Recurrence

\[ T(n) \leq 2T(n/2) + cn \]
\[ T(1) \leq c \]

▶ Goal: "solve" recurrence \( \implies \) find simple expression for \( T(n) \)

▶ Three approaches
1. Unrolling ✓
2. Recursion tree (another version of unrolling)
3. Guess and verify (proof by induction)

Recurrence Solving (2): Recursion Tree

\[ T(n) + cn \]
\[ T(n/2) + cn/2 \]
\[ T(n/4) + cn/4 \]

Each level adds work to expand nodes

Conclusion: \( T(n) \leq cn \log n \)

Recurrence Solving (3): Guess and Verify

\[ T(n) \leq 2T(n/2) + cn \]
\[ T(2) \leq c \]

▶ Guess solution. \( T(n) \leq cn \log n \)

▶ Prove by (strong) induction

▶ Base case
\[ T(2) \leq c < c \cdot 2 = c \cdot 2 \log 2 \quad \checkmark \]

▶ Why \( T(2) \) instead of \( T(1) \)?

Clicker Question

\[ T(n) + cn \]
\[ T(n/2) + cn/2 \]
\[ T(n/4) + cn/4 \]

What is the work done at each level of recursive calls?
A. \( cn \)
B. \( 2^k T(n/2^k) \)
C. \( 2^k T(n/2^k) + cn \)
D. \( cn \), except \( cn \log n \) for all the leaves.

Induction step

Strong induction:
Assume \( T(m) \leq c \cdot m \log m \) for all \( m < n \), prove for \( n \)

\[ T(n) \leq 2T(n/2) + cn \]
\[ \leq 2c(n/2) \log(n/2) + cn \quad \text{by ind. hyp.} \quad m = n/2 \]
\[ = cn(\log n - 1) + cn \]
\[ = cn \log n \]

{slides credit: Marius Minea}
Wow, that worked out well...

Constants were chosen carefully $\Rightarrow$ everything simplifies nicely. If you don’t know correct constants, leave them as variables and set them at the end (important for more complicated cases).

Recurrence:

$T(n) \leq 2T(n/2) + cn$

Guess:

$T(n) \leq kn \log n$

Selecting Constants

Base case: $T(2) \leq a$ $\Rightarrow k \cdot 2 \log 2 = 2k$. Need $k \geq a/2$.

Induction step:

$T(n) \leq 2 \cdot T(n/2) + cn$

$= 2 \cdot k (n/2) \log(n/2) + cn$

$= kn \log n + (c - k)n$

$\leq kn \log n$

Need $k \geq c$. Induction proof works for $k = \max(c, a/2)$ (constant!)

A More General Recurrence

$T(n) \leq q \cdot T(n/2) + cn$

▶ What does the algorithm look like?

▶ $q$ recursive calls to itself on problems of half the size

▶ $O(n)$ work outside of the recursive calls

▶ Exercises: $q = 1$, $q > 2$ (recursion trees)

Useful Fact: Geometric Sum

If $r \neq 1$ then

$1 + r + r^2 + \ldots + r^d = \frac{r^{d+1} - 1}{r - 1}$

General Case

Work at level $j$ of recursion tree:

$q^j \times \frac{cn/2^j}{j} = \left(\frac{q}{2}\right)^j cn$

Total work:

$T(n) = cn \cdot \sum_{j=0}^{d} \left(\frac{q}{2}\right)^j \quad (d = \log_2 n)$

$q = 1$? Easy to see $T(n) \leq 2cn = O(n)$.

General Case ($q > 2$)

$T(n) = cn \cdot \sum_{j=0}^{d} \left(\frac{q}{2}\right)^j \quad (d = \log_2 n)$

Let $r = q/2 > 1$. Then

$\sum_{j=0}^{d} r^j = \frac{r^{d+1} - 1}{r - 1}$

Therefore,

$T(n) = cn \cdot O(n^{\log_2 q - 1})$

E.g., $q = 3$, $T(n)$ is $O(n^{1.59})$
Summary

Useful general recurrence and its solutions:

\[ T(n) \leq q \cdot T(n/2) + cn \]

1. \( q = 1 \): \( T(n) = O(n) \) dominated by root
2. \( q = 2 \): \( T(n) = O(n \log n) \) same work every level
3. \( q > 2 \): \( T(n) = O(n \log_2 q) \) dominated by leaves

Work at is either exponentially decreasing, staying same, or exponentially increasing with level

Algorithms with these running times?
1. ???
2. MSS, Mergesort
3. Integer multiplication...

