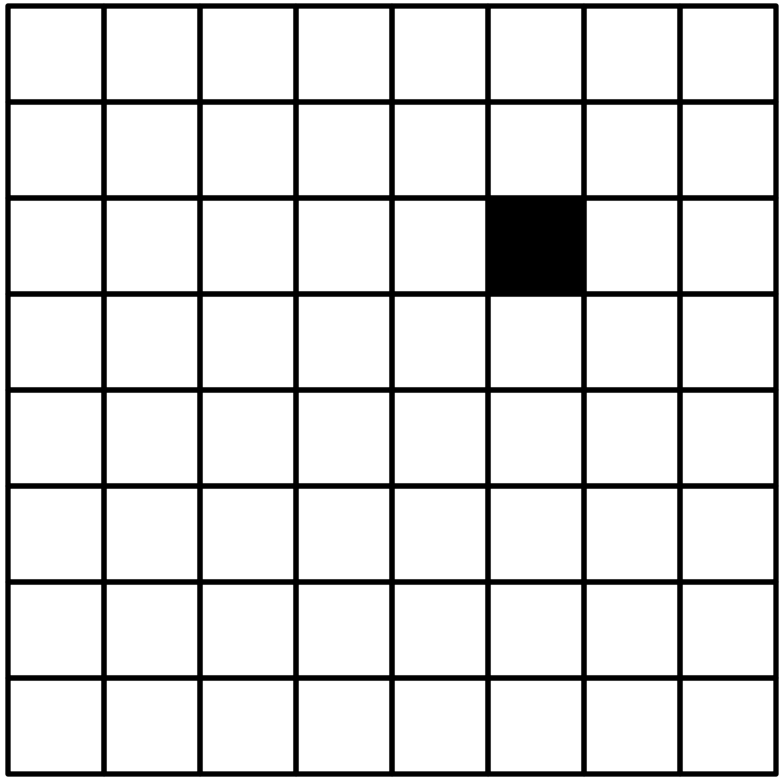
CS4102 Algorithms

Spring 2022

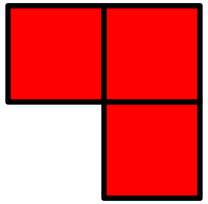
Warm up

Can you cover an 8×8 grid with 1 square missing using “trominoes?”

Can you cover this?



With these?



https://nstarr.people.amherst.edu/trom/puzzle-8by8/

Announcements

Cheer Loudest For Pizza

At UVA games, there's a contest where two adjacent sections cheer. The loudest section gets pizza! Let's give each of the n sections a chance to win

1. Divide the sections into two halves
2. One half cheers, then the other half cheers
3. The louder half continues, the other half is eliminated
4. Repeat until just two sections left.

The louder of the final two wins free pizza!

What are we going to count?

Let's solve the recurrence!

Special case: $n = 2^k$

$T(2) = 2$

$T(n) = 2 + \cancel{T(n/2)}$

$2 + \cancel{T(n/4)}$

$2 + \cancel{T(n/8)}$

$\vdots$

2

$\left.\right\} k - 1$

$$T(n) = 2 + \sum_{i=1}^{\log_2 n - 1} 2 = 2\,(\log_2 n - 1) + 2 = 2\log_2 n$$

What if $n \neq 2^k$?

- More sections to compete → more time

  – $\forall\ 0 < n < m, T(n) < T(m)$, where $2^{\mathrm{k}} < n < 2^{k+1} = m$,
  i.e., $\mathrm{k} < \log n < k + 1$

  – $T(n) \leq T(m) = T\left(2^{k+1}\right) = T\left(2^{\lceil \log_2 n \rceil}\right) = 2\lceil \log_2 n \rceil$

