

CS331, Fall 2026

Lecture 4 (9/2)

Today: — Word RAM
— Largest jump
— Largest subsequence sum
— Balanced parentheses

Fibonacci (Part III, Section 1)

(From last time...)

Input: $n \in \mathbb{N}$ Output: F_n , n^{th} Fibonacci number

First try:

FibNaive(n):

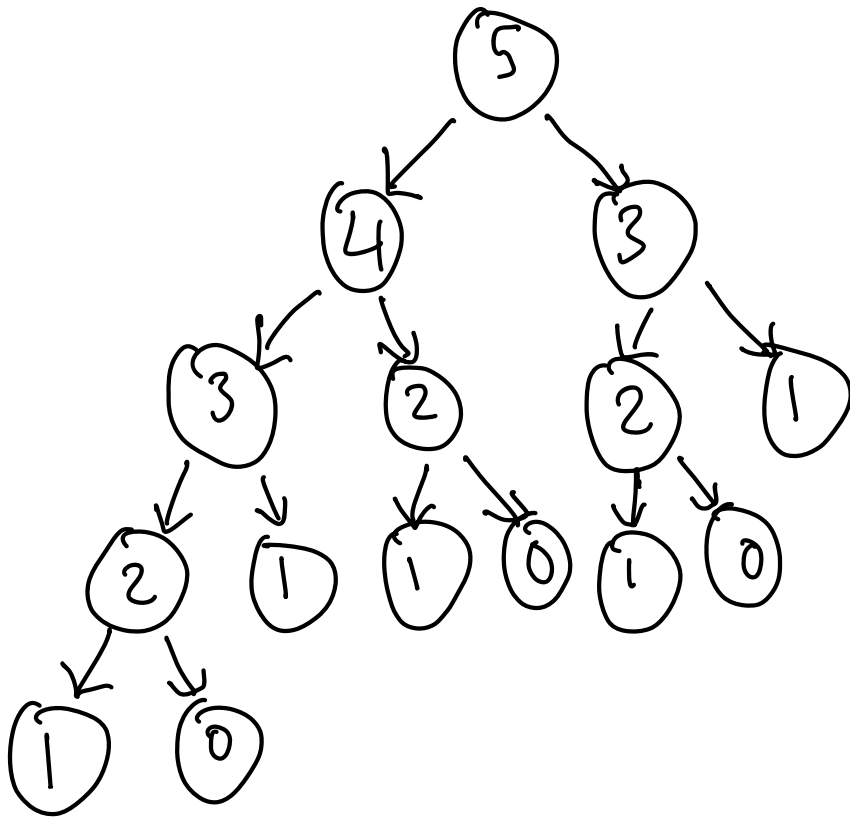
If $n == 0$: Return 1

If $n == 1$: Return 2

Return FibNaive($n-1$) + FibNaive($n-2$)

X Warning: this will take exponential time!

e.g. FibNaive(5):

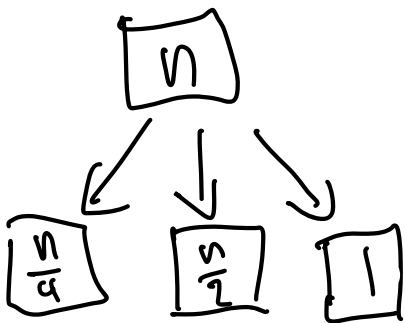


Death = n
branching
factor ≈ 2

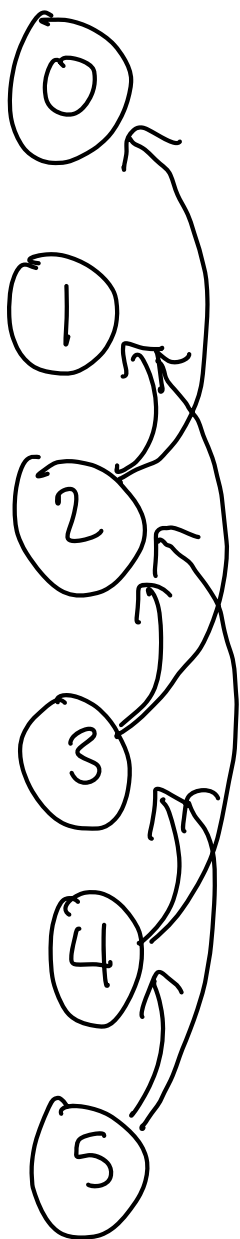
Idea: do things bottom-up to avoid recomputation

