CS 4102: Algorithms

Lecture 3: Karatsuba, Tree Method

David Wu

Fall 2019

Warm Up

Simplify:

$$(1 + a + a^2 + a^3 + a^4 + \cdots + a^L)(a - 1) = ?$$

$$(a + a^2 + a^3 + a^4 + a^5 + \cdots + a^L + a^{L+1}) +$$

$$(-a - a^2 - a^3 - a^4 - a^5 - \cdots - a^L - 1) =$$

$$a^{L+1} - 1$$

$$\sum_{i=0}^{L} a^i = \frac{a^{L+1} - 1}{a - 1}$$

Today’s Keywords

Divide and Conquer

Recurrences

Merge Sort

Karatsuba

Tree Method

CLRS Readings: Chapter 4

Homeworks

HW0 due **Today, 11pm**

- Submit 2 attachments (**zip** and **pdf**)

HW1 due **Thursday, September 12, 11pm**

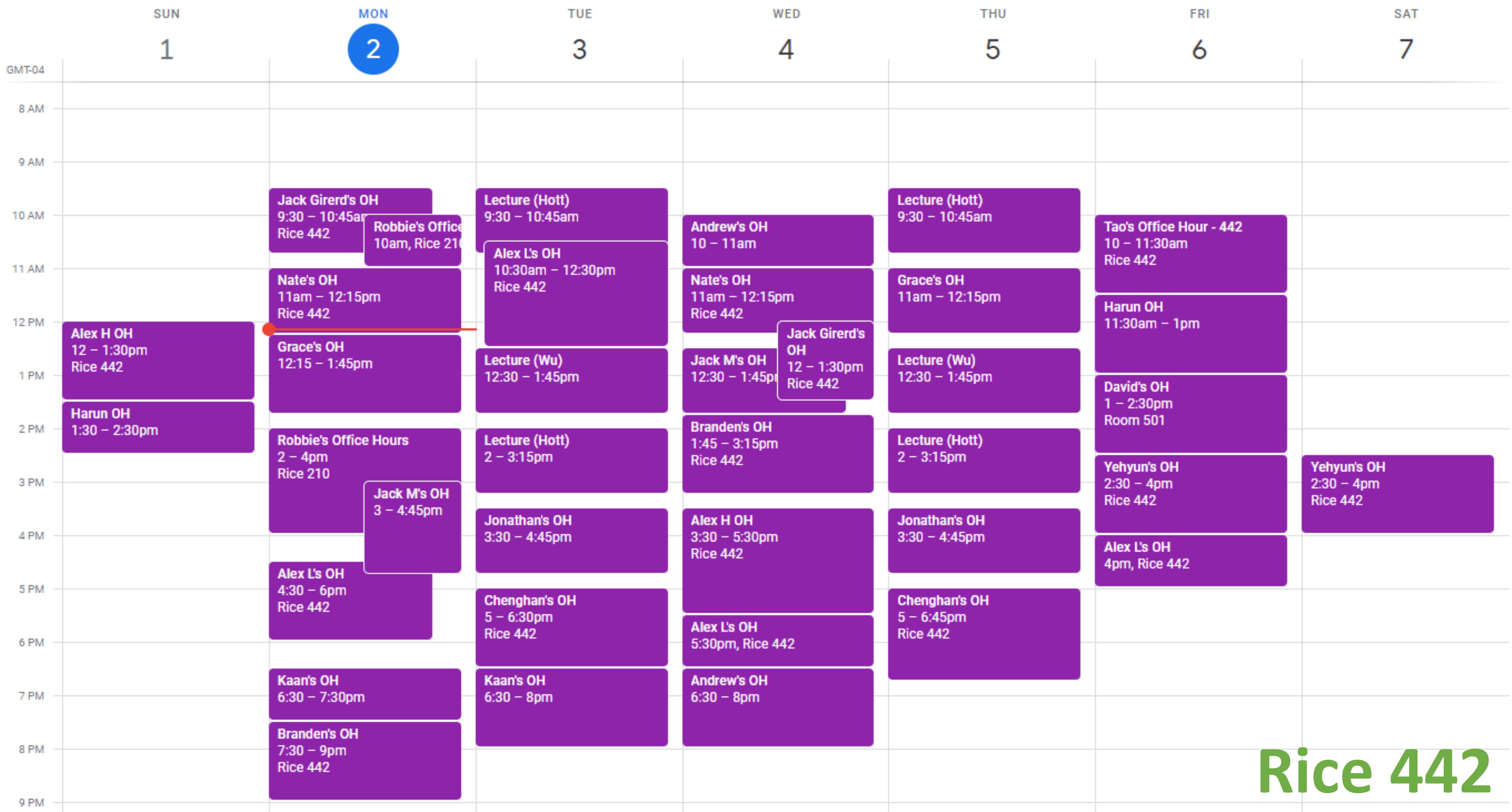
- Start early!
- Written (use Latex!) – Submit both **zip** and **pdf**!
- Asymptotic notation
- Recurrences
- Divide and Conquer

