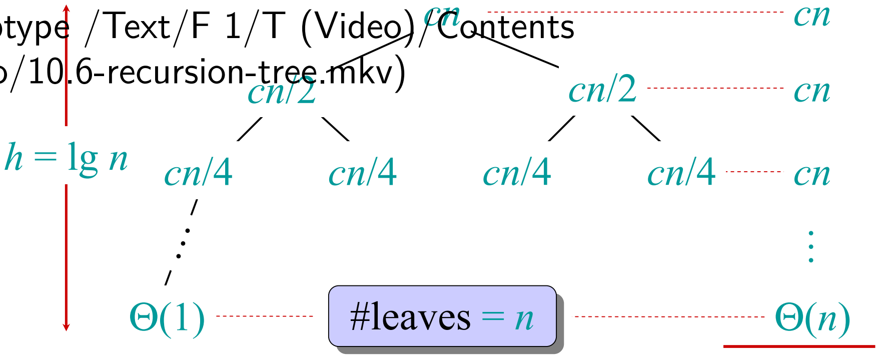
/Subtype /Text/F 1/T (Video)/Contents
(video/10.5 merge sort 2.mkv)



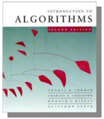
Recursion tree

Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.

/Subtype /Text/F 1/T (Video)/Contents (video/10.6-recursion-tree.mkv)



Total $= \Theta(n \lg n)$



Appendix: geometric series

/Subtype /Text/F 1/T (Video)/Contents (video/10.7-geometric-series.mkv)

$$1 + x + x^2 + \cdots + x^n = \frac{1 - x^{n+1}}{1 - x} \quad \text{for } x \neq 1$$

$$1 + x + x^2 + \cdots = \frac{1}{1 - x} \quad \text{for } |x| < 1$$

Return to last slide viewed.

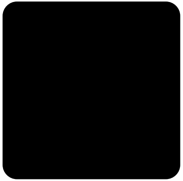
Integer Multiplication

The Problem

Input	positive integers x and y, each n bits long
Output	positive integer z where $z = x \cdot y$

- A straightforward approach using base-2 arithmetic, akin to how we multiply by hand, takes $\Theta(n^2)$ time.
- Can we do better with divide and conquer?

/Subtype /Text/F 1/T (Video)/Contents (video/10.8-integer-mult.mkv)



Splitting the input

Split the bitstrings in half, generating x_L, x_R, y_L, y_R such that

$$x = 2^{\frac{n}{2}} \cdot x_L + x_R$$
$$y = 2^{\frac{n}{2}} \cdot y_L + y_R\,.$$

- Like base $2^{\lfloor \frac{n}{2} \rfloor}$
- Assume that n is a power of 2, so $\frac{n}{2}$ will always be integer.

/Subtype /Text/F 1/T (Video)/Contents (video/10.9-base-n-2.mkv)

A first approach

Express our multiplication of the n-bit integers as four multiplications of $\frac{n}{2}$-bit integers:

$$\begin{aligned} x \cdot y &= (2^{\frac{n}{2}} \cdot x_L + x_R) \cdot (2^{\frac{n}{2}} \cdot y_L + y_R) \\ &= 2^n \cdot x_L y_L + 2^{\frac{n}{2}} \cdot (x_L y_R + x_R y_L) + x_R y_R \end{aligned}$$

This gives a recurrence of

$$T(n) \;=\; 4T\left(\frac{n}{2}\right) + cn$$

/Subtype /Text/F 1/T (Video)/Contents (video/10.A-first-approach.mkv)

A first approach

Express our multiplication of the n-bit integers as four multiplications of $\frac{n}{2}$-bit integers:

$$\begin{aligned} x \cdot y &= (2^{\frac{n}{2}} \cdot x_L + x_R) \cdot (2^{\frac{n}{2}} \cdot y_L + y_R) \\ &= 2^n \cdot x_L y_L + 2^{\frac{n}{2}} \cdot (x_L y_R + x_R y_L) + x_R y_R \end{aligned}$$

This gives a recurrence of

$$T(n) \;=\; 4T\left(\frac{n}{2}\right) + cn$$

/Subtype /Text/F 1/T (Video)/Contents (video/10.B-bad-news.mkv)

Bad news

Finding a better recurrence / algorithm.

We want to compute

$$2^n \cdot x_L y_L + 2^{\frac{n}{2}} \cdot (x_L y_R + x_R y_L) + x_R y_R$$

- Can we compute $(x_L y_R + x_R y_L)$, the coefficient of $2^{\frac{n}{2}}$, more efficiently?
- How about re-using $x_L y_L$ and $x_R y_R$?

/Subtype /Text/F 1/T (Video)/Contents (video/10.C-better [redacted]

Gauss's trick

From the binomial expansion

$$(x_L + x_R)(y_L + y_R) = x_L y_L + x_L y_R + x_R y_L + y_R x_R$$

we get that

$$x_L y_R + x_R y_L \; = \; (x_L + x_R)(y_L + y_R) - x_L y_L - x_R y_R$$

/Subtype /Text/F 1/T (Video)/Contents (video/10.D-gauss.[redacted]

Recursive Algorithm

To compute

$$2^n \cdot x_L y_L + 2^{\frac{n}{2}} \cdot (x_L y_R + x_R y_L) + x_R y_R$$

1. find x_L, x_R, y_L, y_R and $x_L + x_R, y_L + y_R$ [$O(n)$]
2. find $x_L y_L$, $x_R y_R$, and $(x_L + x_R)(y_L + y_R)$ recursively
3. and assemble the results in linear time

/Subtype /Text /F1/T (Wait)/Contents (2)

Recursive Algorithm

To compute

$$2^n \cdot x_L y_L + 2^{\frac{n}{2}} \cdot (x_L y_R + x_R y_L) + x_R y_R$$

1. find x_L, x_R, y_L, y_R and $x_L + x_R, y_L + y_R$ [$O(n)$]
2. find $x_L y_L$, $x_R y_R$, and $(x_L + x_R)(y_L + y_R)$ recursively
3. and assemble the results in linear time

Roughly speaking, the recurrence is

$$T(n) \approx 3T\left(\frac{n}{2}\right) + cn$$

- one subproblem is actually one bit bigger. Does it matter?