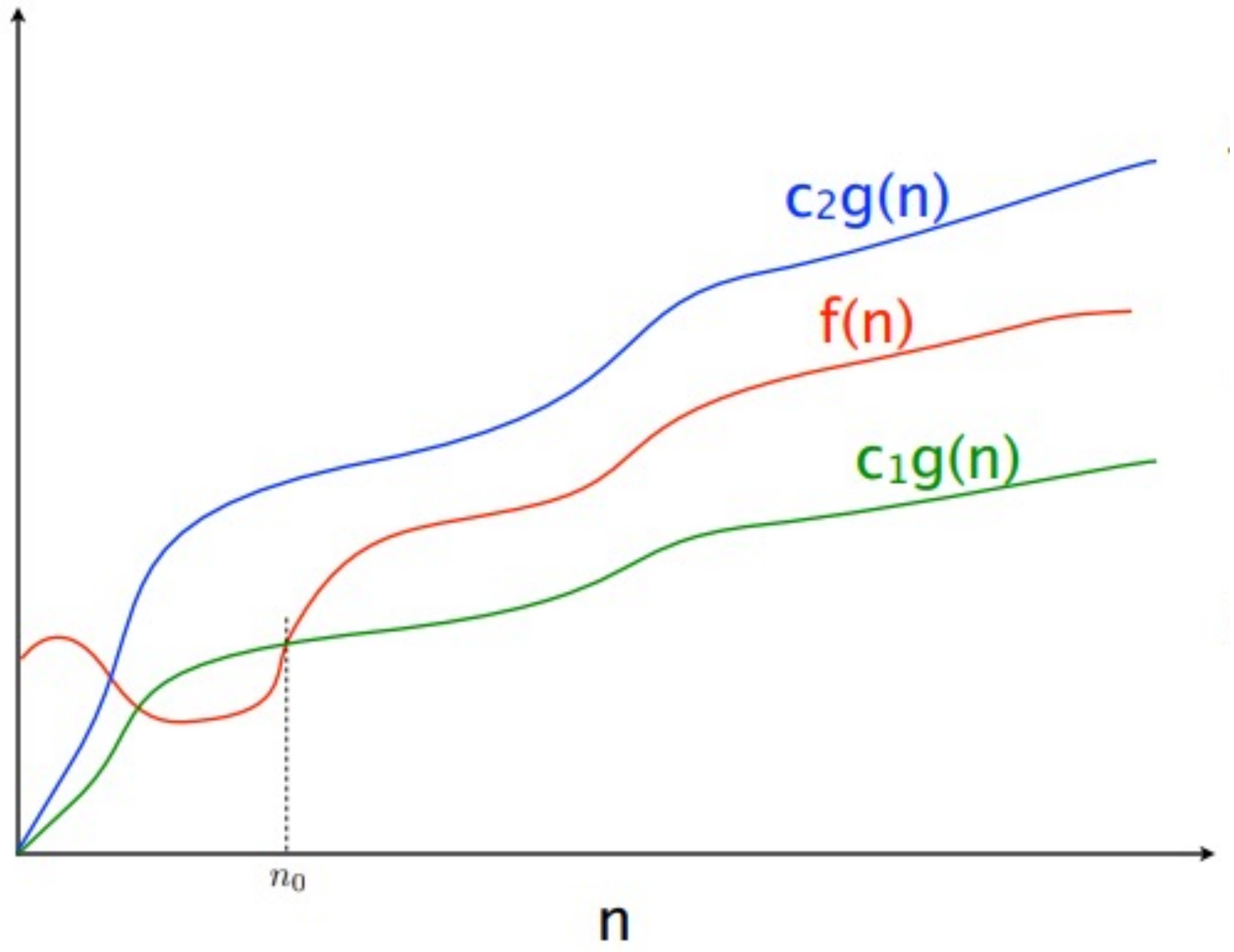
$$T(n) \leq 2\lceil \log_2 n \rceil = O(\log_2 n)$$

Why?

Asymptotic Notation*

- $\boldsymbol{O(g(n))}$
  - At most within constant of g for large n
  - $\{$functions $f \mid \exists$ constants $c, n_0 > 0$ s.t. $\forall n > n_0, f(n) \leq c \cdot g(n)\}$
  - Set of functions that grow "in the same way" as or more *slowly* than g(n)
- $\boldsymbol{\Omega(g(n))}$
  - At least within constant of g for large n
  - $\{$functions $f \mid \exists$ constants $c, n_0 > 0$ s.t. $\forall n > n_0, f(n) \geq c \cdot g(n)\}$
  - Set of functions that grow "in the same way" as or more *quickly* than g(n)
- $\boldsymbol{\Theta(g(n))}$
  - "Tightly" within constant of g for large n
  - $\Omega(g(n)) \cap O(g(n))$
  - Set of functions that grow "in the same way" as g(n)

*CLRS Chapter 3



$$f(n) = O(g(n))$$

$$f(n) = \Theta(g(n))$$

$$f(n) = \Omega(g(n))$$

Asymptotic Bounds

- The Sets "big oh" $O(g)$, "big theta" $\Theta(g)$, "big omega" $\Omega(g)$ – remember these meanings:
  - $O(g)$: functions that grow **no faster** than g, or **g is an asymptotic upper bound**
  - $\Omega(g)$: functions that grow **at least as fast** as g, or **g is an asymptotic lower bound**
  - $\Theta(g)$: functions that grow **at the same rate** as g, or **g is an asymptotic tight bound**

Asymptotic Notation Example

- Show: $n \log n \in O(n^2)$

Asymptotic Notation Example

Direct Proof!

- To Show: $n \log n \in O(n^2)$
  - **Technique:** Find $c, n_0 > 0$ s.t. $\forall n > n_0, n \log n \leq c \cdot n^2$
  - **Proof:** Let $c = 1, n_0 = 1$. Then,

$$n_0 \log n_0 = (1) \log (1) = 0,$$
$$c\, n_0^2 = 1 \cdot 1^2 = 1,$$
$$0 \leq 1.$$

$$\forall n \geq 1, \log(n) < n \Rightarrow n \log n \leq n^2 \quad \square$$

Asymptotic Notation Example

- Show: $n^2 \notin O(n)$

Asymptotic Notation Example

- To Show: $n^2 \notin O(n)$
  - **Technique: Contradiction**
  - **Proof:** Assume $n^2 \in O(n)$. Then $\exists c, n_0 > 0 \text{ s.t. } \forall n > n_0, n^2 \leq cn$
    Let us derive constant c. For all $n > n_0 > 0$, we know:
    $cn \geq n^2$,
    $c \geq n$.