Clicker Question

Which of the following is not true?
A. \( n \log n = O(n^2) \)
B. \( n \log n = O(n^{1.1}) \)
C. There exists some \( k \) such that \( n \log n = \Theta(n^k) \).
D. \( n \log n = \Omega(n \log \log n) \)

Integer Multiplication

Motivation: multiply two 30-digit integers?

\[
\begin{array}{c}
153819617987625488624070712657 \\
x 925421863832406144537293648227 \\
--------------------
\end{array}
\]

\[
\begin{array}{c}
574 \\
861 \\
287 \\
--------
\end{array}
\]

\[
\begin{array}{c}
37884 \\
\end{array}
\]

Running time? \( \Theta(n) \)

Warm-Up: Addition

Input: two \( n \)-digit binary integers \( x \) and \( y \)
Goal: compute \( x + y \)

Let’s do everything in base-10 instead of binary to make examples more familiar.

Grade-school algorithm:

\[
\begin{array}{c}
1854 \\
+ 3242 \\
-------
\end{array}
\]

\[
\begin{array}{c}
5096 \\
\end{array}
\]

Running time? \( \Theta(n) \)

Integer Multiplication Problem

Input: two \( n \)-digit base-10 integers \( x \) and \( y \)
Goal: compute \( xy \)

Can anyone think of an algorithm?

Grade-School Algorithm (Long Multiplication)

Example: \( n = 3 \)

\[
\begin{array}{c}
287 \\
x 132 \\
------
\end{array}
\]

\[
\begin{array}{c}
574 \\
861 \\
287 \\
--------
\end{array}
\]

\[
\begin{array}{c}
37884 \\
\end{array}
\]

\[
287 \times 132 = (2 \times 287) + 10 \cdot (3 \times 287) + 100 \cdot (1 \times 287)
\]

Running time? \( \Theta(n^2) \)
But \( xy \) has at most \( 2n \) digits. Can we do better?
**Better Divide and Conquer**

**Divide and Conquer – First Try: An Example**

Idea: split \( x \) and \( y \) in half (assume \( n \) is a power of 2)

\[
\begin{align*}
x &= 3280 \quad 2367 \\
y &= 4508 \quad 1854
\end{align*}
\]

Then use distributive law

\[
x y = (10^n/2 x_1 + x_0) \times (10^n/2 y_1 + y_0)
= 10^n x_1 y_1 + 10^n/2 (x_1 y_0 + x_0 y_1) + x_0 y_0
\]

Have reduced the problem to multiplications of \( n/2 \)-digit integers and additions of \( n \)-digit numbers.

(Ignore time to multiply by \( 10^k \). Why?)

**Better Divide and Conquer**

Same starting point:

\[
x y = 10^n x_1 y_1 + 10^n/2 (x_1 y_0 + x_0 y_1) + x_0 y_0
\]

**Trick:** use three multiplications to compute the following:

\[
\begin{align*}
A &= (x_1 + x_0)(y_1 + y_0) = x_1 y_1 + x_1 y_0 + x_0 y_1 + x_0 y_0 \\
B &= x_1 y_1 \\
C &= x_0 y_0
\end{align*}
\]

Then

\[
x y = 10^n B + 10^n/2 (A - B - C) + C
\]

**Total:** three multiplications of \( n/2 \)-digit integers, six additions of at most \( n \)-digit numbers

We beat long multiplication!
Can be done even faster (split \( x \) and \( y \) into \( k \) parts instead of two)

**Divide and Conquer – First Try: Analysis**

Recursive algorithm:

\[
x y = 10^n x_1 y_1 + 10^n/2 (x_1 y_0 + x_0 y_1) + x_0 y_0
\]

**Running time?**

Four multiplications of \( n/2 \) digit numbers plus three additions of at most \( n \)-digit numbers

\[
T(n) \leq 4T(n/2) + cn
= O(n \log 4)
= O(n)
\]

We did not beat the grade-school algorithm. :(

**Master Theorem**

Consider the general recurrence:

\[
T(n) \leq a T(n/b) + cn^d
\]

(What algorithm?) This solves to:

\[
T(n) = \begin{cases} 
\Theta(n^d) & \text{if } \log_b a < d \\
\Theta(n^{d \log_b a}) & \text{if } \log_b a = d \\
\Theta(n^{ \log_b a}) & \text{if } \log_b a > d
\end{cases}
\]

Intuition: work at each level of the recursion tree is (1) decreasing exponentially, (2) staying the same, (3) increasing exponentially.

**Clicker Question**

Master Theorem:

\[
T(n) \leq a T(n/b) + cn^d
\]

\[
T(n) = \begin{cases} 
\Theta(n^d) & \text{if } \log_b a < d \\
\Theta(n^{d \log_b a}) & \text{if } \log_b a = d \\
\Theta(n^{ \log_b a}) & \text{if } \log_b a > d
\end{cases}
\]

Suppose \( T(n) = 9T(n/3) + n^d \). What is the largest value for \( d \) below such that \( T(n) = \Theta(n^2) \)?

A. \( d = 1 \)
B. \( d = 1.5 \)
C. \( d = 2 \)
D. \( d = 3 \)