

Course «Algorithms» · Lecture 5

Mathematical Analysis of Recursive Algorithms

Textbook: A. Levitin, Section 2.4

Today’s goals

- Turn a recursive algorithm into a **recurrence relation**
- Solve it by **backward substitutions**
- Read the cost from a **recursion tree**
- Use the **smoothness rule** for $n = 2^k$
- See why naive recursion can be **exponential**

Last week we counted loops with sums. Recursion has no loop to count, so we need a new tool.

The five-step plan

1. Choose the **input size** parameter n
2. Identify the **basic operation**
3. Check whether the count depends only on n (best / worst / average)
4. Write a **recurrence** with an **initial condition**
5. **Solve** it, or at least find its order of growth

Without the initial condition the recurrence has infinitely many solutions.

Example 1: factorial

```
F(n):
  if n = 0 return 1
  else return F(n – 1) * n
```

Basic operation: multiplication. M(n) = number of multiplications.

$$M(n) = M(n-1) + 1 \quad (n > 0), \qquad M(0) = 0$$

M(n – 1) to compute F(n – 1), plus one more to multiply by n.

Backward substitutions

Replace M(n – 1) by its own formula, again and again, until a pattern appears:

$$M(n) = M(n-1) + 1 = M(n-2) + 2 = M(n-3) + 3 = \ldots = M(n-i) + i$$

$$i = n: \qquad M(n) = M(0) + n = n \in \Theta(n)$$

Same count as the loop version. But recursion also uses a **stack of depth Θ(n)**.

Example 2: Tower of Hanoi

Move n disks from peg 1 to peg 3; never put a larger disk on a smaller one.

1. Move n – 1 disks from peg 1 to peg 2 (recursively)
2. Move the largest disk from peg 1 to peg 3
3. Move n – 1 disks from peg 2 to peg 3 (recursively)

$$M(n) = 2M(n-1) + 1 \quad (n > 1), \qquad M(1) = 1$$

Solving the Hanoi recurrence

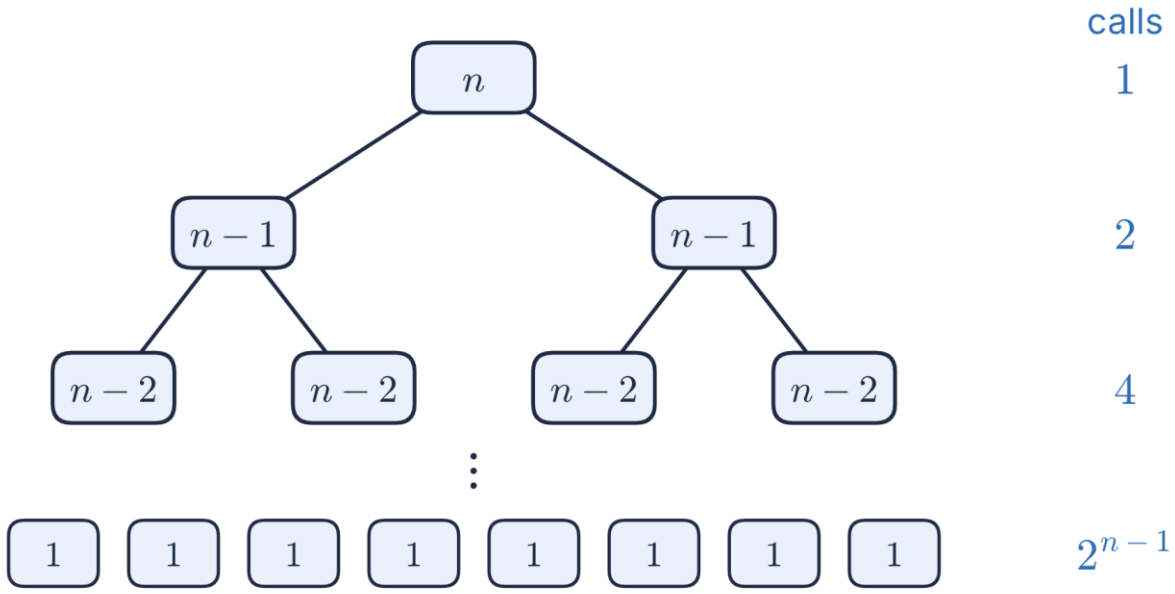
$$M(n) = 2M(n-1) + 1 = 2^2 M(n-2) + 2 + 1 = 2^3 M(n-3) + 2^2 + 2 + 1$$

