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 true — lies!

In reality:

\[
\begin{array}{c}
31415962 \\
\times 27182818 \\
251327696 \\
31415962 \\
251327696 \\
62831924 \\
251327696 \\
31415962 \\
219911734 \\
62831924 \\
853974377340916
\end{array}
\]

How to formalize?

(To a computer)

Runtime? \( O(n^2) \) & 2-n-bit #s
Better: A trick:

\[(10^m a + b)(10^m c + d)\]

\[= 10^{2m} ac + 10^m (bc+ad) + bd\]

Example: \[\left\{ \begin{array}{c}
963,245 \\\n624,197
\end{array} \right\} + m=3:\]

\[(963 \cdot 10^3 + 245) \\
(624 \cdot 10^3 + 197)\]

\[= 10^6 \cdot (963 	imes 624) + 10^3 \cdot (245 \cdot 624 + 963 \cdot 197) + \ldots\]
Make this an algorithm:

\[
M(n) = \begin{cases} 
  4M\left(\frac{n}{2}\right) + O(1) & \text{if } n \text{ is even} \\
  O(n) & \text{if } n = 1 
\end{cases}
\]

where \( n = \log_{b}a \) and \( \log_{b}a \) is the master theorem.

\[ M(n) = O(n^{2}) \]

(No better!)
Hrm - not better after all...

Another trick!

\[ ac + bd - (a-b)(c-d) = bc + ad \]

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 and improved pseudo code:

```python
FASTMULTIPLY(x, y, n):
    if n = 1
        return x \cdot y
    else
        m \leftarrow \lfloor n/2 \rfloor
        a \leftarrow \lfloor x/10^m \rfloor; b \leftarrow x \mod 10^m
        d \leftarrow \lfloor y/10^m \rfloor; c \leftarrow y \mod 10^m
        e \leftarrow FASTMULTIPLY(a, c, m)
        f \leftarrow FASTMULTIPLY(b, d, m)
        g \leftarrow FASTMULTIPLY(a - b, c - d, m)
        return 10^{2m}e + 10^m(e + f - g) + f
```

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

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

\[ M(n) = O(n^{\log_2 3}) = n^{1.5...} \]
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:

\[ X = \{8, 6, 7, 5, 3, 1, 9\} \]

In or out?

Formalize this: recursion!

Consider \( x \in X \), recurse on \( t \cdot x \)

In or out? (check that \( x < t \))

or base case?

if \( \text{set} = \{3\} \), fail

if \( t = 0 \), success
Pseudocode:

\[
\text{SubsetSum}(X[1..n], T): \\
\text{if } T = 0 \\
\quad \text{return True} \\
\text{else if } T < 0 \text{ or } n = 0 \\
\quad \text{return False} \\
\text{else} \\
\quad \text{return (SubsetSum}(X[1..n-1], T) \lor \text{SubsetSum}(X[1..n-1], T-X[n]))
\]

Runtime:
\[
S(n) = 2S(n-1) + O(1) \\
S_n = 2S_{n-1} + 8 \\
(x-2) \\
S_n = c \cdot 2^n + O(1) \\
\in O(2^n)
\]
Correctness:

Proof by induction on \( n \), the size of \( X \):

**Base case:**
- If \( T = 0 \), then \( \exists \exists C X \)
- 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 \# in \( X[n] \) in \( X \) is either in the target subset or not.
- Recurse on both possibilities.

(50 checked all possibilities)
Next time:

On to dynamic programming!