    Since c is dependent on n, it is not a constant.
    Contradiction. Therefore $n^2 \notin O(n)$. □

Proof by Contradiction!

Proof Techniques

- Direct Proof ✓
  - From the assumptions and definitions, directly derive the statement
- Proof by Contradiction ✓
  - Assume the statement is true, then find a contradiction
- Proof by Cases
- Induction

More Asymptotic Notation

- $\boldsymbol{o(g(n))}$
  - Smaller than *any* constant factor of g for sufficiently large n
  - {functions f : $\forall$ constants $c > 0, \exists n_0$ such that $\forall n > n_0, f(n) < c \cdot g(n)$}
  - Set of functions that always grow more slowly than g(n)

Equivalently, ratio of $\frac{f(n)}{g(n)}$ is <u>decreasing</u> and tends towards 0:

$$f(n) \in o\big(g(n)\big) \Leftrightarrow \lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$$

More Asymptotic Notation

- $\boldsymbol{o(g(n))}$
  - Smaller than *any* constant factor of g for sufficiently large n
  - $\{$functions f : $\forall$ constants $c > 0$, $\exists n_0$ such that $\forall n > n_0, f(n) < c \cdot g(n)\}$
  - Set of functions that always grow more slowly than g(n)
- $\boldsymbol{\omega(g(n))}$
  - Greater than *any* constant factor of g for large n
  - $\{$functions f : $\forall$ constants $c > 0$, $\exists n_0$ such that $\forall n > n_0, f(n) > c \cdot g(n)\}$
  - Set of functions that always grow more quickly than g(n)

Equivalently, $f(n) \in \omega\big(g(n)\big) \Leftrightarrow \lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$

Another Asymptotic Notation Example

Direct Proof

- **Show:** $n \log n \in o(n^2)$
- **Proof Technique:** Show the statement directly, using either definition
- For every constant $c > 0$, we can find an n_0 such that $\frac{\log n_0}{n_0} = c$.
  Then for all $n > n_0$, $n \log n < c\, n^2$ since $\frac{\log n}{n}$ is a decreasing function
  $\forall$ constants $c > 0$, $\exists n_0$ such that $\forall n > n_0, f(n) < c \cdot g(n)$
- Equivalently, $\lim_{n\to\infty} \frac{n \log n}{n^2} = \lim_{n\to\infty} \frac{\log n}{n} = 0$ (why is this true?)

Summary: Using Limit Definition

Comparing f(n) and g(n) as n approaches infinity, calculate this:

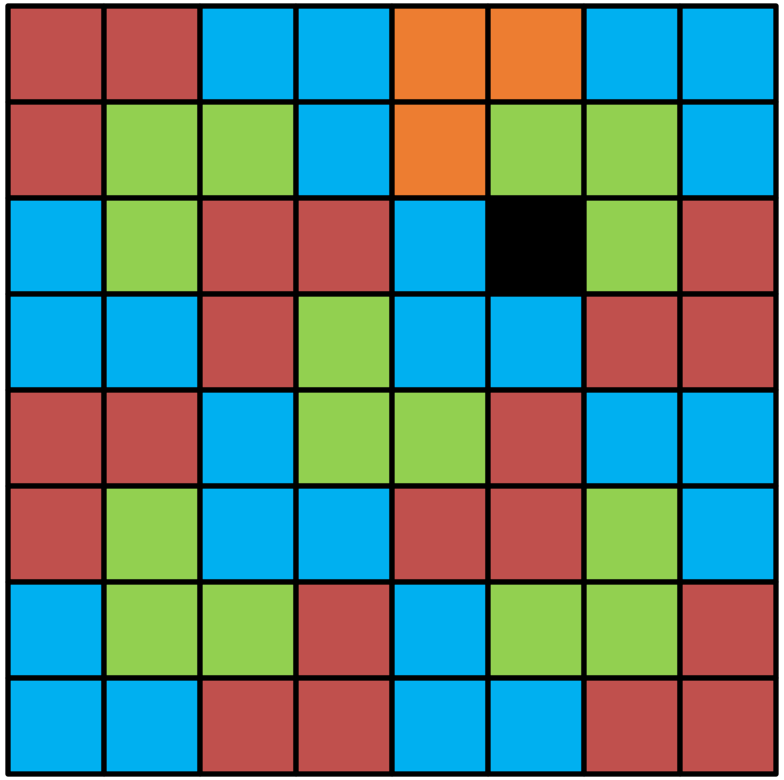
$$\lim_{n \to \infty} \frac{f(n)}{g(n)}$$

If the result….

- $< \infty$, including the case in which the limit is 0, then $f \in O(g)$
- > 0, including the case in which the limit is ∞, then $f \in \Omega(g)$
- $= c$ and $0 < c < \infty$ then $f \in \Theta(g)$
- $= 0$ then $f \in o(g)$
- $= \infty$ then $f \in \omega(g)$

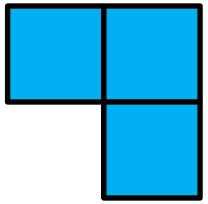
Back to Trominoes

A solution!



