CSCI 3100: Algorithms

Lecture 4:
Recursion (cont)

Today:

- HW0 due
- HW1 posted - due next week
- More on recursion:

  Questions from the reading?

- No office hours today

Last time (reading):

Quick sort

Any questions?

Takeaway:

Sorting is key CS problem.

Today: Multiplication

In general, we say this is $O(1)$ time → **lies!**

In reality:

```
         31415962
       × 27182818
       ----------
        251327696
        31415962
      251327696
      62831924
    251327696
    31415962
   219911734
   62831924
   --------------
   853974377340916
```

Runtime: $O(n^2)$
2-n-bit #s

How to formalize?
(to a computer)

Runtime? (2- n-bit #s)

Better: A trick:

$$(10^m a + b)(10^m c + d)$$
$$= 10^{2m} ac + 10^m (bc + ad) + bd$$

pf of correctness

Example $\left[\begin{matrix} 963{,}245 \\ 624{,}197 \end{matrix}\right\}$ + $m = 3$:

$$\left(\overset{a}{963} \cdot 10^3 + \overset{b}{245}\right) \times \left(\overset{c}{624} \cdot 10^3 + \overset{d}{197}\right)$$

$$= 10^6 \cdot (963 \times 624) + 10^3 \cdot (245 \cdot 624 + 963 \cdot 197) + \ldots$$

Make this an algorithm:

$M(n)$

$\underline{\text{MULTIPLY}(x, y, n):}$
 if $n = 1$
  return $x \cdot y$ — $O(1)$
 else
  $m \leftarrow \lceil n/2 \rceil$ ← $O(1)$
  $a \leftarrow \lfloor x/10^m \rfloor$; $b \leftarrow x \bmod 10^m$
  $d \leftarrow \lfloor y/10^m \rfloor$; $c \leftarrow y \bmod 10^m$ — $O(1)$
  $e \leftarrow \text{MULTIPLY}(a, c, m)$ ← $M(\frac{n}{2})$
  $f \leftarrow \text{MULTIPLY}(b, d, m)$ ''
  $g \leftarrow \text{MULTIPLY}(b, c, m)$ ''
  $h \leftarrow \text{MULTIPLY}(a, d, m)$ ''
  return $10^{2m}e + 10^m(g + h) + f$

1 addition

2 more

Runtime:

$M(n) = 4M(\frac{n}{2}) + O(1)$

$n^{\log_b a} = n^2$ vs $O(1)$

(by Master thm)

$\Rightarrow M(n) = \Theta(n^2)$

(No better!)

Hrm - not better after all...

Another trick!

$$ac + bd - \underbrace{(a-b)(c-d)}_{[ac - bc - ad + bd]} = \underbrace{bc + ad}$$

Huh?

Recall:

$$(10^m a + b)(10^m c + d)$$
$$= 10^{2m} ac + 10^m (bc + ad) + bd$$

Conclusion:

By black magic of algebra, I can get 4 multiplied value by only doing 3 multiplications.

New & improved pseudocode:

```
FASTMULTIPLY(x, y, n):
    if n = 1
        return x · y
    else
        m ← ⌈n/2⌉
        a ← ⌊x/10^m⌋;  b ← x mod 10^m
        d ← ⌊y/10^m⌋;  c ← y mod 10^m
        e ← FASTMULTIPLY(a, c, m)
        f ← FASTMULTIPLY(b, d, m)
        g ← FASTMULTIPLY(a − b, c − d, m)
        return 10^{2m} e + 10^m (e + f − g) + f
```

$O(1)$ total

Analysis: $M(n) = 3M(\frac{n}{2}) + O(1)$

$n^{\log_b a} = n^{\log_2 3} \ll n^2$

$M(n) = O(n^{\log_2 3}) \approx n^{1.5\ldots}$

Some comments

- In practice, done in base 2, not 10.

- Actually, this can break down even more!

  If we apply another recursive layer, can get $O(n \log n)$ eventually.

  (Ever heard of Fast Fourier transforms?)

  ↳ see Lect. notes pt 2 if curious

Here ends lect. notes #1

Another recursive strategy:
Backtracking
(lecture notes pt 3)

Idea: Build up a solution iteratively.

Setting: an algorithm needs to try multiple options.

Strategy: Make a recursive call for each possibility.

Downside:
SLOW

First example: Subset Sum

Given a set X of positive integers and a target value t, is there a subset of X which sums to t?

Ex: $X = \{8, 6, 7, 3, 10, 5, 9\}$

$t = 15$

Yes: $8+7$

$10+5$

How would we solve?

recursion!

Consider recursively:

$$X = \{\underline{8}, 6, 7, 5, 3, 1, 9\}$$

in or out?

Formalize this: recursion!

Consider $x \in X$.

In or out? $\rightarrow$ recurse on $t - x$

(check that $\underline{x < t}$)

* $\underline{\text{base case?}}$

if set $= \{\}$, fail

if $t = 0$, success

Pseudocode:

V, n, T

$S(n)$

```
SubsetSum(X[1..n], T):
    if T = 0
        return True
    else if T < 0 or n = 0
        return False
    else
        return (SubsetSum(X[1..n-1], T) ∨ SubsetSum(X[1..n-1], T - X[n]))
```

$O(1)$

$S(n-1)$ $S(n-1)$

X_n is not in subset

OR

X_n is in subset

Runtime:

$$S(n) = 2S(n-1) + O(1)$$

$$S_n = 2S_{n-1} + 8$$

$$(x-2)$$

$$S_n = c \cdot 2^n + O(1)$$

$$\boxed{= O(2^n)}$$

Correctness:

Proof by induction on n, the size of X:

Base case:

if $T=0$, then $\{\} \subset X$

if $n=0$, then T had also better be 0! Otherwise clearly can't hit T

IH: Algorithm works for sets of size $n-1$

IS: Consider set of size n.

Last #, $X[n]$, in X is either in the taget subset of not.

Recurse on both possibilities (so checked all possibilities) ☒

Next time:

On to dynamic programming!