# 15-210
# Parallel and Sequential Algorithms and Data Structures

## Lecture 3

## Algorithmic Techniques and Divide-and-Conquer

# Synopsis

- Algorithmic Techniques
- Divide-and-Conquer
  - Analysis of Costs
- The Maximum Contiguous Subsequence Sum Problem

# Algorithmic Techniques

- Brute Force
  - Try all possibilities
  - Almost always intractable
  - Useful for testing small cases
  - Code usually easy to write

- Reducing one problem to another
  - Transform the structure or the instance of a problem.
  - Shortest Superstring $\rightarrow$ Traveling Salesperson Problem
  - Apply algorithms for the new problem

# Inductive Techniques

- Solve one or more smaller problems to solve the large problem.
- Techniques differ on
  - The number of subproblems
  - How subproblem solutions are used

- Divide-and-Conquer
- Greedy
- Contraction
- Dynamic Programming

# Divide-and-Conquer

- Divide a problem of size $n$ into $k > 1$ problems
  - Sizes $n_1, n_2, \ldots, n_k$
- Solve each problem recursively.
- Combine the subproblem solutions.

# Greedy

- Given a problem of size $n$
- Remove one (or more) elements using a greedy approach
  - Smallest, two smallest, nearest, lowest, etc.
- Solve the remaining smaller problem
  - Usually smaller by 1 or 2 items.

# Contraction

- Given a problem of size $n$
- Generate a significantly smaller (contracted) instance
  - e.g., of size $n/2$
- Solve the smaller instance
- Use the result to solve the original problem.
- One recursive call instead of multiple!

# Dynamic Programming

- Like Divide-and-Conquer
- Solutions to subproblems used multiple times!
- Compute once and store, and then reuse.

# ADTs and Data Structures

- Techniques rely on Abstract Data Types (for functionality)
  - and on data structures that implement them (for costs)

- Sequences, Sets, Tables, Priority Queues, Graphs, Trees, . . .

# Randomization

- Introduce randomness at a choice point
  - Quicksort: choose a pivot randomly

- Testing for primality
  - Miller-Rabin primality test
  - $3/4$ of numbers $< n$ are "witnesses" to $n$'s compositeness.
  - Randomly choose 100 numbers $< n$
  - $P(\text{Failing to find a witness}) = 1 - (\frac{1}{4})^{100}$
  - $P(n \text{ is prime}) = 1 - (\frac{1}{4})^{100} = 0.9999\ldots9327\ldots$

# Synopsis

- Algorithmic Techniques
- Divide-and-Conquer
  - Analysis of Costs
- The Maximum Contiguous Subsequence Sum Problem

# Divide-and-Conquer

- Very versatile.
- Easy to implement.
- Parallelizable
- Code follows the structure of a proof.
- Cost reasoning follows code structure.
  - Recurrences

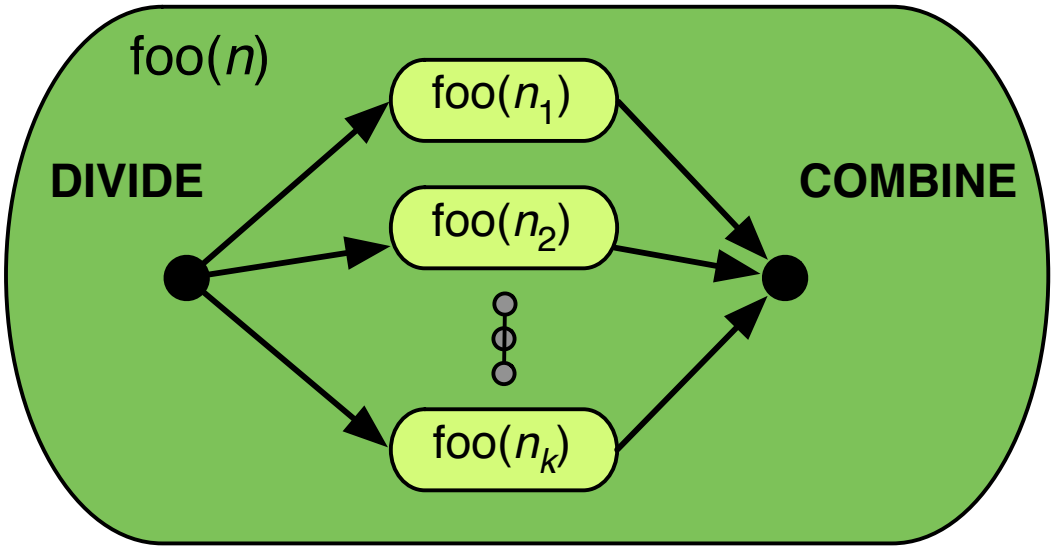
# Strengthening the Problem

- Compute more than "superficially" needed.
- No increase to work or span.
- More efficient combine step.
- At the end, this extra information can be discarded.

# General Structure

- Base case(s)
  - When problem small enough, use a different technique.
  - For example, in quicksort, switch to insertion sort to sort $< 30$ elements.
- Inductive Step
  - Divide into parts
    - Sometimes quite simple: e.g., mergesort
    - Sometimes a bit tricky: e.g., quicksort
  - Solve subproblems (in parallel)
  - Combine results
    - Sometimes quite simple: e.g., quicksort
    - Sometimes a bit tricky: e.g., mergesort
- Costs can be in the divide or combine steps or in both.

