lecture 17

Divide and Conquer

- Karatsuba multiplication
  https://class.coursera.org/algo-004/lecture/167
- The Master Method
  https://class.coursera.org/algo-004/lecture

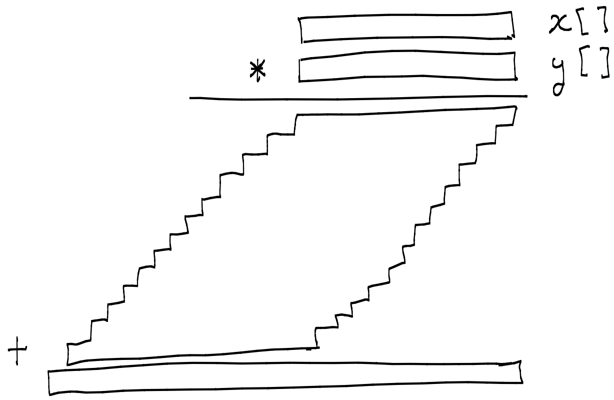
Grade School Addition

```
  352        x[n]
+ 964        y[n]
-----      ---------
 1316      sum[n+1]
```

Addition of two n digit numbers takes time $O(n)$.

Grade School Multiplication

```
      352         x[n]
    x 964         y[n]
   ------
     1408    }
    2112     }  tmp[n][2n]
   3168      }
   ------
   339328         r[2n]
```



If x and y have n digits each, then computing $x * y$ is $O(n^2)$.

Given n digit numbers x[], y[]

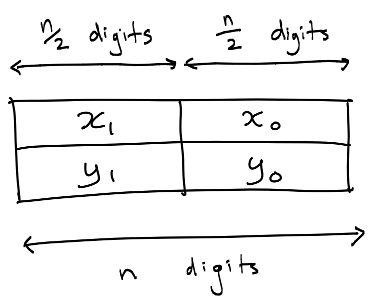
```
for i = 1 to n                 // 2 for loops =>
   for j = 1 to n {            // O(n^2)
      tmp = x[i] * y[j]
      etc.
```

(If you want to see all the pseudocode for x[] * y[] see my COMP 250 lecture 1
http://www.cim.mcgill.ca/~langer/250/1-gradeschool-slides.pdf)

}

Is there a faster multiplication algorithm?

Idea: divide and conquer!



$$x = x_1 * 10^{n/2} + x_0$$
$$y = y_1 * 10^{n/2} + y_0$$

e.g. $3527 = 3500 + 27$
$= 35 \times 10^2 + 27$

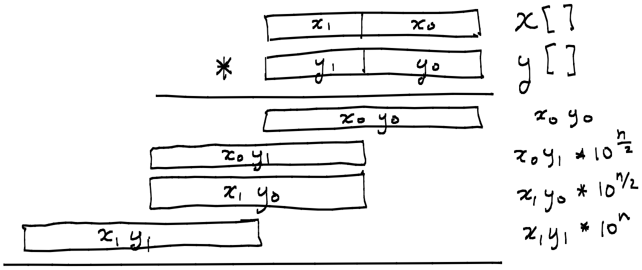
$x = $ | x_1 | x_0 |
$y = $ | y_1 | y_0 |

$x =$	x_1	x_0
$y =$	y_1	y_0

$$x * y$$

$$= (x_1 * 10^{\frac{n}{2}} + x_0) * (y_1 * 10^{\frac{n}{2}} + y_0)$$

$$= x_1 y_1 * 10^n + (x_0 y_1 + x_1 y_0) * 10^{n/2} + x_0 y_0$$



Note:

$* 10^{n/2}$ shifts left by $\frac{n}{2}$ positions $\Rightarrow O(n)$

$* 10^n$ " " " n positions $\Rightarrow O(n)$

(and filling with 0's).