Homework Help Algorithm

Algorithm: How to ask a question about homework (efficiently)

1. Check to see if your question is already on Piazza
2. If it is not on Piazza, ask on Piazza
3. Look for other questions you know the answer to, and provide answers to any that you see
4. TA office hours
5. Instructor office hours
6. Email, set up a meeting

Office Hours



Rice 442

Divide and Conquer

[CLRS Chapter 4]

Divide:

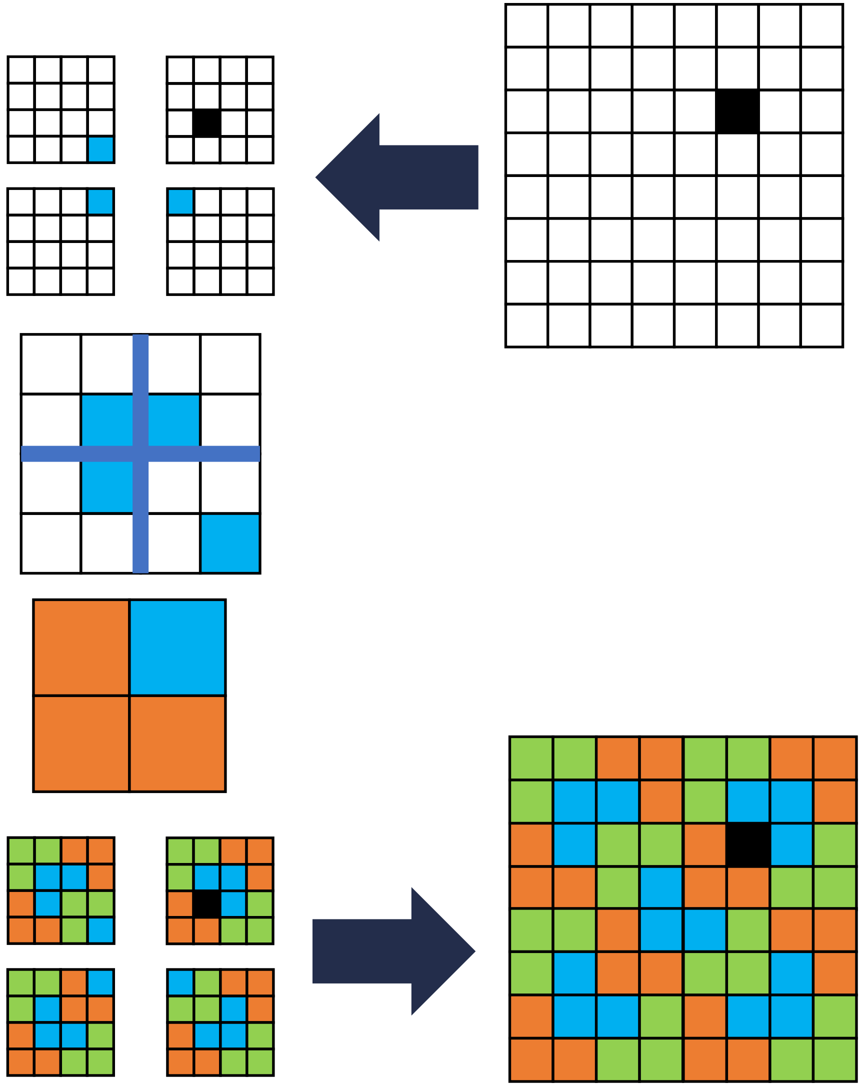
- Break the problem into multiple subproblems, each smaller instances of the original