$$= 2^i M(n-i) + 2^{i-1} + \ldots + 2 + 1 = 2^i M(n-i) + 2^i - 1$$

$$i = n-1: \qquad M(n) = 2^{n-1} + 2^{n-1} - 1 = 2^n - 1 \in \Theta(2^n)$$

10 disks: 1023 moves. 64 disks: about $1.8 \cdot 10^{19}$ moves.

The recursion tree



Each call makes one move.

Total: $1 + 2 + \ldots + 2^{n-1} = 2^n - 1$.

Not a bad algorithm: any solution needs $2^n - 1$ moves. The exponential cost belongs to the problem itself.

Example 3: binary digits

BinRec(n):

if n = 1 return 1

else return BinRec(⌊n/2⌋) + 1

Counts the binary digits of n. Basic operation: the addition. A(n) = number of additions.

$$A(n) = A(\lfloor n/2 \rfloor) + 1 \quad (n > 1), \qquad A(1) = 0$$

Trick: take n = 2^k

The floor makes substitution messy. For powers of two it disappears:

$$A(2^k) = A(2^{k-1}) + 1 = A(2^{k-2}) + 2 = \ldots = A(2^{k-i}) + i$$

$$A(2^k) = k \quad \Rightarrow \quad A(n) = \lfloor \log_2 n \rfloor \in \Theta(\log n)$$

Smoothness rule: log n is smooth and A(n) is non-decreasing, so Θ(log n) for powers of two gives Θ(log n) for all n.

