CSE 548: Analysis of Algorithms

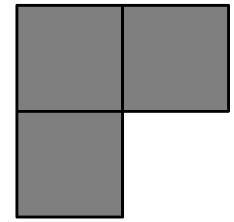
Lecture 2
(Divide-and-Conquer Algorithms: Integer Multiplication)

Rezaul A. Chowdhury

Department of Computer Science
SUNY Stony Brook
Fall 2017

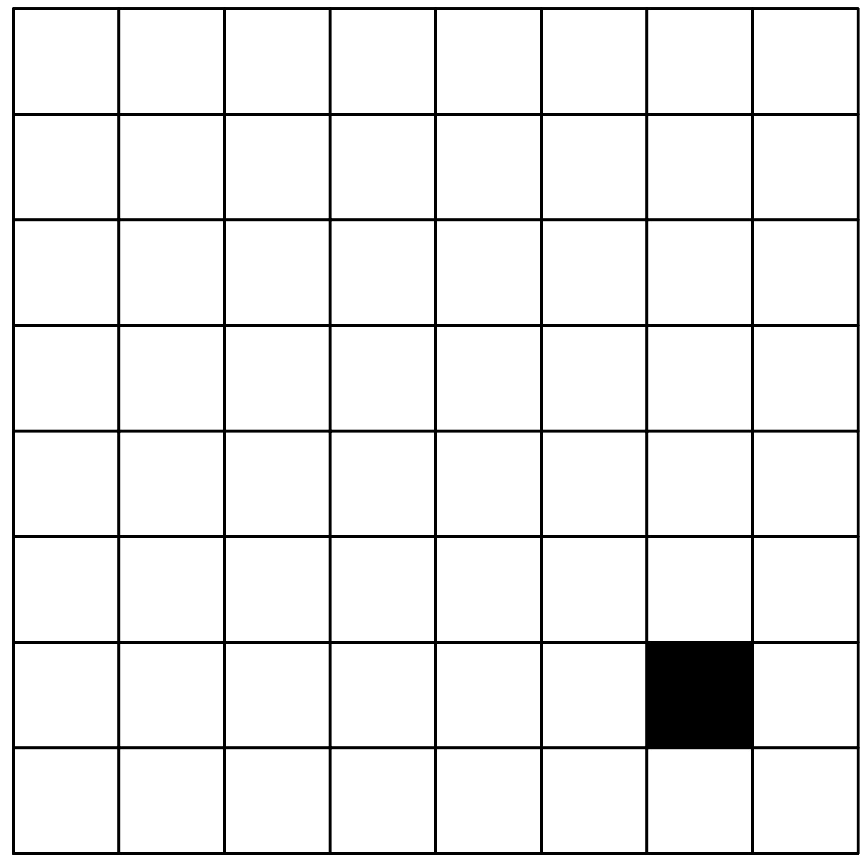
Tromino Cover

A *right tromino* is an L-shaped tile formed by three adjacent squares.



Puzzle: You are given a $2^n \times 2^n$ board with one missing square.