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



Analyzing Divide and Conquer

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

Divide: $D(n)$ time

Conquer: Recurse on smaller problems of size $s_1, \ldots, s_k$

Combine: $C(n)$ time

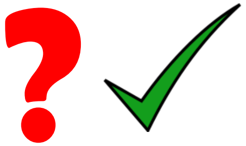
Recurrence:

- $T(n) = D(n) + \sum_{i \in [k]} T(s_i) + C(n)$

Recurrence Solving Techniques

Tree

get a picture of recursion



Guess/Check

guess and use induction to prove

"Cookbook"

MAGIC!

Substitution

substitute in to simplify

Merge Sort

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

Inputs:

- 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)

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 problems, size $\frac{n}{2}$

Combine: n comparisons

Recurrence:

- $T(n) = 2T(n/2) + n$

Recurrence Solving Techniques

Tree

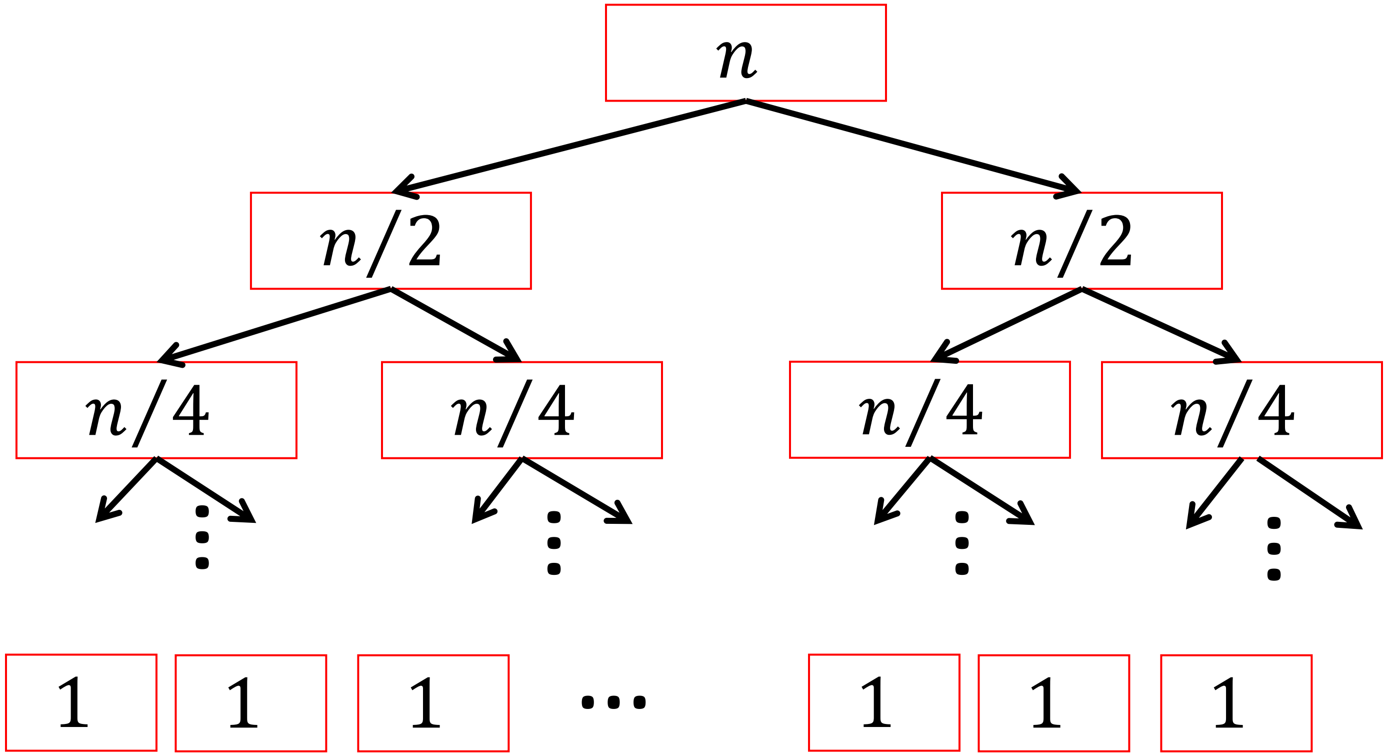
Guess/Check

"Cookbook"