Example 4: naive Fibonacci

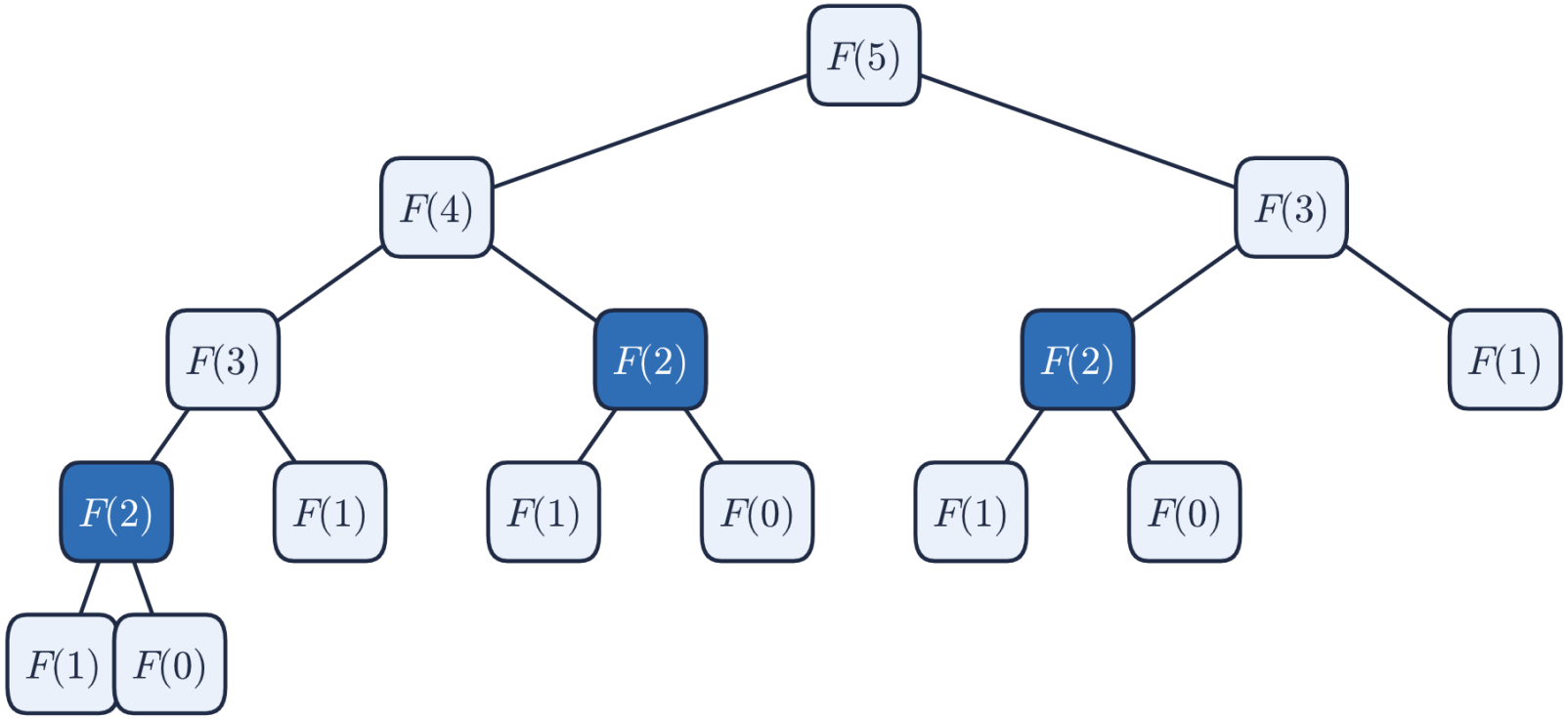
F(n):
if n ≤ 1 return n
else return F(n − 1) + F(n − 2)

$$A(n) = A(n-1) + A(n-2) + 1, \qquad A(0) = A(1) = 0$$

$$A(n) \in \Theta(\varphi^n), \qquad \varphi = \frac{1+\sqrt{5}}{2} \approx 1.618$$

F(50) this way needs more than 10^{10} additions, for a number a loop finds in 49.

Why so slow? The same work, again and again



F(2) is computed 3 times, F(3) twice. The tree grows like φ^n.

Recursion vs. iteration

Algorithm	Recursive	Better way
Factorial	$\Theta(n)$ time, $\Theta(n)$ stack	loop: $\Theta(n)$, $\Theta(1)$ space
Hanoi	$\Theta(2^n)$	no better: $2^n - 1$ is a lower bound
Binary digits	$\Theta(\log n)$	same order
Fibonacci	$\Theta(\varphi^n)$	loop or memoization: $\Theta(n)$

Recursion is fine when subproblems do not overlap. When they do, store the answers — this leads to dynamic programming.

Summary

- Recursive algorithm → recurrence + initial condition
- Backward substitutions: find the pattern, then hit the base case
- Factorial: n; Hanoi: $2^n - 1$; BinRec: $\lfloor \log_2 n \rfloor$
- For $\lfloor n/2 \rfloor$ take $n = 2^k$, then apply the smoothness rule
- Overlapping subproblems (Fibonacci) make recursion exponential
- Stack depth counts as memory

Review questions

1. List the five steps of the plan. Why is the initial condition necessary?
2. Write the Hanoi recurrence and solve it. Why is $2^n - 1$ a lower bound for the problem?
3. Solve $A(n) = A(\lfloor n/2 \rfloor) + 1$, $A(1) = 0$ for $n = 2^k$. How does the result extend to all n?
4. Why is naive recursive Fibonacci exponential? Name two $\Theta(n)$ alternatives.

Reading: Levitin, Sections 2.4–2.5 and Appendix B; CLRS, Chapter 4.