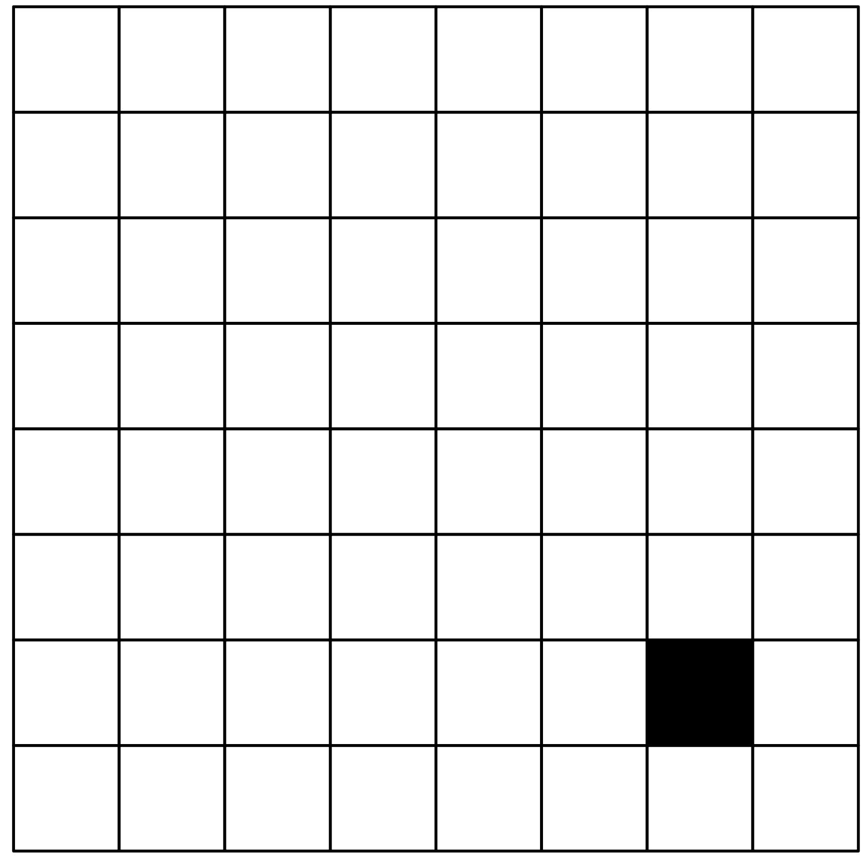
- you must cover all squares except the missing one exactly using right trominoes
- the trominoes must not overlap



$2^3 \times 2^3$ board

Tromino Cover

Steps

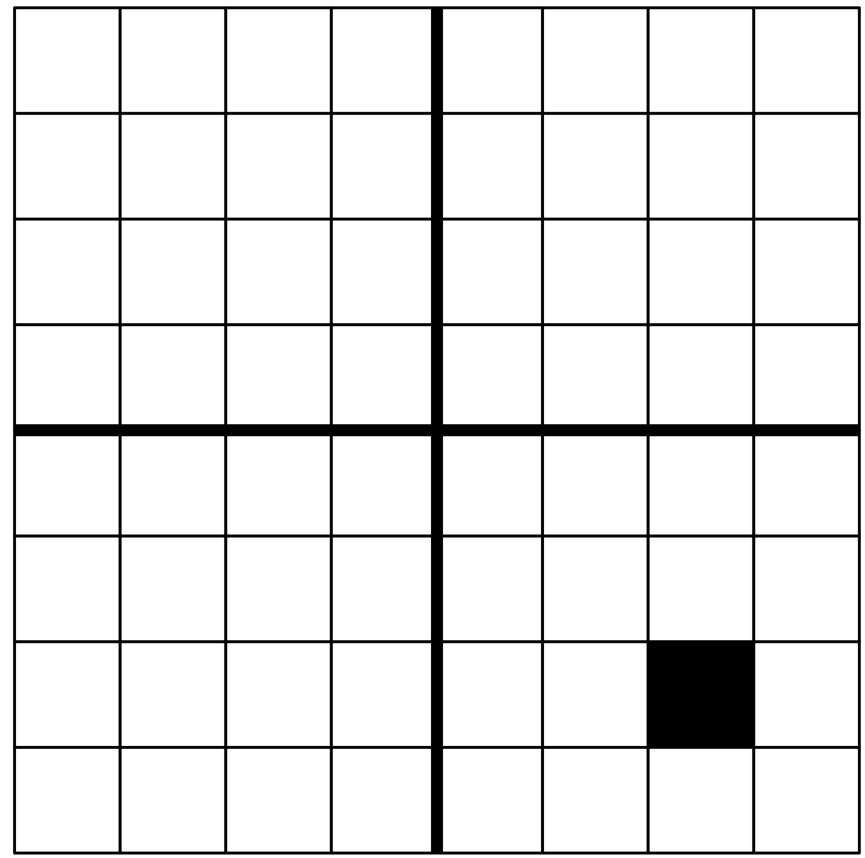


$2^3 \times 2^3$ board

Tromino Cover

Steps

– Divide the $2^n \times 2^n$ board into 4 disjoint $2^{n-1} \times 2^{n-1}$ subboards.