Substitution

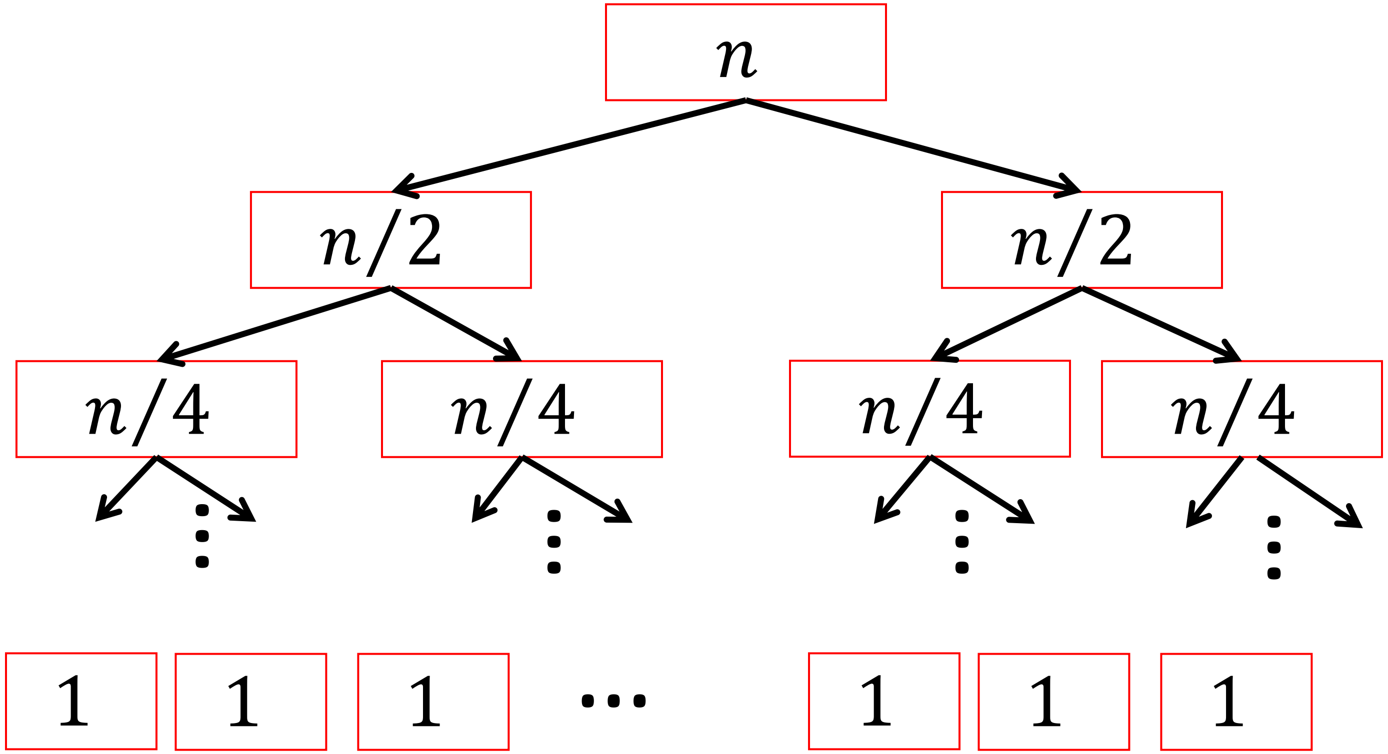
Tree Method

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



Tree Method

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

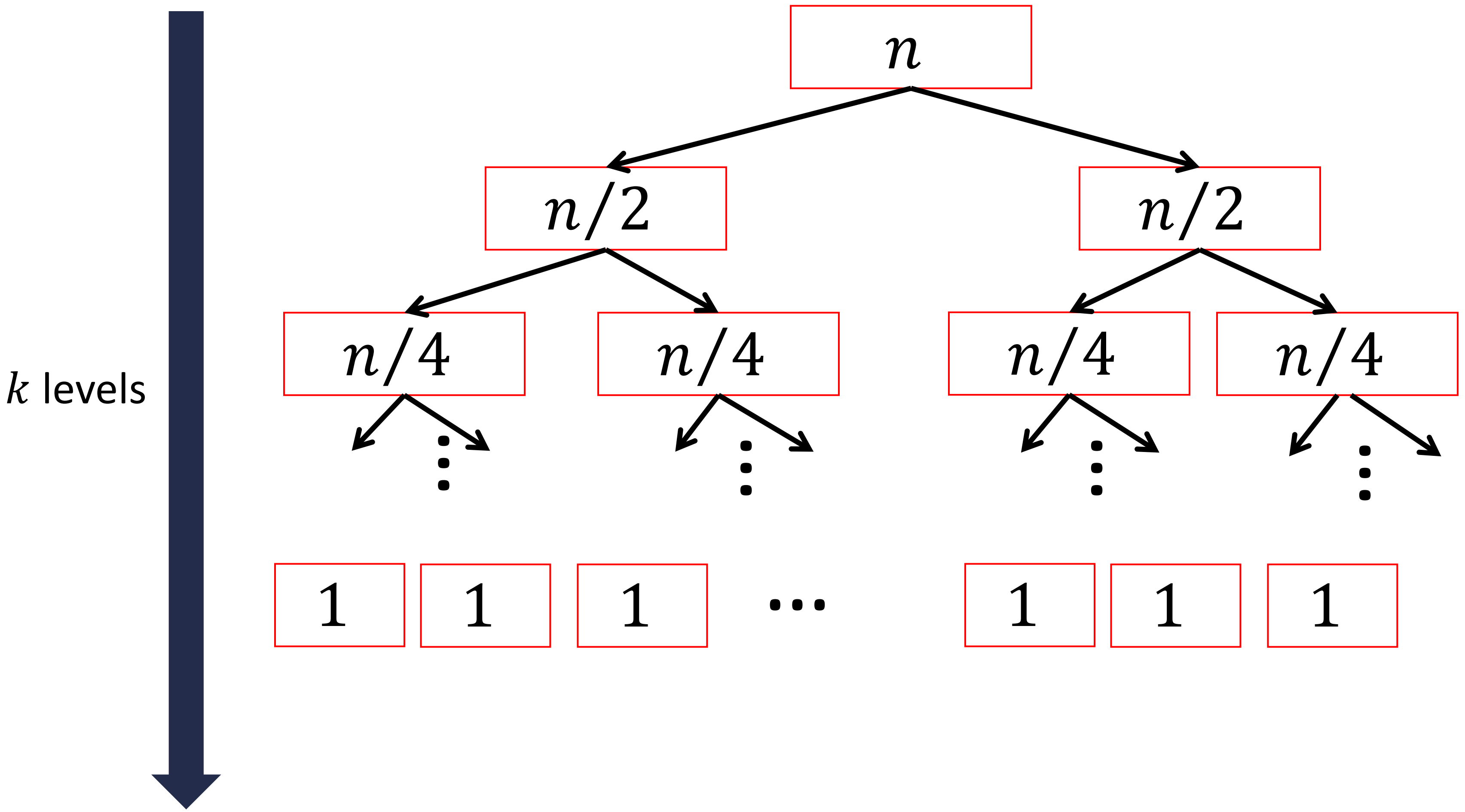


Number of subproblems

1

Tree Method

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



Number of subproblems	Cost of subproblem
1	n
2	$n/2$
4	$n/4$
2^k	$\frac{n}{2^k} = 1$

Tree Method

3. Use asymptotic notation to simplify

$$T(n) = 2T(n/2) + n$$

How many levels?

Problem size at k^{th} level: $\frac{n}{2^k}$

Base case: $n = 1$

At level k, it should be the case that $\frac{n}{2^k} = 1$

$$n = 2^k \Rightarrow k = \log_2 n$$

Number of subproblems	Cost of subproblem
1	n
2	$n/2$
4	$n/4$
2^k	$\frac{n}{2^k} = 1$

Tree Method

3. Use asymptotic notation to simplify

$$T(n) = 2T(n/2) + n$$

$$k = \log_2 n$$

What is the cost?

Cost at level i: $2^i \cdot \frac{n}{2^i} = n$