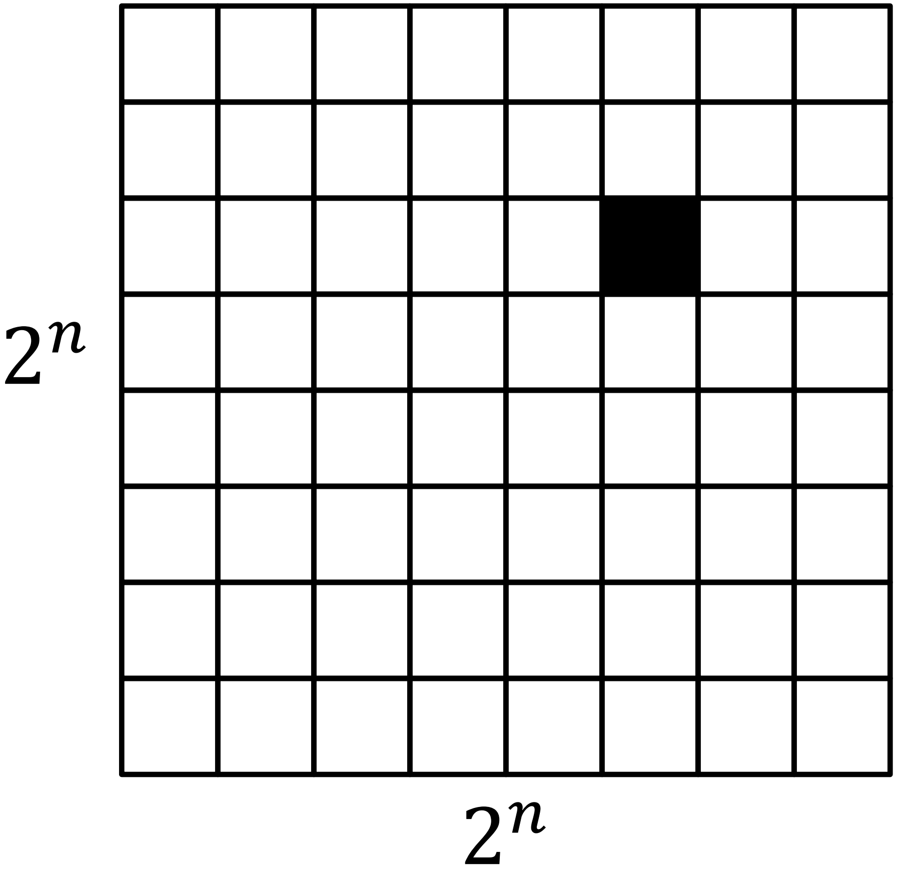
Can you cover an 8×8 grid with 1 square missing using “trominoes?”