Intuition: induction is typically proven from bottom-up.

In recursive algos ("strong induction") multiple larger calls can use the same smaller call.



if every algo uses base case,
save it & reuse it!



Dependency graph for Fib



"Memoized" implementation

Fib(n):

$L \leftarrow \text{Array.init}(n+1)$ // $L[i] = \text{Fib}_i$

$L[1] \leftarrow 1$

$L[2] \leftarrow 2$

For $3 \leq i \leq n+1$: $L[i] \leftarrow L[i-1] + L[i-2]$

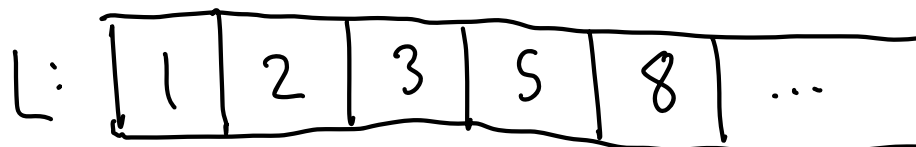
Return $L[n+1]$

Key takeaways:

1) order matters!

(top-down vs. bottom-up)

2) save your work



"memos"

Dynamic programming:

Smarter and more flexible recursion

(up to now, focus is "divide-and-conquer")

Applications:

- Coding interviews (seriously.)
- Reinforcement learning (founder: Bellman)
- Game theory / economics

Basic idea: 1) define subproblems you want to solve / "memoize"

2) define order to solve them

It can be hard to understand at first.

Payoffs enormous: exponential $\Rightarrow$ near-linear time?

Next 4 lectures: many examples as a field guide

Word RAM model (Part I, Section 7)

Warmup: Digits

How many digits is $n \in \mathbb{N}$?

$$\text{Base } 10: 1 + \lfloor \log_{10}(n) \rfloor = O(\log_{10}(n))$$

$$\text{Base } 2: 1 + \lfloor \log_2(n) \rfloor = O(\log_2(n))$$

$$\text{Base } b: 1 + \lfloor \log_b(n) \rfloor = O(\log_b(n))$$

constant

all the same!

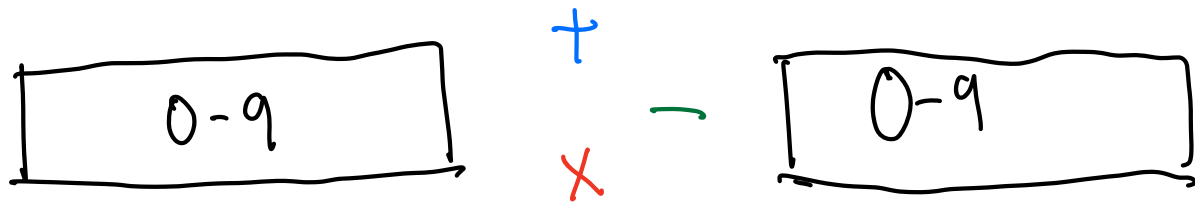
$$\text{Recall that } \log_2(b) \log_b(c) = \log_2(c)$$

$$\text{Proof: } (2^{\log_2(b)})^{\log_b(c)} = b^{\log_b(c)} = c = 2^{\log_2(c)}$$

$$\text{Hence, } \frac{\log_2(n)}{\log_b(n)} = \underbrace{\log_2(b)}_{\text{constant}}$$