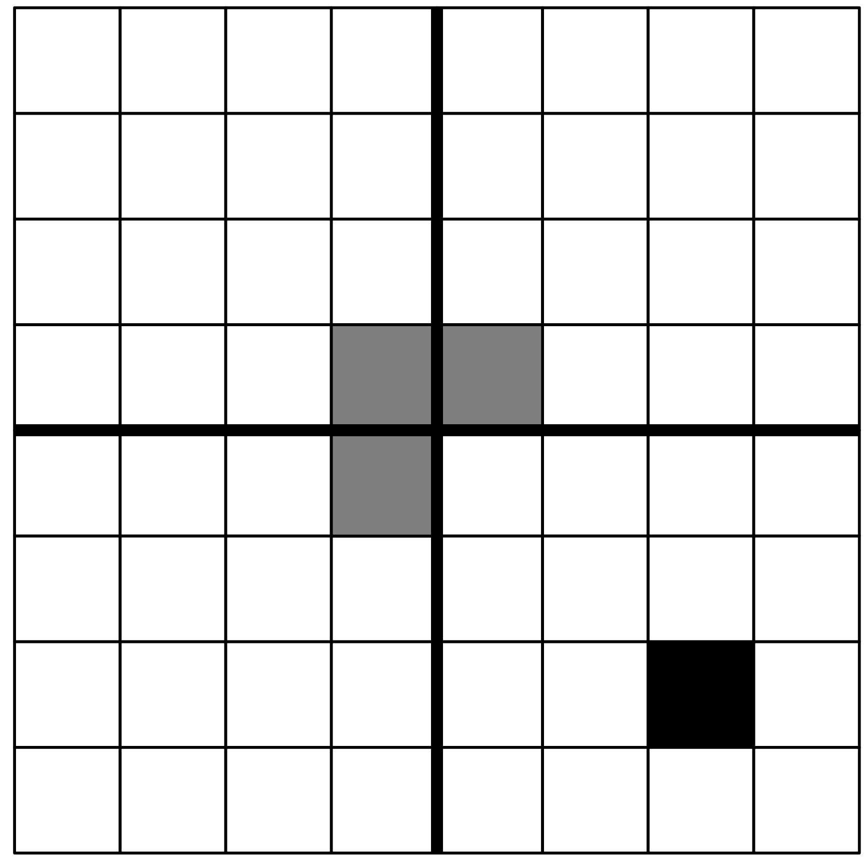


$2^3 \times 2^3$ board

Tromino Cover

Steps

- Divide the $2^n \times 2^n$ board into 4 disjoint $2^{n-1} \times 2^{n-1}$ subboards.
- Place a tromino at the center so that it fully covers one square from each of the three (3) subboards with no missing square, and misses the fourth subboard completely.