Total cost: $T(n) = \sum_{i=0}^{\log_2 n} n = n \sum_{i=0}^{\log_2 n} 1 = n \log_2 n$

$$= \Theta(n \log n)$$

Number of subproblems	Cost of subproblem
1	n
2	$n/2$
4	$n/4$
2^k	$\frac{n}{2^k} = 1$

Multiplication

Want to multiply large numbers together

$$\begin{array}{r} 4\,1\,0\,2 \\ \times\,1\,8\,1\,9 \\ \hline \end{array}$$

n-digit numbers

How do we measure input size?

number of digits

What do we "count" for run time?

number of <u>elementary</u> operations
(single-digit multiplications)

"Schoolbook" Multiplication

How many multiplications?

$$\begin{array}{r} 4\,1\,0\,2 \\ \times\ \boxed{1}\,\boxed{8}\,\boxed{1}\,\boxed{9} \\ \hline 3\,6\,9\,1\,8 \\ 4\,1\,0\,2 \\ 3\,2\,8\,1\,6 \\ +\ 4\,1\,0\,2 \\ \hline 7\,4\,6\,1\,5\,3\,8 \end{array}$$

n-digit numbers

n mults

n mults

n mults

n mults

n levels

$\Rightarrow \Theta(n^2)$

What about cost of additions?

$\Theta(n^2)$