Let $t(n)$ be the time required to multiply two n digit numbers.

$$x * y$$

$$= x_1 y_1 * 10^n + (x_0 y_1 + x_1 y_0) * 10^{n/2} + x_0 y_0$$

$t(\frac{n}{2})$ $t(\frac{n}{2})$ $t(\frac{n}{2})$ $t(\frac{n}{2})$

Thus

$$t(n) = 4\, t\left(\frac{n}{2}\right) + cn$$

However, it can be shown using back substitution that ...
(see Master Method later today)

$$t(n) = 4\, t\left(\frac{n}{2}\right) + cn$$

$$= 4 \left\{ 4\, t\left(\frac{n}{4}\right) + c\frac{n}{2} \right\} + cn$$

$$= \vdots$$

$$= O(n^2)$$

$\leftarrow$ no benefit over grade school method

Trick (Karatsuba):

$x =$	x_1	x_0
$y =$	y_1	y_0

$$x * y$$

$$= x_1 y_1 * 10^n + (x_0 y_1 + x_1 y_0) * 10^{n/2} + x_0 y_0$$

We don't need to know $x_0 y_1$ and $x_1 y_0$ individually. We just need to know their sum, and (next slide) their sum can be computed using $x_0 y_0$ and $x_1 y_1$ and one other product i.e. three multiplications in $O(n)$.

$$x * y$$

$$= x_1 y_1 * 10^n + (x_0 y_1 + x_1 y_0) * 10^{n/2} + x_0 y_0$$

$$(x_0 y_1 + x_1 y_0) = (x_1 + x_0)(y_1 + y_0) - x_1 y_1 - x_0 y_0$$

Thus,

$$t(n) = 3\, t\left(\frac{n}{2}\right) + cn.$$

$\uparrow$ different c than before

Recurrence	Example	Complexity
$t(n) = t(\frac{n}{2}) + c$	binary search	$O(\log n)$
$t(n) = 2\,t(\frac{n}{2}) + cn$	mergesort, closest pair of 2D points	$O(n \log n)$
$t(n) = 3\,t(\frac{n}{2}) + cn$	Karatsuba multiplication	$O(\,?\,)$
$t(n) = 4\,t(\frac{n}{2}) + cn$	failed attempt at fast multiplication	$O(n^2)$

lecture 17

Divide and Conquer

- Karatsuba multiplication

  https://class.coursera.org/algo-004/lecture/167

- The Master Method

  https://class.coursera.org/algo-004/lecture

Suppose we have a divide and conquer algorithm that gives a recurrence :

$$t(n) = a\, t(\tfrac{n}{b}) + c\, n^d$$

- a is the number of subproblems
- $\frac{n}{b}$ is the size of each subproblem
- n^d is the overhead for problem of size n (to partition and combine solutions)

I'll set $c = 1$.

$$t(n) = a\, t(\tfrac{n}{b}) + c\, n^d$$

Examples :