# General Structure




$$W(n) = W_{\text{divide}}(n) + \sum_{i=1}^{k} W(n_i) + W_{\text{combine}}(n)$$

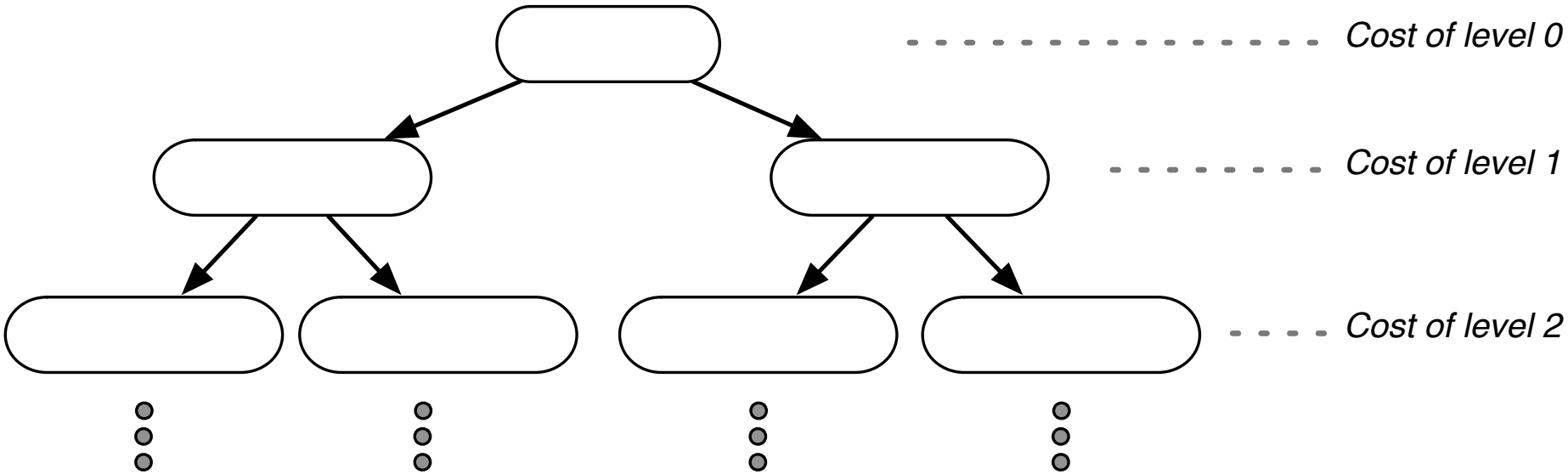
$$S(n) = S_{\text{divide}}(n) + \max_{i=1}^{k} S(n_i) + S_{\text{combine}}(n)$$

# Solving Recurrences

- Tree method (Brick method)
- Substitution method

# The Tree Method

- Expand recurrence into a tree structure.




- Add/Max costs at levels.

# The Tree Method

- Solve $W(n) = 2W(n/2) + O(n)$

- In general, solve

$$W(n) = 2W(n/2) + g(n)$$

where $g(n) \in O(f(n))$

# The Tree Method

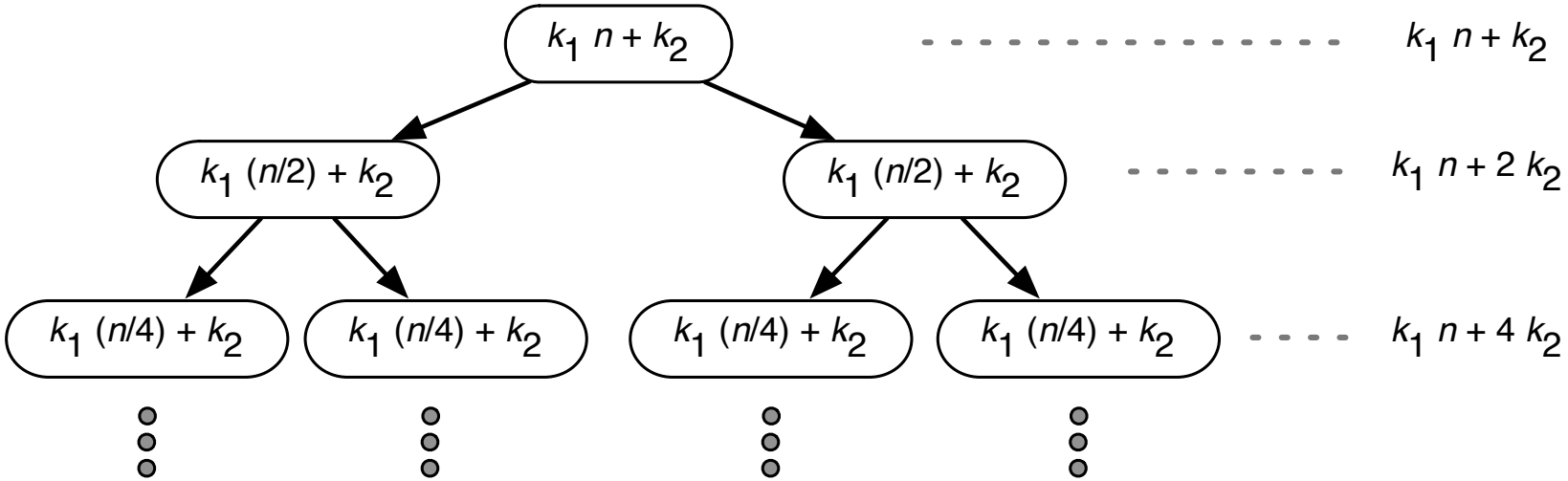
- $g(n) \in O(f(n)) \Rightarrow g(n) \leq c \cdot f(n)$
  - For some $c > 0, N_0 > 0$ and $n \geq N_0$