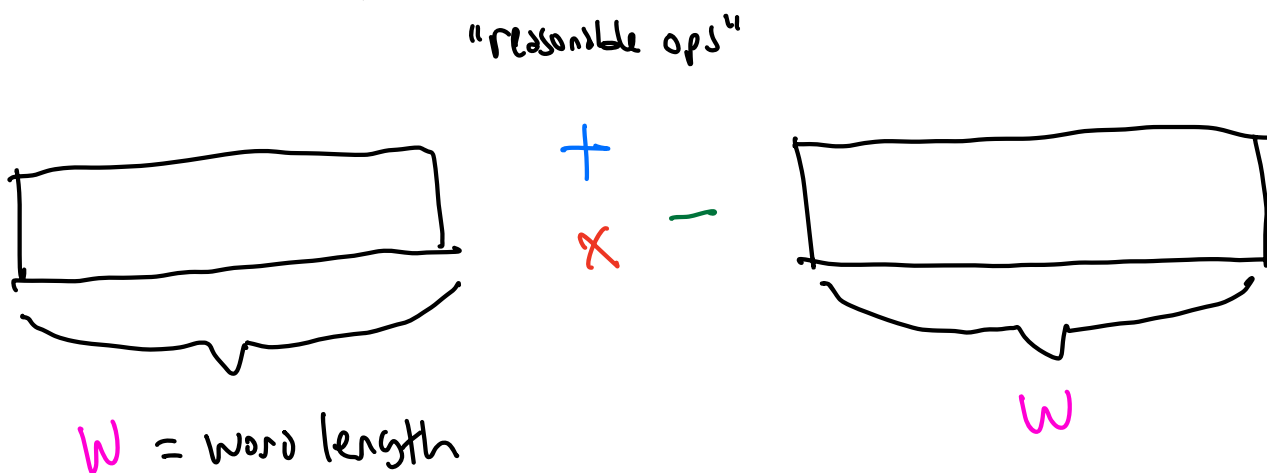
We'll be lazy
& just write
" $\log(n)$ "

Bit complexity model: bit/digit ops take $O(1)$ time.



So, $\underbrace{\text{Fib}(n)}_{O(n) \text{ digits}} + \underbrace{\text{Fib}(n)}_{O(n) \text{ digits}}$ takes $O(n)$ time.

Word RAM model:



Assume: takes $O(1)$ time.

Think: $w = 64$, maybe few 100's tops.

Ok to assume $\log(n) \leq w$.

Will do henceforth
unless stated otherwise

Unreasonable to assume $n \leq w$.

keep in eye out!

Examples

- Incrementing 2 for loop counter
 $i \in (n)$ is $\log(n)$ digits, $O(1)$ time.
- Comparing two numbers in sorting
if both $\leq 2^w$, $O(1)$ time.
by default, OK to assume

Largest jump (Part III, Section 2.1)

Input: L is a list of n elements in $\mathbb{R}$

Output: (i, j) with $1 \leq i \leq j \leq n$

maximizing $L[j] - L[i]$

1	2	3	4	5	6	7
8	12	4	9	17	1	13

Example from last class

Buy ↓ Sell →	1	2	3	4	5	6	7
1	0	4	-4	1	9	-7	5
2		0	-8	-3	5	-11	1
3			0	5	13	-3	9
4				0	8	-8	4
5					0	-16	-4
6						0	12
7							0
Best:	0	4	0	5	13	0	12

Time: $O(n^2)$ (for each i, j , compute $L[j] - L[i]$)

How to improve?

Idea: What do we need to know for $Best[j]$?

$$Best[j] = \max_{i \in [j]} L[j] - L[i] = L[j] - \min_{i \in [j]} L[i]$$



Faster algo, take 1:

(1) Pass over list, maintain running min.

e.g.