- binary search

  $a = 1$, $b = 2$, $c = 1$, $d = 0$

- merge sort

  $a = 2$, $b = 2$, $c = 1$, $d = 1$

- Karatsuba multiplication

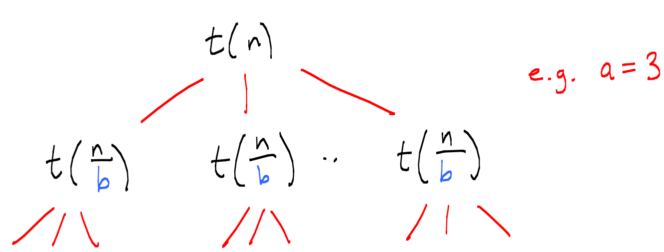
  $a = 3$, $b = 2$, $c = 1$, $d = 1$

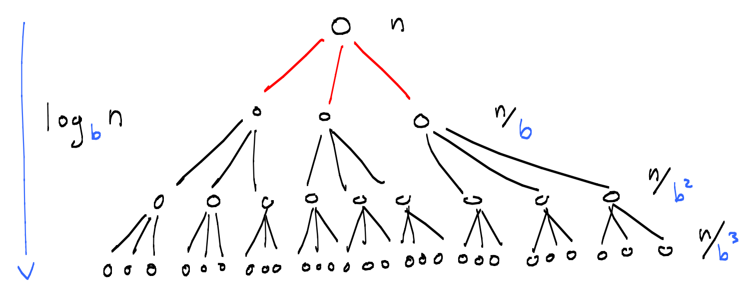
- failed attempt at fast multiplication

  $a = 4$, $b = 2$, $c = 1$, $d = 1$

$$t(n) = a\, t(\tfrac{n}{b}) + c\, n^d$$

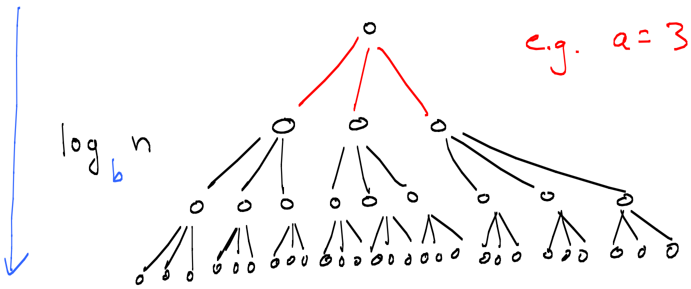
- Each node of the call tree has a children (a is the "branching factor" of the call tree)
- The problem size is reduced by factor b .





height of call tree $= \log_b n$

Recursion stops at the base case, typically when problem size is a small number e.g. 1 .



number of leaves (base case of recursion)

$$= a^{\log_b n}$$

$$t(n) = a\, t\left(\frac{n}{b}\right) + c\, n^d$$

assume $c = 1$

Tim Roughgarden:

"the forces of good and evil"

good – the size of each subproblem shrinks with each recursive call $(b > 1)$

evil – the number of subproblems increases at each level of the call tree. $(a > 1)$

– what about d ?

Let the battle begin ...

Assume $n = b^k$ for simplicity.

$$t(n) = a\, t\left(\frac{n}{b}\right) + n^d$$

$$= a\left[a\, t\left(\frac{n}{b^2}\right) + \left(\frac{n}{b}\right)^d \right] + n^d$$

level 2

$$= a^2\, t\left(\frac{n}{b^2}\right) + a\left(\frac{n}{b}\right)^d + n^d$$

$$a\, t\left(\frac{n}{b^3}\right) + \left(\frac{n}{b^2}\right)^d$$

level 3

$$= a^3\, t\left(\frac{n}{b^3}\right) + a^2\left(\frac{n}{b^2}\right)^d + a\left(\frac{n}{b}\right)^d + n^d$$

level k

$$= a^k\, t\left(\frac{n}{b^k}\right) + \sum_{i=0}^{k-1} a^i \left(\frac{n}{b^i}\right)^d$$

level $\log_b n$

$$= a^{\log_b n}\, t(1) + \sum_{i=0}^{\log_b n - 1} a^i \left(\frac{n}{b^i}\right)^d$$

$a^{\log_b n}$: number of leaves

$t(1)$: work at each leaf

a^i: number of internal nodes at level i

$\left(\frac{n}{b^i}\right)^d$: work at each internal node at level i

$$t(n) = a^{\log_b n}\, t(1) + \sum_{i=0}^{\log_b n - 1} a^i \left(\frac{n}{b^i}\right)^d$$

$$= n^d \sum_{i=0}^{\log_b n} \left(\frac{a}{b^d}\right)^i$$

Assume $t(1) = 1$, and note $\frac{n}{b^{\log_b n}} = 1$.

$$t(n) = n^d \sum_{i=0}^{\log_b n} \left(\frac{a}{b^d}\right)^i$$

geometric series

$$\sum_{i=0}^{k} r^i$$

three cases

1) $r = 1$
2) $r < 1$
3) $r > 1$

Case 1 $(r=1)$: $a = b^d$

Here we have the same amount of work at each level.

$$1 + r + r^2 + r^3 + \dots r^k$$

$$= 1 + 1 + 1 + 1 + \dots 1$$

$$= k + 1$$

$$= \log_b n + 1$$

$$\Rightarrow \quad t(n) = O(n^d \log_b n)$$

eg. mergesort

$$t(n) = a\, t\left(\frac{n}{b}\right) + c n^d$$

$a = 2$, $b = 2$, $d = 1$

$$t(n) = O(n^d \log_b n) = O(n \log_2 n)$$

The same amount of total work done at each level i, namely $O(n)$.