- $g(n) \leq k_1 \cdot f(n) + k_2$ for some $k_1, k_2$ and $n \geq 1$
  - e.g., $k_1 = c$ and $k_2 = \sum_{i=1}^{N_0} |g(i)|$ (Why?)

- Solve $W(n) \leq 2W(n/2) + k_1 \cdot n + k_2$
  - $f(n) = n$ in our case.

# The Tree Method

- Solving $W(n) \leq 2W(n/2) + k_1 \cdot n + k_2$




- Questions:
  - Number of levels in the tree?
  - Problem size at level $i$?
  - Cost for each node at level $i$?
  - Number of nodes at level $i$?
  - Total cost at level $i$?

# The Tree Method

- Total cost at level $i$ is at most

$$2^i \cdot \left(k_1 \frac{n}{2^i} + k_2\right) \quad = \quad k_1 \cdot n + 2^i \cdot k_2$$

- Total cost over all levels is

$$\begin{aligned} W(n) &\leq \sum_{i=0}^{\log_2 n} \left(k_1 \cdot n + 2^i \cdot k_2\right) \\ &= k_1 n(1 + \log_2 n) + k_2(2^0 + 2^1 + \cdots + 2^{log_2 n}) \\ &\leq k_1 n(1 + \log_2 n) + 2k_2 n \text{ (Why?)} \\ &\in O(n \log n) \end{aligned}$$

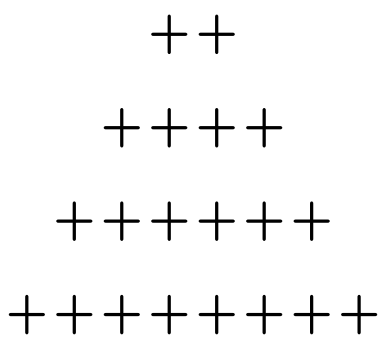
# The Brick Method

- Look at the cost structure at the levels of the cost tree
  - Leaves dominated
  - Balanced
  - Root dominated

# Leaves-Dominated Cost Trees

- For some $\rho > 1$, for all levels $i$

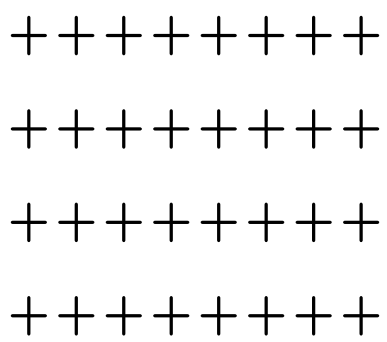
$$\text{cost}_{i+1} \geq \rho \cdot \text{cost}_i$$



- Overall cost is $O(\text{cost}_d)$ where $d$ is the depth.

# Balanced Cost Trees

- All levels have about the same cost

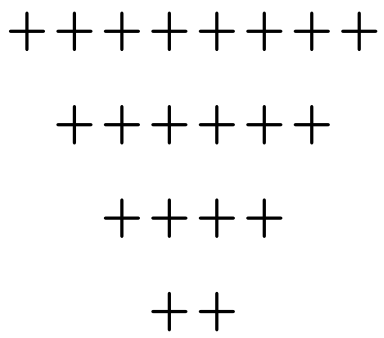


- Overall cost is $O(d \cdot \max_i \text{cost}_i)$ where $d$ is the depth.