What about a 4x4 grid? 2x2? ☺



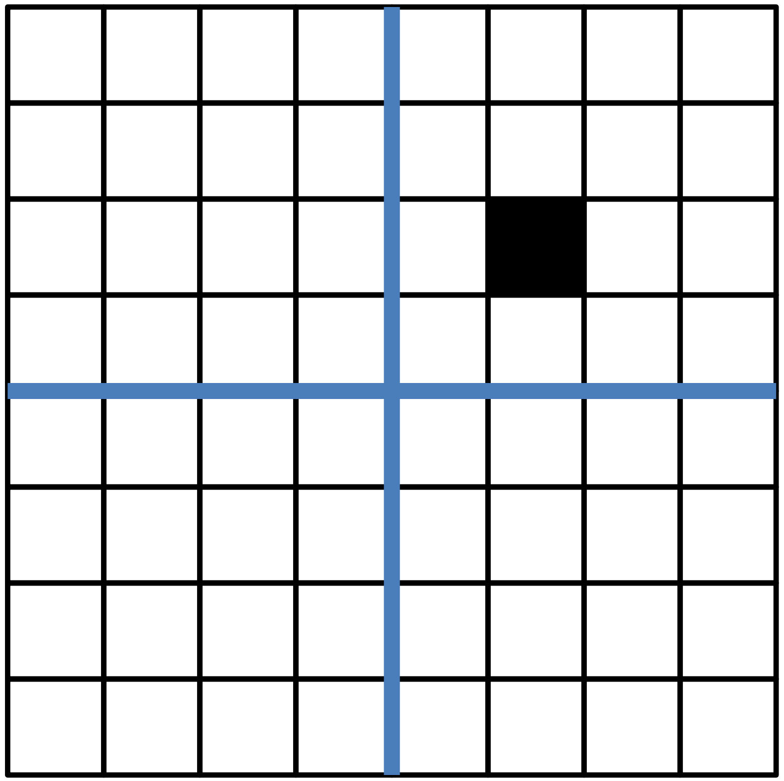
Tromino

Trominoes Puzzle Solution



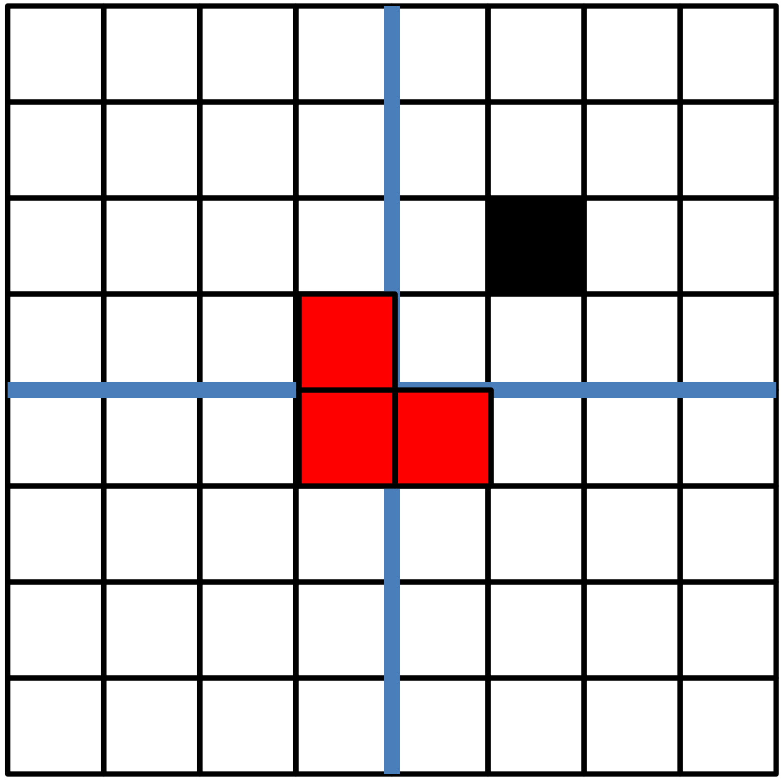
What about larger boards?