8	12	4	9	17	1	13
---	----	---	---	----	---	----

MinUpTo 8 8 4 4 4 1 1

$$\text{MinUpTo}[i] = \min_{i \in [j]} L[i]$$

(2) Compute all $\text{Best}[i] = L[i] - \text{MinUpTo}[i]$

Both steps $O(n)$ time. Why?

$$\text{MinUpTo}[i] = \min \left(\underbrace{\text{MinUpTo}[i-1]}_{\text{memoized}}, L[i] \right)$$

Faster algo, take 2:

All subproblems: (i, j) , $n + \binom{n}{2} = \Theta(n^2)$ of them.

Special subproblems : (i^*, j) where $i^* = \text{MinUpTo}[j]$

$$\text{Best}(i) = L[i] - L(i^*).$$

only $O(n)$ of them.

Are Special Subproblems harder?

Not with memoization!

$$\text{Best}(j) = \max \left(0, \text{Best}(j-1) + \underbrace{L(j) - L(j-1)}_{\text{difference in sell price}} \right)$$

buy on day j
 $i^* = j$

buy on day $< j$, keep i^* the same

No need for **MinUpTo**. $O(1)$ time per **Best(j)**
 $= O(n)$ total.

General strategy: 1) Design any correct algo.

2) Repeated structure? Memoize!

3) Go back to step 1). Simplify.

(the hardest step)

Two common forms

— Prefixes

— Multidimensional

Largest subsequence sum (Part IV, Section 2.2)

Input: L is a list of n elements in $\mathbb{R}$

Output: (i, j) with $1 \leq i \leq j \leq n$

maximizing $L[i] + L[i+1] + \dots + L[j-1] + L[j]$

Subsequence sum

Example

25	-60	7	-13	30
----	-----	---	-----	----

First try: start

row →	1	2	3	4	5
1	25	-35	-28	-41	-11
2		-60	-53	-66	-36
3			7	-6	24
4				-13	17
5					30

Naive: $\underbrace{O(n^2)}_{\text{\# subproblems}} \times \underbrace{O(n)}_{\text{time / subproblem}} = O(n^3)$

Better: $O(n^2) \times \underbrace{O(1)}_{\substack{\text{process row} \\ \text{by row, left-to-right}}} = O(n^2)$

$$\sum_{k=i}^j L(k) = \underbrace{\sum_{k=i}^{j-1} L(k)}_{\text{memoized}} + L(j)$$

Time to simplify.

Can we define $O(n)$ special subproblems?

$Best(j)$ = largest subsequence sum ending on j

Recursive formula:

$$Best(j) = \max \left(L(j), Best(j-1) + L(j) \right)$$

don't include $L(j-1)$ include $L(j-1)$.
may as well continue in best possible way

Again, $O(1)$ per subproblem using *memoization*

$\Rightarrow$ LSS in $O(n)$ time. ☺

(Kadane, at a CMU seminar)

Balanced parentheses (Part III, Section 2.3)

Input: L is sequence of n chars: (or)

Output: True or False, ^{↑ assume even}

Can we match parentheses s.t.
each (paired with a later)?

Examples

() (())) (

(((())))

(()) (())

Naive strategy: try all possible matchings

$$\binom{n}{2} \binom{n-2}{2} \dots \binom{4}{2} \binom{2}{2} = \frac{n!}{2^n} \gg \text{poly}(n)$$

How to solve in polynomial time? Multidimensional DP!

$S[i][j]$ = True or False,

Can we balance $L[i:j]$

Subarray between $L[i]$, $L[j]$

Recursive definition: Who is $L[i]$ paired to?

(balanced) OR $(\text{balanced}) (\text{balanced})$
 $i \qquad j \qquad i \qquad k \quad k+1 \qquad j$

$S[i][j]$ = True iff one of following holds

- $L[i] = ($ AND $L[j] =)$ AND $S[i+1][j-1]$
- $S[i][k]$ AND $S[k+1][j]$

possible pairings of i
↙

Recursion order: by length. $O(n^2) \times O(n) = O(n^3)$