# Root-Dominated Cost Trees

- For some $\rho < 1$, for all levels $i$

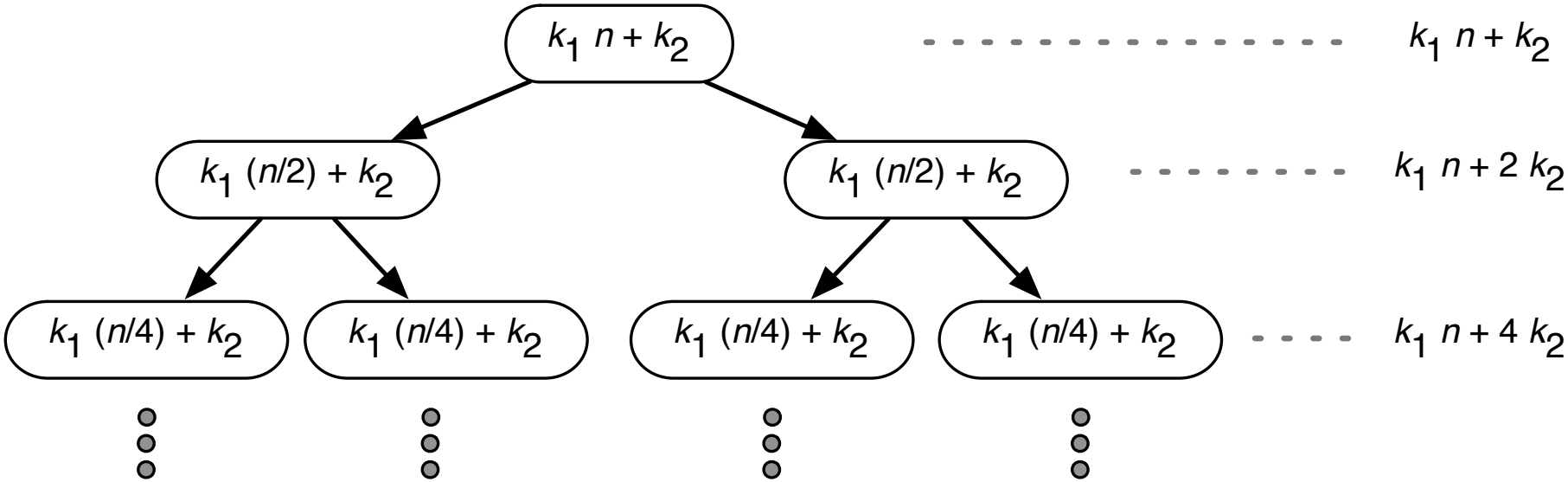
$$\text{cost}_{i+1} \leq \rho \cdot \text{cost}_i$$



- Overall cost is $O(\text{cost}_0)$ where $d$ is the depth.

# The Brick Method

- What type of a cost tree is this?



# Synopsis

- Algorithmic Techniques
- Divide-and-Conquer
  - Analysis of Costs
- The Maximum Contiguous Subsequence Sum Problem

# The MCSS Problem

## The Maximum Contiguous Subsequence Sum Problem

- Given a sequence of numbers $S = \langle s_1, \ldots, s_n \rangle$,
- Find

$$\text{mcss}(S) = \max_{1 \le i \le j \le n} \left\{ \sum_{k=i}^{j} s_k \right\}$$

- $S = \langle 0, -1, \mathbf{2}, -\mathbf{1}, \mathbf{4}, -1, 0 \rangle$, $mcss(S) = 5$
- How many possible subsequences are there?
- All positive numbers?

# Brute Force Algorithm

- Compute the sum of all $O(n^2)$ possible subsequences (in parallel)
  - Use plus reduce
- Subsequence $(i, j)$ needs
  - $O(j - i)$ work (Why?)
  - $O(\log(j - i))$ span (Why?)

$$
\begin{aligned}
W(n) &= 1 + \sum_{1 \le i \le j \le n} W_{\text{reduce}}(j - i) \le 1 + n^2 \cdot W_{\text{reduce}}(n) \\
&= 1 + n^2 \cdot O(n) \in O(n^3) \\
S(n) &= 1 + \max_{1 \le i \le j \le n} S_{\text{reduce}}(j - i) \le 1 + S_{\text{reduce}}(n) \in O(\log n)
\end{aligned}
$$

# Brute Force Algorithm

- Compute maximum over all $O(n^2)$ sums
  - Use max reduce
  - Needs $O(n^2)$ work and $O(\log n)$ span
  - Can be ignored (Why?)

- Total costs for brute force are:
  - $O(n^3)$ work
  - $O(\log n)$ span

# Divide-and-Conquer – I

$$\langle \text{——} \; L \; \text{——} \parallel \text{——} \; R \; \text{——} \rangle$$

$$\Downarrow$$

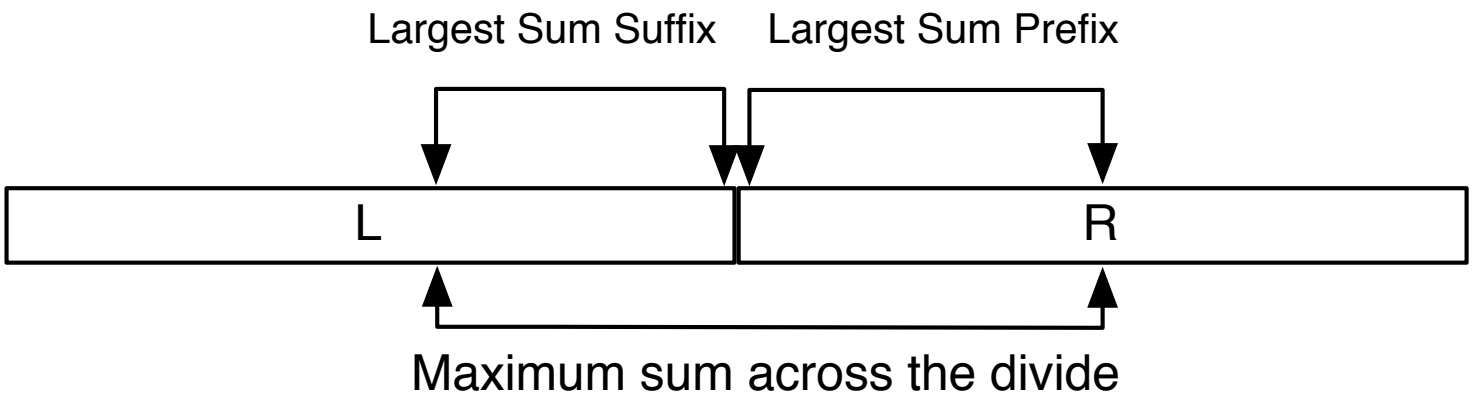
$$L = \underbrace{\langle \quad \cdots \quad \rangle}_{\text{mcss}=56} \qquad R = \underbrace{\langle \quad \cdots \quad \rangle}_{\text{mcss}=17}$$