case 2 $(r<1)$: $a < b^d$

Here we have a decreasing amount of work at each level.

$$1 + r + r^2 + r^3 + \dots r^k$$

$$= \frac{1 - r^{k+1}}{1 - r}$$

$$< \frac{1}{1-r}, \quad \text{if } r < 1$$

$$= \text{constant} \quad (\text{independent of } n)$$

$$\Rightarrow \quad t(n) = O(n^d)$$

case 3 $(r>1)$: $a > b^d$ $\quad \left(r = \frac{a}{b^d} > 1\right)$

Here we have an increasing amount of work to do at each level. The leaves dominate.

$$1 + r + r^2 + r^3 + \dots r^k$$

$$= \frac{r^{k+1} - 1}{r - 1}$$

$$< c\, r^k, \quad \text{for some } c \text{ which depends on } r$$

$$r^k = \left(\frac{a}{b^d}\right)^k$$

let $k = \log_b n$

$$= \left(\frac{a}{b^d}\right)^{\log_b n}$$

$$= a^{\log_b n} \Big/ (b^d)^{\log_b n}$$

$$= n^{\log_b a} / n^d$$

see end of lecture

$$t(n) = n^d \sum_{i=0}^{\log_b n} r^i$$

$$< n^d \, c \, r^{\log_b n}$$

$$= n^d \, c \left(\frac{a}{b^d}\right)^{\log_b n}$$

$$< n^d \cdot c \, \frac{n^{\log_b a}}{n^d}$$

$$= c \, n^{\log_b a}$$

e.g. Karatsuba multiplication

$$t(n) = a\, t\left(\frac{n}{b}\right) + c n^d$$

$a = 3$, $b = 2$, $d = 1$ $\qquad r = \frac{a}{b^d} > 1$

$$t(n) = O(n^{\log_2 3}) \approx O(n^{1.6})$$

Master Method (Summary):

$$t(n) = a\, t\left(\frac{n}{b}\right) + n^d, \quad t(1) = 1$$

$t(n)$ is
- $O(n^d \log_b n)$, $a = b^d$ — same work at each level
- $O(n^d)$, $a < b^d$ — root dominates
- $O(n^{\log_b a})$, $a > b^d$ — leaves dominate

Review of exponents and logs

$$x^{(yz)} = (x^y)^z \neq x^{(y^z)}$$

e.g. $x^{2 \cdot 3} = (x^2)^3 = (x^2)(x^2)(x^2) = x^6 \qquad \neq \qquad x^{(2^3)} = x^8$

e.g. $$(b^d)^{\log_b n} = b^{d \log_b n} = (b^{\log_b n})^d = n^d$$

Review of exponents and logs

for any $a, b, x > 0$

$$\log_b x = \log_b a \cdot \log_a x$$

Why?

$$x \equiv a^{\log_a x}$$

take $\log_b$ of both sides

$$\log_b x = \log_b \left(a^{\log_a x}\right)$$

$$= \log_a x \cdot \log_b a$$

Claim: $a^{\log_b n} = n^{\log_b a}$

Proof:

$$a^{\log_b n} = a^{\log_b a \, \log_a n}$$
$$= \left(a^{\log_a n}\right)^{\log_b a}$$
$$= n^{\log_b a}$$

STUDY BREAK

Divide and Conquer

16. closest pair of points in 2D (PDF)
    mergesort recurrence, closest pair in O(n log n)
17. Karatsuba multiplication, Master method
18. quicksort & quickselect (deterministic)
    median-of-median-of-5, O(n) for select, O(n log n) for sort

Probability and information theory

19. random variables and expectation
    expectation vs. amortization, linearity of expectation
20. quicksort and quick select (randomized)
    random input vs. randomized algorithm, average case analysis
21. lower bounds for comparison sorting, sorting in linear time
    decision trees, counting sort, bucket sort
22. data compression
    coding and entropy, run length coding, Huffman coding

Wrapup

23. review for final exam 1 (material covered in midterms 1 and 2)
24. review for final exam 2 (material covered after midterm 2)

Roughgarden
https://class.coursera.org/algo-004/lecture
weeks 2 & 3