Trominoes Puzzle Solution



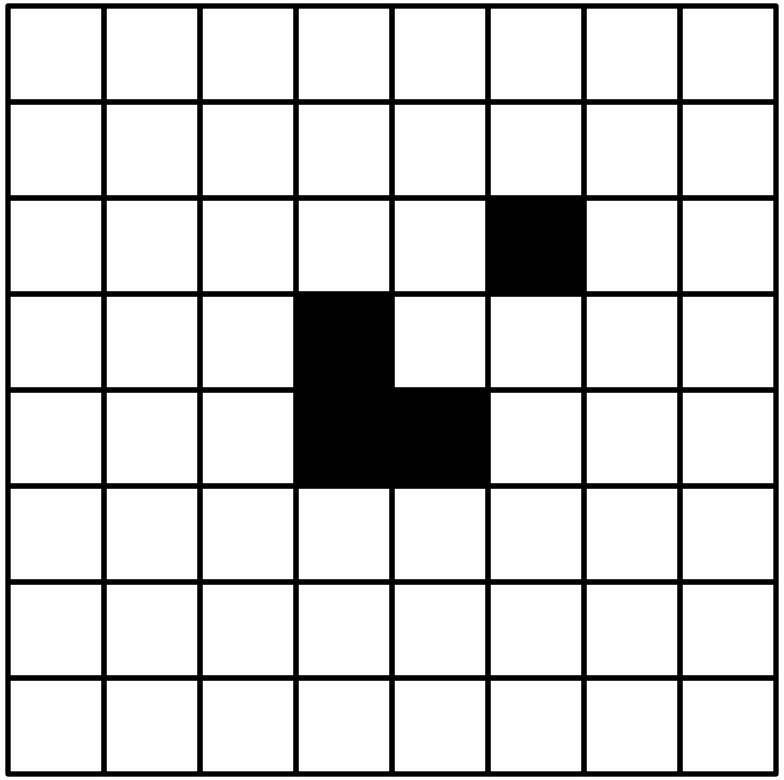
Divide the board into quadrants

Trominoes Puzzle Solution



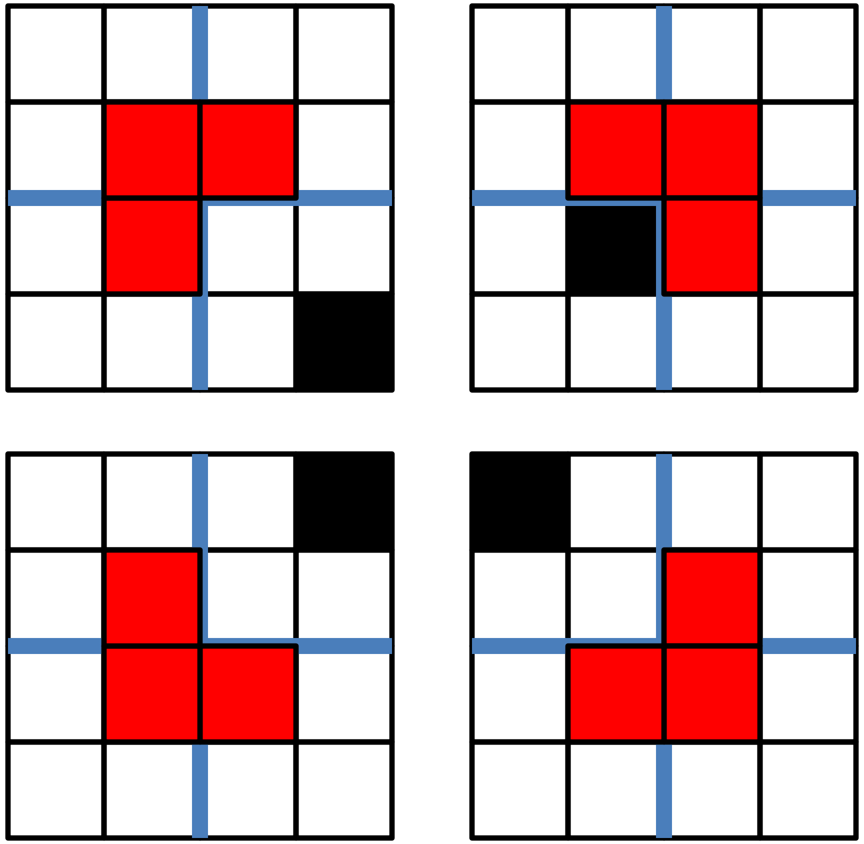
Place a tromino to occupy the three quadrants without the missing piece

Trominoes Puzzle Solution



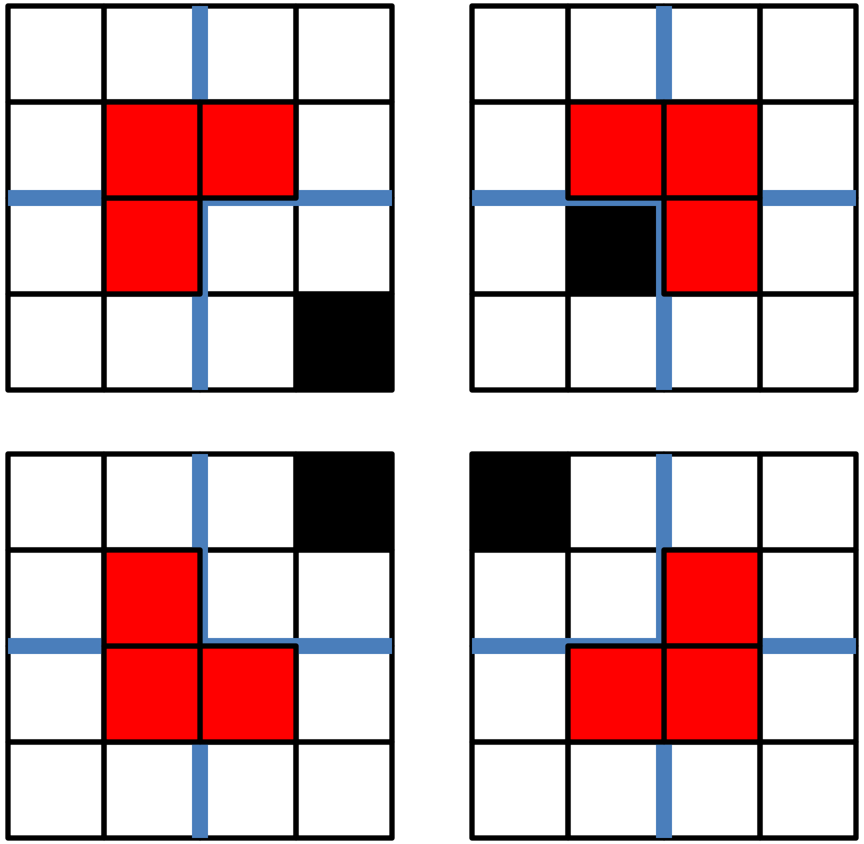
Each quadrant is now a smaller subproblem