- Let's solve $S = \langle -2, -1, 2, 3, 2, -2 \rangle$
- Is this right?
- How do we combine subproblem results?

# Divide-and-Conquer – I

- Recursion handles
  - When mcss(S) subsequence is in the left.
  - When mcss(S) subsequence is in the right.

- What happens when mcss(S) spans across the divide point?



# Divide-and-Conquer – I

```
1  fun mcss(s) =
2     case (showt s)
3        of EMPTY = −∞
4         | ELT(x) = x
5         | NODE(L, R) =
6              let   (m_L, m_R) = ( mcss(L) || mcss(R) )
7                    m_A = bestAcross(L, R)
8              in  max{m_L, m_R, m_A}
9              end
```

- $W(n) = 2W(n/2) + O(n)$ (Why?) $\rightarrow W(n) \in O(n \log n)$
- $S(n) = S(n/2) + O(\log n)$ (Why?) $\rightarrow S(n) \in O(\log^2 n)$

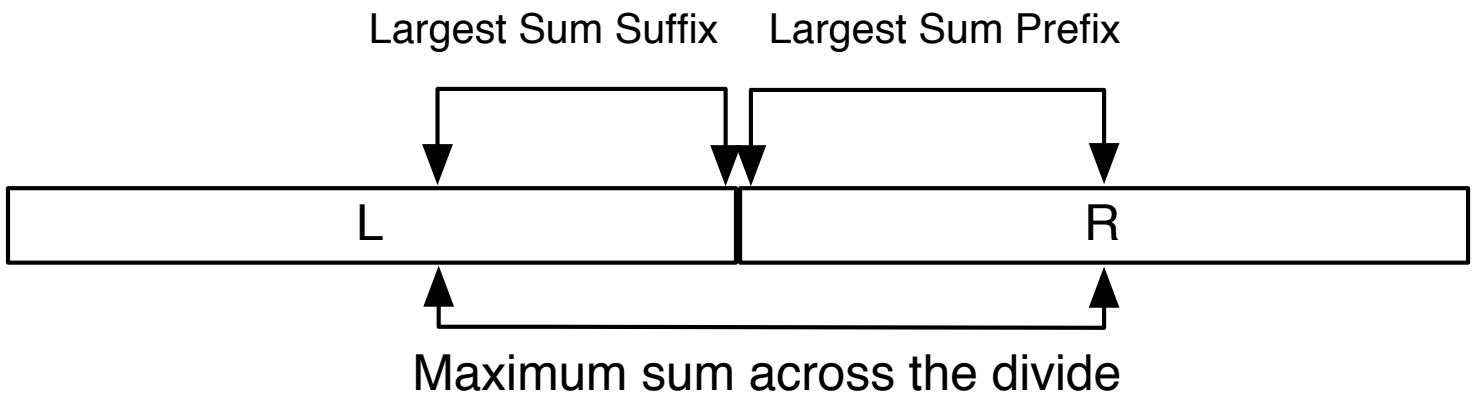
# Divide-and-Conquer – II

## Important Questions

- Can we do better than $O(n \log n)$ work?
- What part of the divide-and-conquer is the bottleneck?
  - Combine takes linear work? (Why?)
- How can we improve?