$2^3 \times 2^3$ board

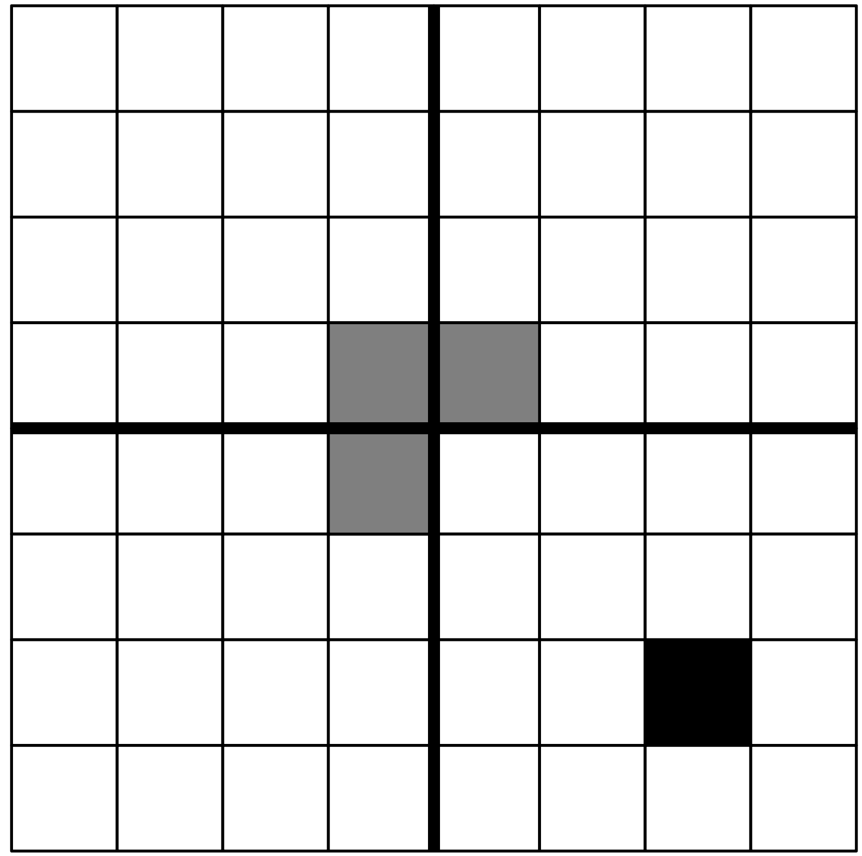
Tromino Cover

Steps

– Divide the $2^n \times 2^n$ board into 4 disjoint $2^{n-1} \times 2^{n-1}$ subboards.

– Place a tromino at the center so that it fully covers one square from each of the three (3) subboards with no missing square, and misses the fourth subboard completely.

This reduces the original problem into 4 smaller instances of the same problem!



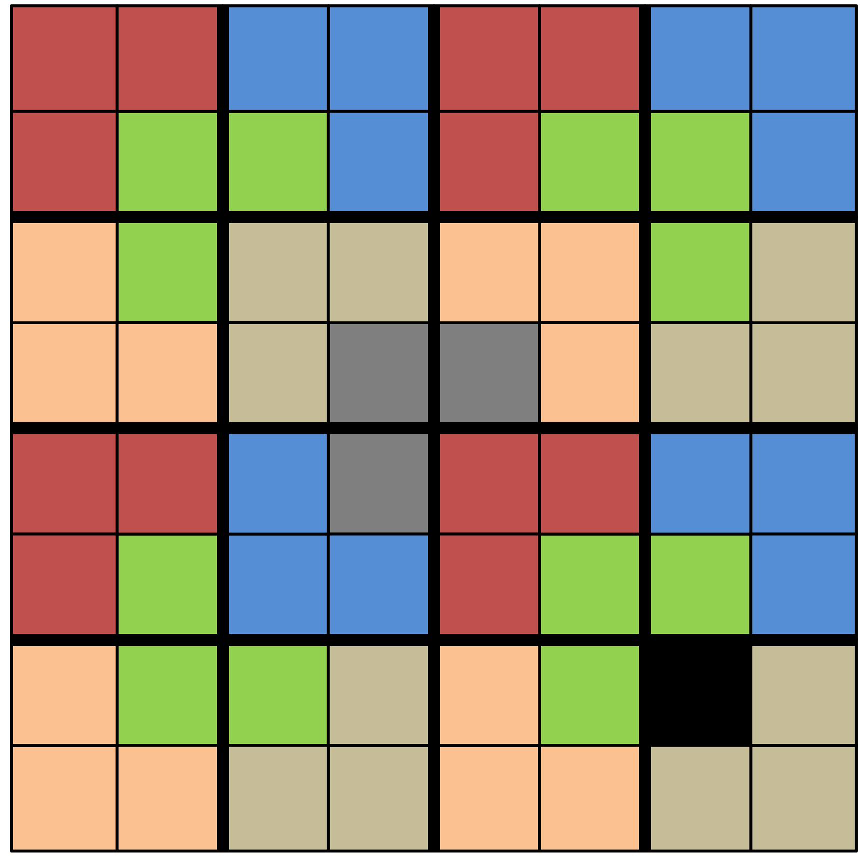
$2^3 \times 2^3$ board

Tromino Cover

Steps

- Divide the $2^n \times 2^n$ board into 4 disjoint $2^{n-1} \times 2^{n-1}$ subboards.
- Place a tromino at the center so that it fully covers one square from each of the three (3) subboards with no missing square, and misses the fourth subboard completely.