Trominoes Puzzle Solution



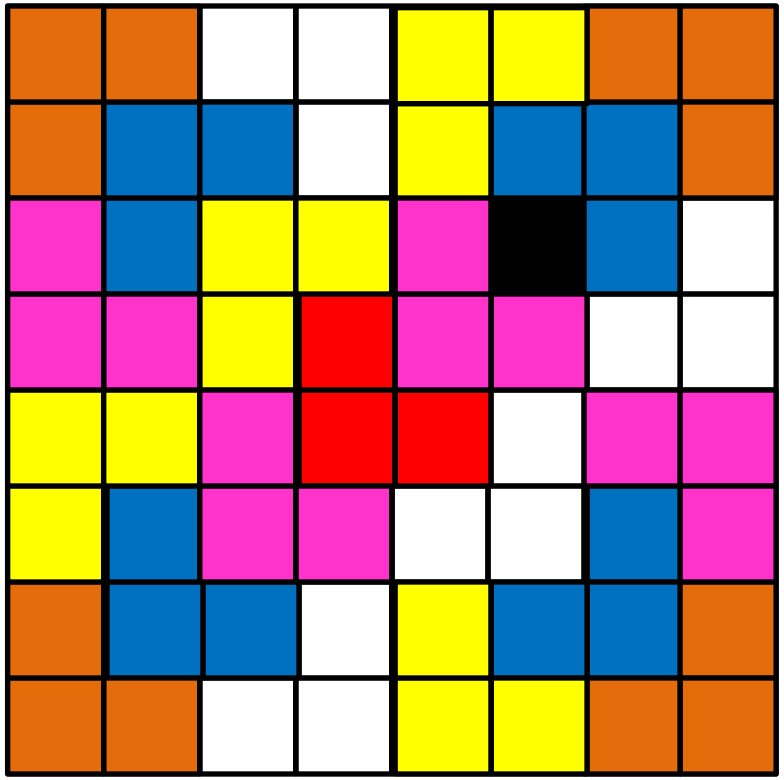
Solve **Recursively**

Divide and Conquer



Our first algorithmic technique!

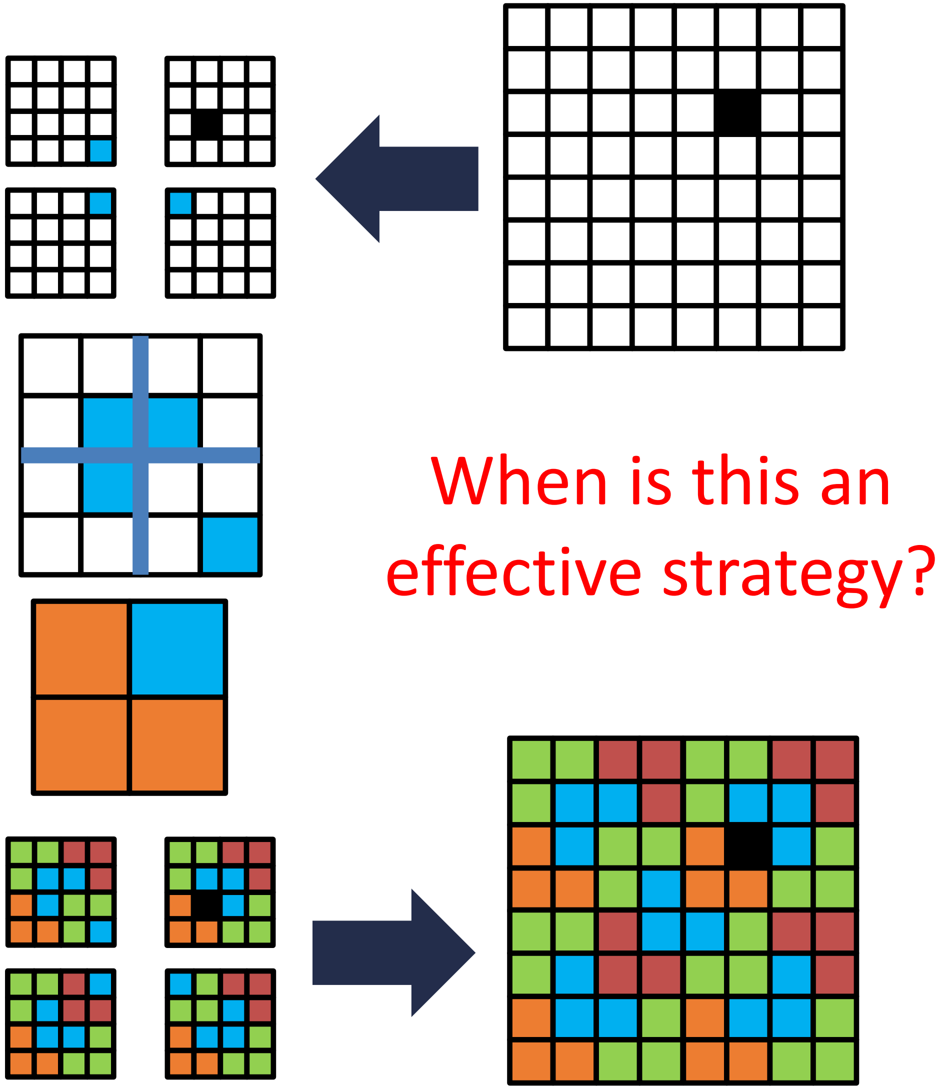
Trominoes Puzzle Solution



Divide and Conquer

[CLRS Chapter 4]

- **Divide**:
  - Break the problem into multiple subproblems, each smaller instances of the original
- **Conquer**:
  - If the suproblems are "large":
    - Solve each subproblem recursively
  - If the subproblems are "small":
    - Solve them directly (base case)
- **Combine**:
  - Merge solutions to subproblems to obtain solution for original problem



Generic Divide and Conquer Solution

```
def myDCalgo(problem):
    if baseCase(problem):
        solution = solve(problem) #brute force if necessary
        return solution
    subproblems = Divide(problem)
    for sub in subproblems:
        subsolutions.append(myDCalgo(sub))
    solution = Combine(subsolutions)
    return solution
```