# Divide-and-Conquer – II

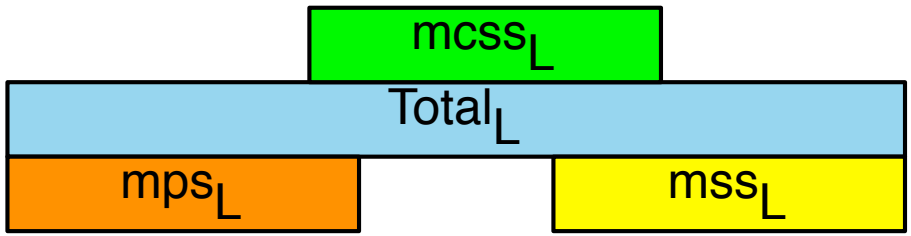
- The answers lie here




- Strengthen the subproblems
  - Compute additional information

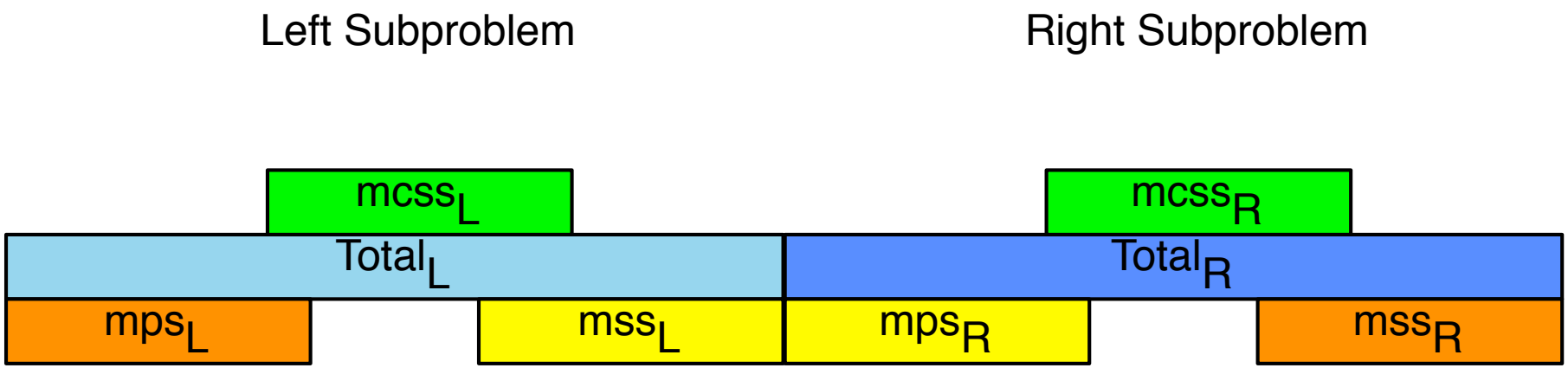
# Divide-and-Conquer – II

Left Subproblem

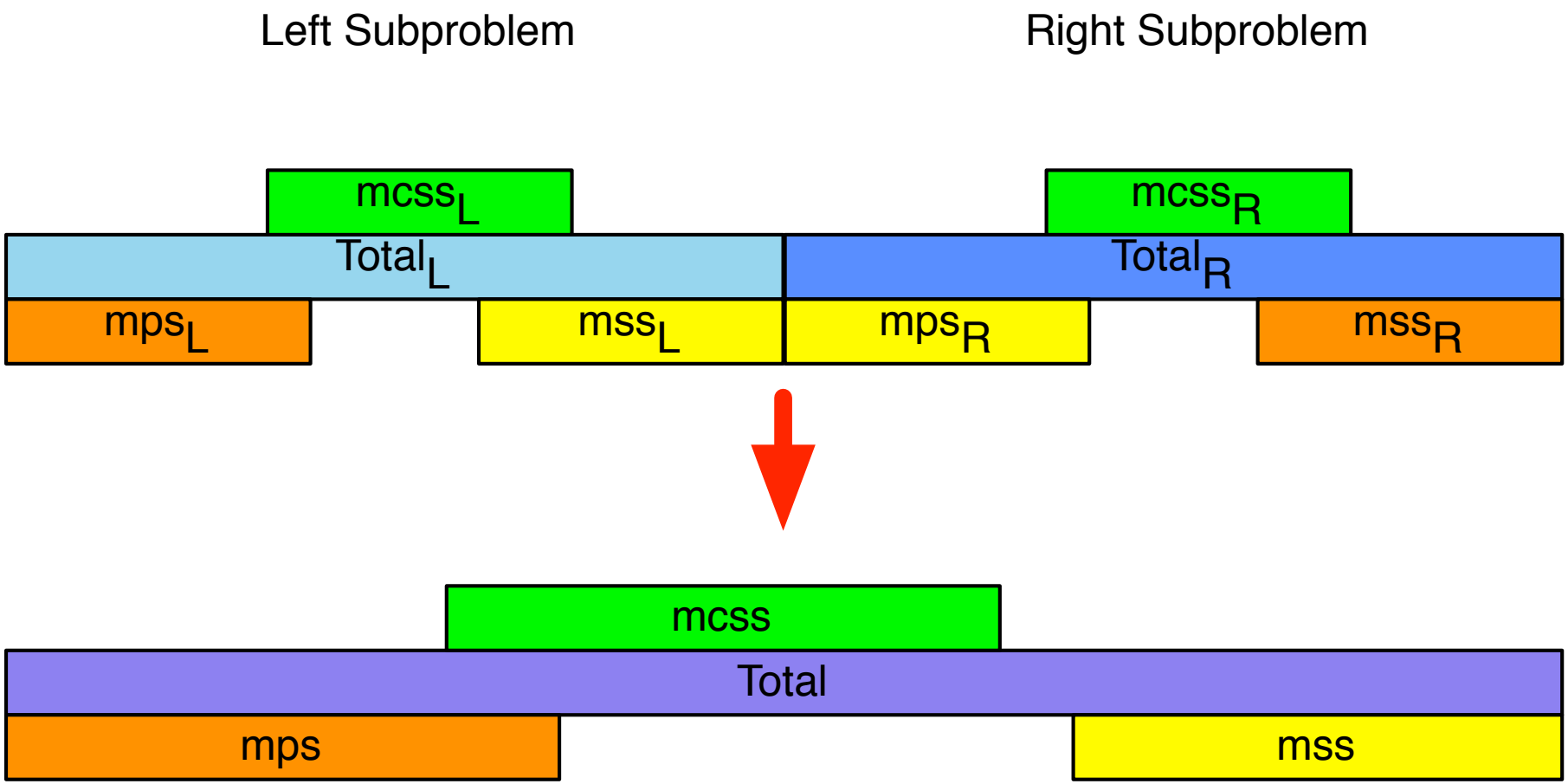



mps = maximum prefix sum  mss = maximum suffix sum

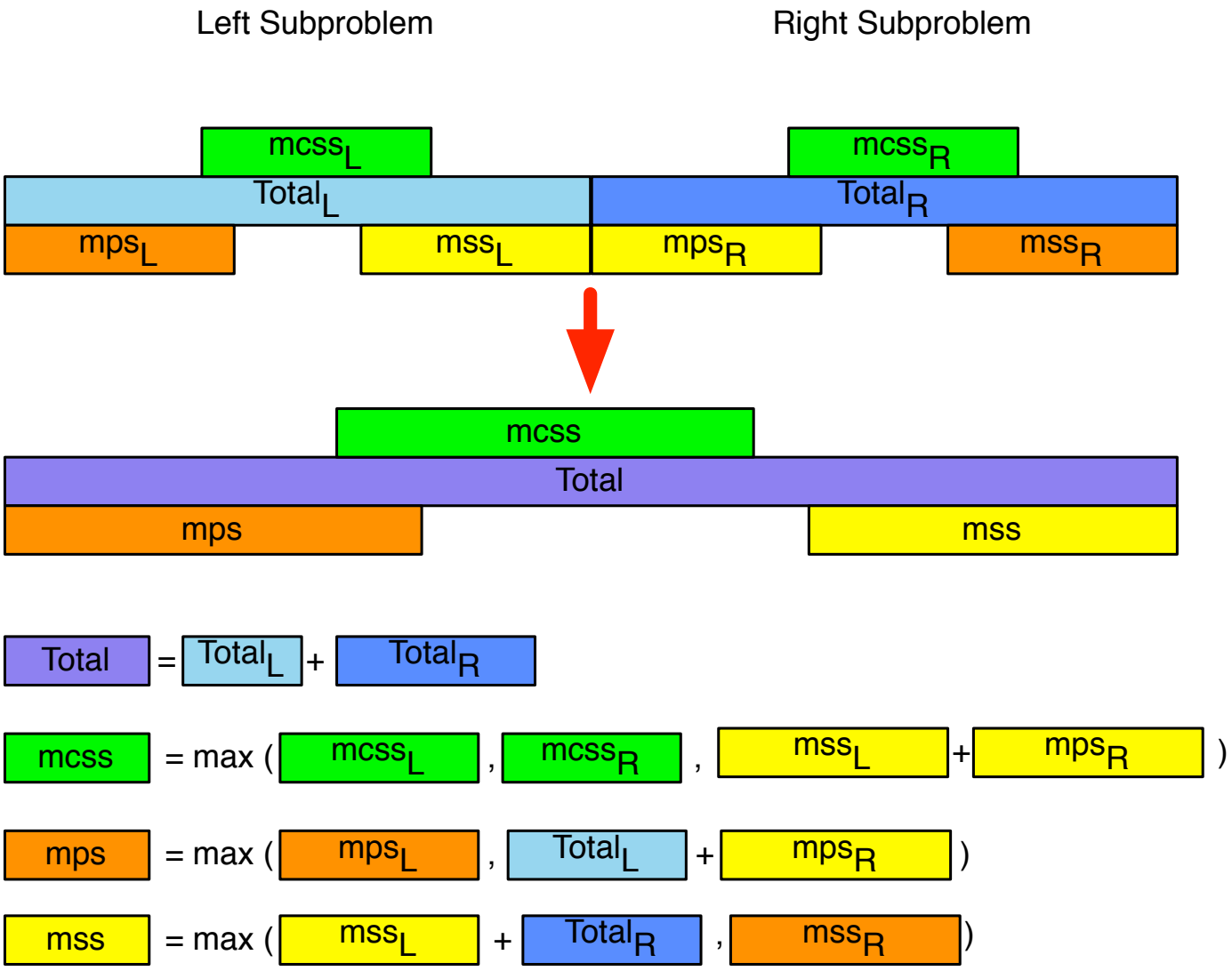
# Divide-and-Conquer – II



# Divide-and-Conquer – II



# Divide-and-Conquer – II



# Divide-and-Conquer – II

```
1   fun mcss(a) =
2   let
3     fun mcss'(a)
4       case (showt a)
5         of EMPTY = (−∞, −∞, −∞, 0)
6          | ELT(x) = (x, x, x, x)
7          | NODE(L, R) =
8              let
9                  ((m1, p1, s1, t1), (m2, p2, s2, t2)) = ( mcss'(L) || mcss'(R) )
10             in
11               (max(s1 + p2, m1, m2), max(p1, t1 + p2), max(s1 + t2, s2), t1 + t2)
12             end
13      (m, p, s, t) = mcss'(a)
14  in m end
```