“Schoolbook” Multiplication

Can we do better?

How many multiplications?

$$
\begin{array}{r}
4\,1\,0\,2 \\
\times\,1\,8\,1\,9 \\
\hline
3\,6\,9\,1\,8 \\
4\,1\,0\,2 \\
3\,2\,8\,1\,6 \\
+\;4\,1\,0\,2 \\
\hline
7\,4\,6\,1\,5\,3\,8
\end{array}
$$

n-digit numbers

n mults

n mults

n mults

n mults

n levels $\Rightarrow \Theta(n^2)$

What about cost of additions?

$\Theta(n^2)$

Divide and Conquer Multiplication

1. Break into smaller subproblems

$$\begin{array}{r} \boxed{a}\,\boxed{b} \\ \times\ \boxed{c}\,\boxed{d} \\ \hline \end{array} \quad \begin{array}{l} = 10^{\frac{n}{2}}\,\boxed{a} + \boxed{b} \\ = 10^{\frac{n}{2}}\,\boxed{c} + \boxed{d} \end{array}$$

$$= \quad 10^{n}\left(\boxed{a} \times \boxed{c}\right) +$$

$$10^{\frac{n}{2}}\left(\boxed{a} \times \boxed{d} + \boxed{b} \times \boxed{c}\right) +$$

$$\left(\boxed{b} \times \boxed{d}\right)$$

Divide and Conquer Multiplication

Divide:

- Break n-digit numbers into four numbers of $n/2$ digits each (call them a, b, c, d)

Conquer:

- If $n > 1$:
  - Recursively compute ac, ad, bc, bd
- If $n = 1$: (i.e. one digit each)
  - Compute ac, ad, bc, bd directly (base case)

Combine:

- $10^n(ac) + 10^{n/2}(ad + bc) + bd$

For simplicity, assume that $n = 2^k$ is a power of 2

Divide and Conquer Multiplication

2. Use recurrence relation to express recursive running time

$$10^n(ac) + 10^{n/2}(ad + bc) + bd$$

Recursively solve

$$T(n)$$

Divide and Conquer Multiplication

2. Use recurrence relation to express recursive running time

$$10^n(ac) + 10^{n/2}(ad + bc) + bd$$

Recursively solve

$$T(n) = 4T\left(\frac{n}{2}\right)$$

Need to compute 4 multiplications, each of size $n/2$

Divide and Conquer Multiplication

2. Use recurrence relation to express recursive running time

$$10^n(ac) + 10^{n/2}(ad + bc) + bd$$

Recursively solve

$$T(n) = 4T\left(\frac{n}{2}\right) + 5n$$

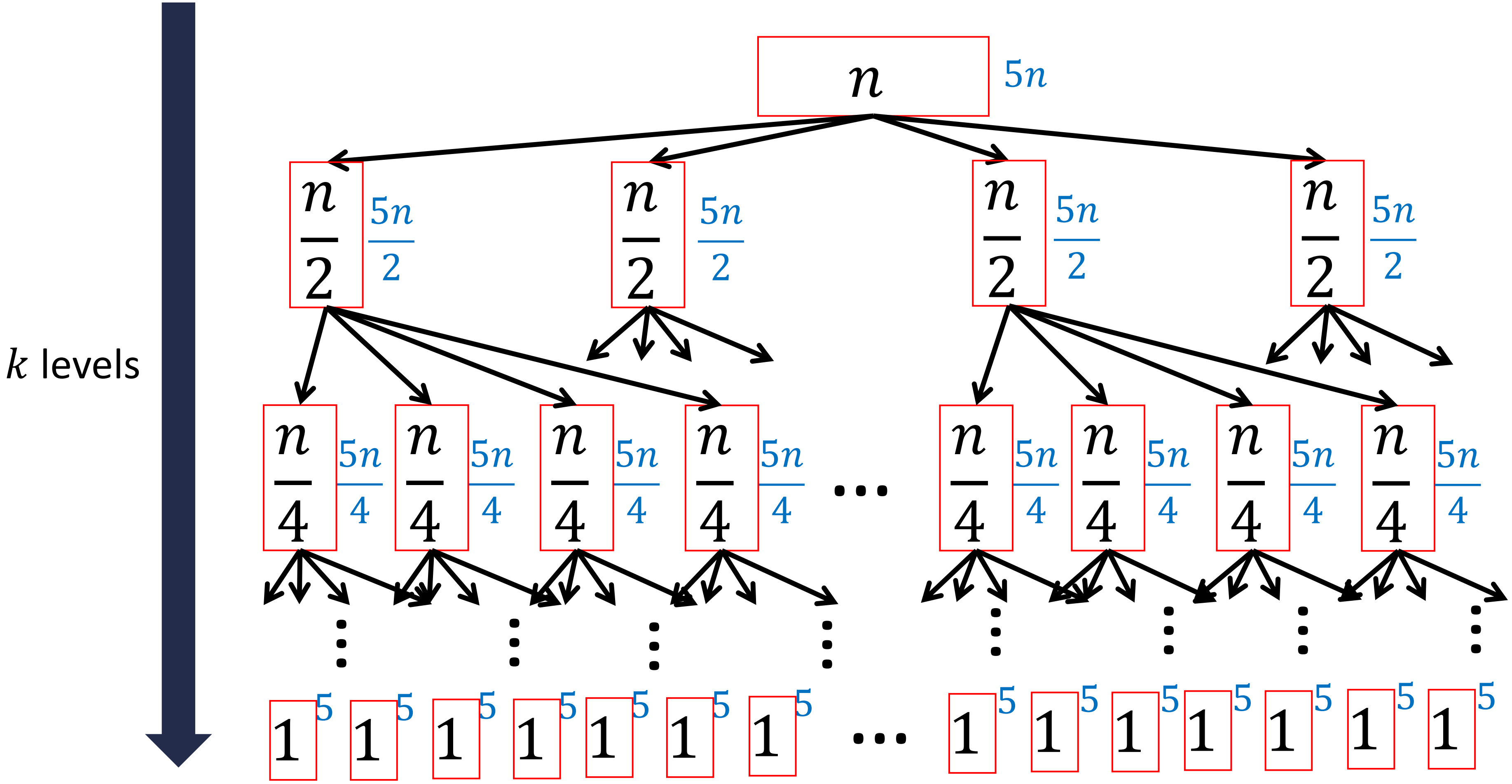
Need to compute 4 multiplications, each of size $n/2$