Merge Sort: Divide and Conquer Sorting

- **Divide**:
  - Break n-element list into two lists of ${}^{n}/_{2}$ elements
- **Conquer**:
  - If $n > 1$:
    - Sort each sublist recursively
  - If $n = 1$:
    - List is already sorted (base case)
- **Combine**:
  - Merge together sorted sublists into one sorted list

Merge

- **Combine**: Merge sorted sublists into one sorted list
- We have:
  - 2 sorted lists (L_1, L_2)
  - 1 output list (L_{out})

While (L_1 and L_2 not empty):

    If $L_1[0] \leq L_2[0]$:

        L_{out}.append(L_1.pop())

    Else:

        L_{out}.append(L_2.pop())

L_{out}.append(L_1)

L_{out}.append(L_2)

$O(n)$

Analyzing Divide and Conquer

1. Break into smaller subproblems
   - Define smaller subproblems, how to divide and combine their results
2. Use recurrence relation to express recursive running time
   - **Divide:** $D(n)$ time,
   - **Conquer:** recurse on small problems, size s
   - **Combine:** $C(n)$ time
   - **Recurrence:**

     $T(n) = D(n) + \sum T(s) + C(n)$
3. Use asymptotic notation to simplify

Analyzing Merge Sort

1. Break into smaller subproblems
2. Use recurrence relation to express recursive running time
3. Use asymptotic notation to simplify

Divide: 0 comparisons

Conquer: recurse on 2 small subproblems, size $\frac{n}{2}$

Combine: n comparisons

Recurrence:

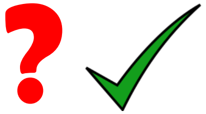
$$T(n) = 2T\left(\frac{n}{2}\right) + n$$

Recurrence Solving Techniques

Four methods for solving recurrences

- Unrolling: expand the recurrence

- Tree: get a picture of recursion

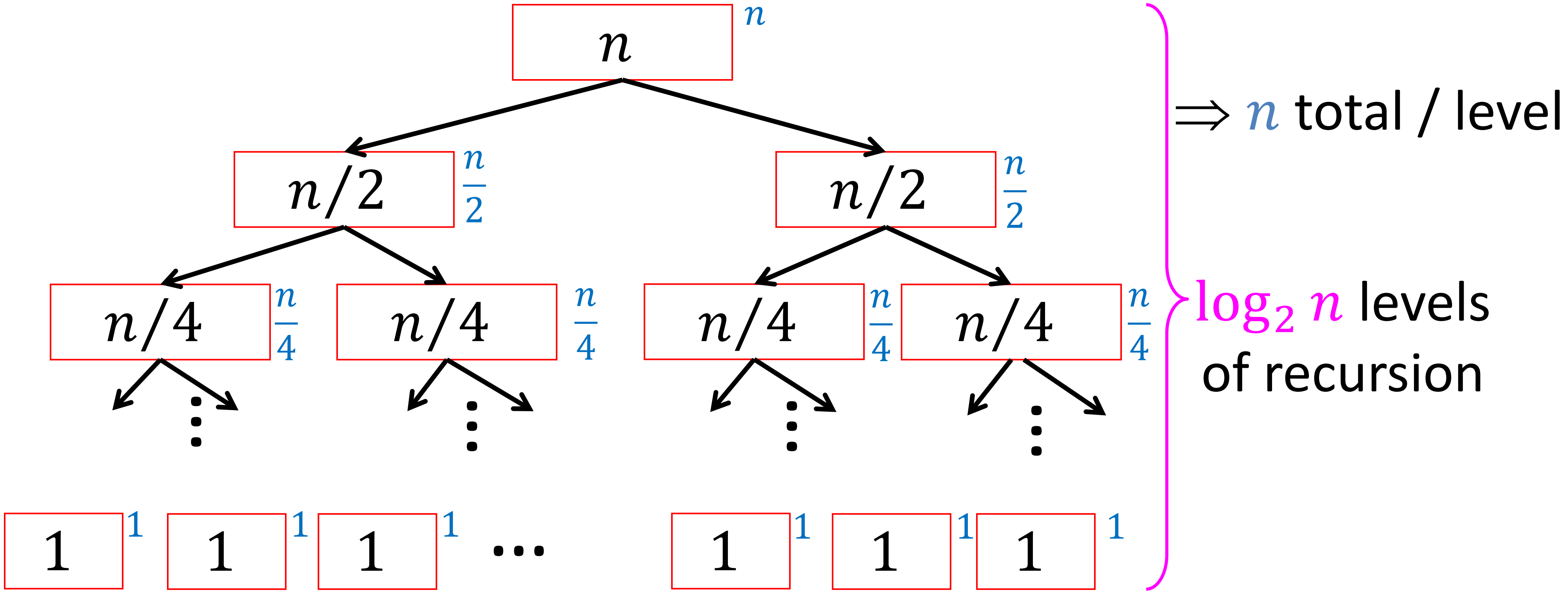


- Guess/Check: Substitution by guessing the solution and using induction to prove

- "Cookbook": Use magic (a.k.a. Master Theorem)

Tree method

$$T(n) = 2T\left(\frac{n}{2}\right) + n$$



$$T(n) = \sum_{i=1}^{\log_2 n} n = n \log_2 n$$