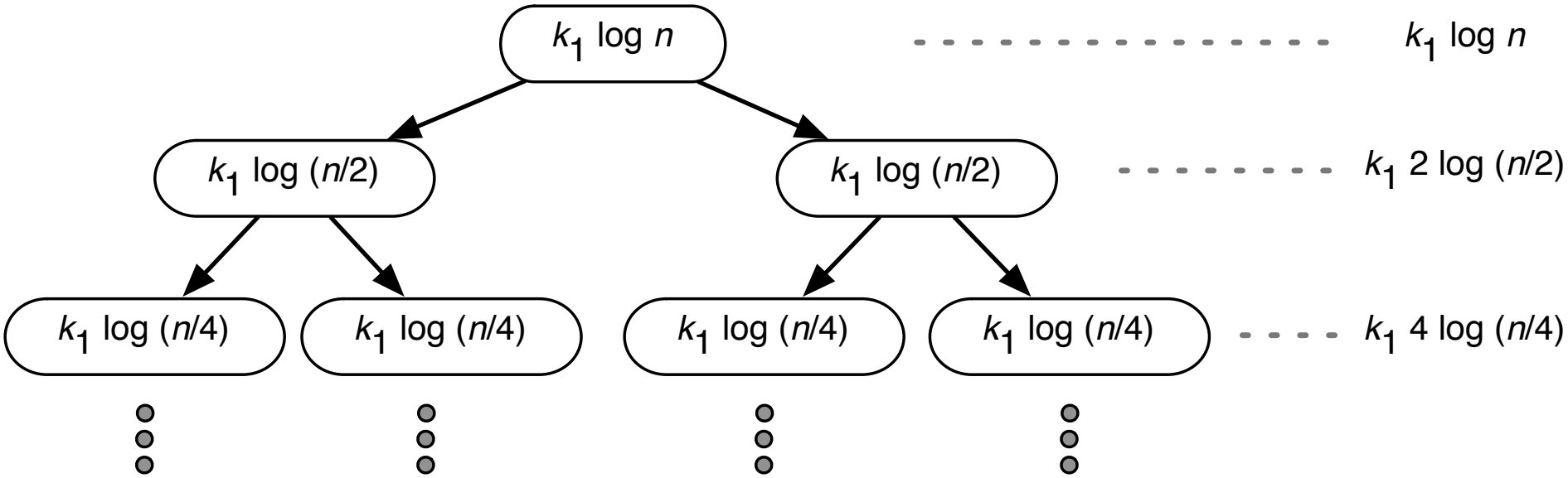
# Cost Analysis

```
1  fun mcss(a) =
2  let
3    fun mcss'(a)
4      case (showt a)
5        of EMPTY = (−∞, −∞, −∞, 0)
6         | ELT(x) = (x, x, x, x)
7         | NODE(L, R) =
8           let
9              ((m1, p1, s1, t1), (m2, p2, s2, t2)) = ( mcss'(L) || mcss'(R) )
10          in
11           (max(s1 + p2, m1, m2), max(p1, t1 + p2), max(s1 + t2, s2), t1 + t2)
12          end
13     (m, p, s, t) = mcss'(a)
14 in m end
```

- Assuming `showt` has $O(\log n)$ work and span.
  - $W(n) = 2W(n/2) + O(\log n)$
  - $S(n) = S(n/2) + O(\log n)$

# Cost Analysis

- $W(n) = 2W(n/2) + O(\log n)$




- $W(n) \leq \sum_{i=0}^{\log n} k_1 2^i \log(n/2^i)$

# Substitution Method

- Solve $W(n) \leq 2W(n/2) + k \cdot \log n$
  - $k > 0$
  - $W(n) \leq k$ for $n \leq 1$
- Guess $W(n) \leq \kappa_1 n - \kappa_2 \log n - \kappa_3$
  - Need to find $\kappa_1$, $\kappa_2$, and $\kappa_3$.
- Base case: $W(1) \leq k \Rightarrow \kappa_1 - \kappa_3 \leq k$

# Substitution Method

- Inductive Step

$$
\begin{aligned}
W(n) &\leq 2W(\frac{n}{2}) + k \cdot \log n \\
&\leq 2(\kappa_1 \frac{n}{2} - \kappa_2 \log(\frac{n}{2}) - \kappa_3) + k \cdot \log n \\
&= \kappa_1 n - 2\kappa_2(\log n - 1) - 2\kappa_3 + k \cdot \log n \\
&= (\kappa_1 n - \kappa_2 \log n - \kappa_3) + (k \log n - \kappa_2 \log n + 2\kappa_2 - \kappa_3) \\
&\leq \kappa_1 n - \kappa_2 \log n - \kappa_3
\end{aligned}
$$

- Choose $\kappa_2 = k$ and $2\kappa_2 - \kappa_3 \leq 0$ (Why?)
- For example, $\kappa_2 = k, \kappa_1 = 3k, \kappa_3 = 2k$ satisfies the constraints.

# Summary

- Algorithmic Paradigms
- Divide-and-Conquer
  - General Form
  - Cost Analysis
  - Tree and Brick Methods
  - Substitution Method
- Maximum Contiguous Subsequence Problem
  - Brute Force
  - Divide-and-Conquer
  - Divide-and-Conquer with Subproblem Strengthening