2 shifts and 3 additions on n-bit values

Divide and Conquer Multiplication

3. Use asymptotic notation to simplify

$$T(n) = 4T(n/2) + 5n$$



Number of subproblems	Cost of subproblem
1	$5n$
4	$\frac{5n}{2}$
16	$\frac{5n}{4}$
4^k	$\frac{5n}{2^k} = 5$

Divide and Conquer Multiplication

3. Use asymptotic notation to simplify

$$T(n) = 4T(n/2) + 5n$$

Number of subproblems	Cost of subproblem
1	$5n$
4	$\frac{5n}{2}$
16	$\frac{5n}{4}$
4^k	$\frac{5n}{2^k} = 5$

How many levels?

Problem size at k^{th} level: $\frac{n}{2^k}$

Base case: $n = 1$

At level k, it should be the case that $\frac{n}{2^k} = 1$

$$n = 2^k \Rightarrow k = \log_2 n$$

Divide and Conquer Multiplication

3. Use asymptotic notation to simplify

$$T(n) = 4T(n/2) + 5n$$

$$k = \log_2 n$$

What is the cost?

Cost at level i: $4^i \cdot \frac{5n}{2^i} = 2^i \cdot 5n$

Total cost: $T(n) = \sum_{i=0}^{\log_2 n} 2^i \cdot 5n = 5n \sum_{i=0}^{\log_2 n} 2^i$

Number of subproblems	Cost of subproblem
1	$5n$
4	$\frac{5n}{2}$
16	$\frac{5n}{4}$
4^k	$\frac{5n}{2^k} = 5$

Divide and Conquer Multiplication

3. Use asymptotic notation to simplify

$$\sum_{i=0}^{L} a^i = \frac{a^{L+1} - 1}{a - 1}$$

$$T(n) = 4T(n/2) + 5n$$

$$= 5n \sum_{i=0}^{\log_2 n} 2^i$$

$$= 5n \cdot \frac{2^{\log_2 n + 1} - 1}{2 - 1}$$

$$= 5n(2n - 1) = \Theta(n^2)$$

No better than the schoolbook method!

Divide and Conquer Multiplication

3. Use asymptotic notation to simplify

$$T(n) = 4T(n/2) + 5n$$

$$= 5n \sum_{i=0}^{\log_2 n} 2^i$$

$$= 5n \cdot \frac{2^{\log_2 n+1} - 1}{2 - 1}$$

$$= 5n(2n - 1) = \Theta(n^2)$$

$$\sum_{i=0}^{L} a^i = \frac{a^{L+1} - 1}{a - 1}$$

Is there a $o(n^2)$ algorithm for multiplication?