  This reduces the original problem into 4 smaller instances of the same problem!
- Solve each smaller subproblem recursively using the same technique.



$2^3 \times 2^3$ board

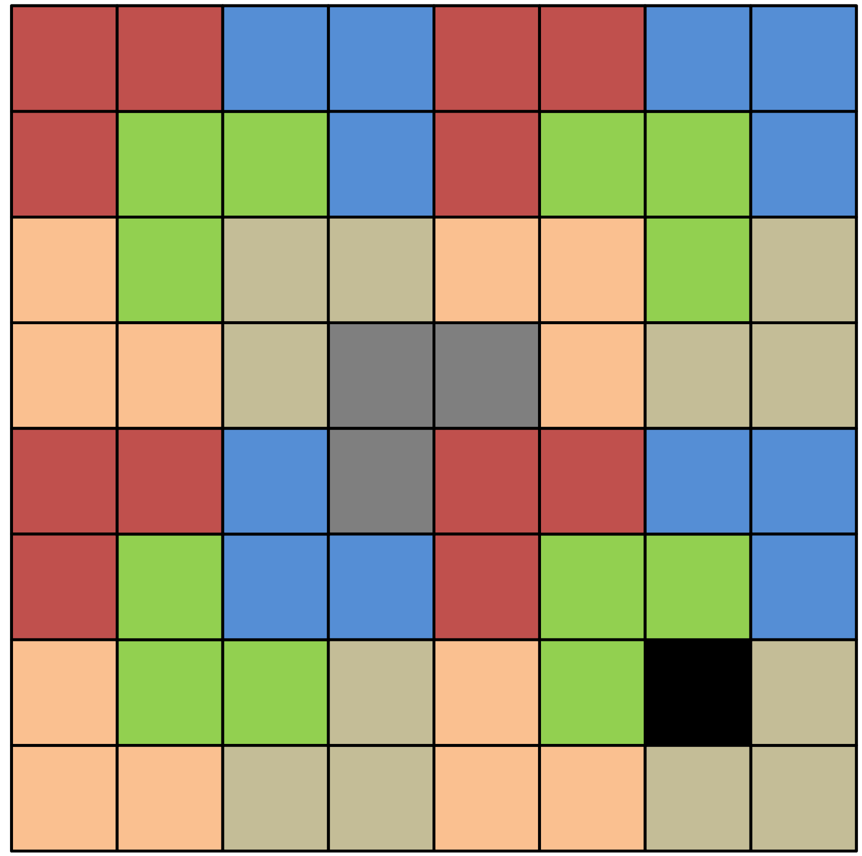
Tromino Cover

Steps

– Divide the $2^n \times 2^n$ board into 4 disjoint $2^{n-1} \times 2^{n-1}$ subboards.

– Place a tromino at the center so that it fully covers one square from each of the three (3) subboards with no missing square, and misses the fourth subboard completely.

This reduces the original problem into 4 smaller instances of the same problem!



$2^3 \times 2^3$ board

– Solve each smaller subproblem recursively using the same technique.

– This algorithm design technique is called *recursive divide & conquer.*

A Latin Phrase

"Divide et impera"

(meaning: "divide and rule" or "divide and conquer")

— Philip II, king of Macedon (382-336 BC),
describing his policy toward the Greek city-states
(some say the Roman emperor Julius Caesar,
100-44 BC, is the source of this phrase)

The strategy is to break large power alliances into smaller ones that are easier to manage (or subdue).

This is a combination of political, military and economic strategy of gaining and maintaining power.

Unsurprisingly, this is also a very powerful problem solving strategy in computer science.

Divide-and-Conquer

1. **Divide:** divide the original problem into smaller subproblems that are easier are to solve
2. **Conquer:** solve the smaller subproblems (perhaps recursively)
3. **Merge:** combine the solutions to the smaller subproblems to obtain a solution for the original problem

Integer Multiplication

Multiplying Two n-bit Numbers

$$x = \underbrace{\overbrace{\boxed{\;x_L\;}}^{\frac{n}{2}\,bits}\overbrace{\boxed{\;x_R\;}}^{\frac{n}{2}\,bits}}_{n\ bits} = 2^{n/2}x_L + x_R$$

$$y = \boxed{\;y_L\;}\boxed{\;y_R\;} = 2^{n/2}y_L + y_R$$

$$xy = \left(2^{n/2}x_L + x_R\right)\left(2^{n/2}y_L + y_R\right) = 2^n x_L y_L + 2^{n/2}(x_L y_R + x_R y_L) + x_R y_R$$

So # $\frac{n}{2}$-bit products: 4

bit shifts (by n or $\frac{n}{2}$ bits): 2

additions (at most $2n$ bits long) : 3

We can compute the $\frac{n}{2}$-bit products recursively.

Let $T(n)$ be the overall running time for n-bit inputs. Then

$$T(n) = \begin{cases} \Theta(1) & if\ n = 1, \\ 4T\left(\frac{n}{2}\right) + \Theta(n) & otherwise. \end{cases} = \Theta\left(n^2\right) \quad \text{(how? derive)}$$

Multiplying Two n-bit Numbers Faster (Karatsuba's Algorithm)

$$\begin{array}{rcc l}
 & \overbrace{\hphantom{x_L x_L x_L}}^{\frac{n}{2}\,bits} & \overbrace{\hphantom{x_R x_R x_R}}^{\frac{n}{2}\,bits} & \\
x = & \boxed{\quad x_L \quad} & \boxed{\quad x_R \quad} & = 2^{n/2}x_L + x_R \\
y = & \boxed{\quad y_L \quad} & \boxed{\quad y_R \quad} & = 2^{n/2}y_L + y_R \\
 & \underbrace{\hphantom{y_L y_L y_L y_L y_L y_L y_L}}_{n\ bits} & &
\end{array}$$

$$xy = \left(2^{n/2}x_L + x_R\right)\left(2^{n/2}y_L + y_R\right)$$

$$= 2^n x_L y_L + 2^{n/2}(x_L y_R + x_R y_L) + x_R y_R$$

$$= 2^n x_L y_L + 2^{n/2}\left((x_L + x_R)(y_L + y_R) - x_L y_L - x_R y_R\right) + x_R y_R$$

So # $\frac{n}{2}$- or $\left(\frac{n}{2} + 1\right)$ -bit products: 3

Then the overall running time for n-bit inputs:

$$T(n) = \begin{cases} \Theta(1) & if\ n = 1, \\ 3T\left(\frac{n}{2}\right) + \Theta(n) & otherwise. \end{cases}$$

$$= \Theta\left(n^{\log_2 3}\right) = O\left(n^{1.59}\right)$$ (how? derive)

Algorithms for Multiplying Two *n*-bit Numbers

Inventor	Year	Complexity
Classical	—	$\Theta(n^2)$
Anatolii Karatsuba	1960	$\Theta(n^{\log_2 3})$
Andrei Toom & Stephen Cook (generalization of Karatsuba's algorithm)	1963 – 66	$\Theta\left(n2^{\sqrt{2\log_2 n}}\log n\right)$
Arnold Schönhage & Volker Strassen (Fast Fourier Transform)	1971	$\Theta(n \log n \log\log n)$
Martin Fürer (Fast Fourier Transform)	2005	$n \log n \, 2^{O(\log^* n)}$

Lower bound: $\Omega(n)$ (why?)