Data Structures and Algorithms

Khazhak Galstyan

Who am I

Khazhak Galstyan

- Yerevan State University B.S. in Informatics and Applied Mathematics
- Currently doing M.S. (Discrete Mathematics and Theoretical Informatics)
- Worked @ CodeSignal, SuperAnnotate, Huawei
- Researcher @ YerevaNN

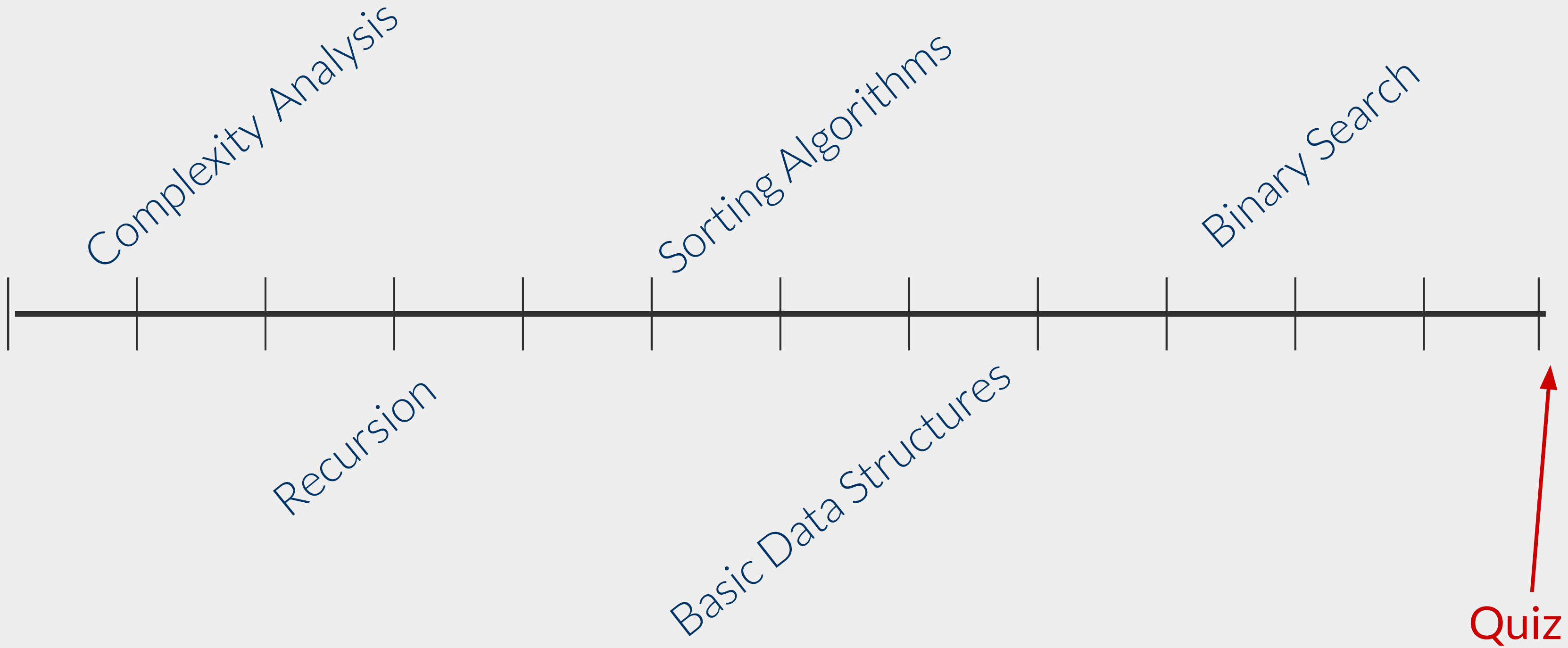
What will you learn (hopefully)

- **Analyse algorithms**
  - How well does an algorithm perform? Is it fast? Is it correct? Provable?
- **Design algorithms**
  - Algorithm design is in the foundation of any programming-related problem and patterns, paradigms and data structures discussed in this class will help you solve algorithmic problems in the future.
- **Talk about algorithms**
  - We'll develop technical language and tools to communicate about data structures and algorithms with partners in crime.

What will you have to do

- Homeworks - 1 to 3 hour homeworks after each class
- Quizzes - 2 2-hour quizzes after classes 5 and 10.

Our Class Overview



Session 1:

Complexity Analysis and a beautiful example

First Problem: Multiplication!

Multiplication: The Problem

Input: Two non-negative integers (x and y)

Output: Their product $(x \cdot y)$

$$453 \times 86 = 38958$$

Multiplication: Elementary School Approach

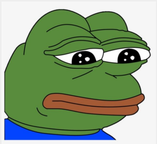
$$\begin{array}{r} 453 \\ \times \quad 87 \\ \hline 3171 \\ 36240 \\ \hline 39411 \end{array}$$

The Algorithm*: Compute all partial products and sum them all up accordingly.

* This is an ugly way to describe an algorithm. Avoid any possible vagueness in an algorithm description.

Multiplication: Elementary School Approach

$$
\begin{array}{r}
45385327687532875567463874566 \\
\times \quad 87734659827649875623498576387 \\
\hline
\end{array}
$$



Multiplication: Efficiency of the Algorithm

$$
\begin{array}{r}
45385327687532875567463874566 \\
\times \quad \underbrace{87734659827649875623498576387}_{\textbf{n}\ \text{digits}}
\end{array}
$$

Multiplication: Efficiency of the Algorithm

$$
\begin{array}{r}
45385327687532875567463874566 \\
\times \quad \underbrace{87734659827649875623498576387}_{\textbf{n}\ \text{digits}}
\end{array}
$$

How many **single digit operations** are performed?

Multiplication: Efficiency of the Algorithm

$$\begin{array}{r} 45385327687532875567463874566 \\ \times\ \underbrace{87734659827649875623498576387}_{\textbf{n}\ \text{digits}} \end{array}$$

How many **single digit operations** are performed?

Computing **n** partial products (at most **n** single digit multiplications and **n** additions each) = **~2n^2 ops**

Summing up all partial products = **~2n^2 ops**

Multiplication: Efficiency of the Algorithm

$$\underbrace{\begin{array}{rr} & 45385327687532875567463874566 \\ \times & 87734659827649875623498576387 \end{array}}_{\mathbf{n}\text{ digits}}$$

How many **single digit operations** are performed?

~4n² ops!

Computing **n** partial products (at most **n** single digit multiplications and **n** additions each) = **~2n² ops**

Summing up all partial products = **~2n² ops**

Can We Do Better?

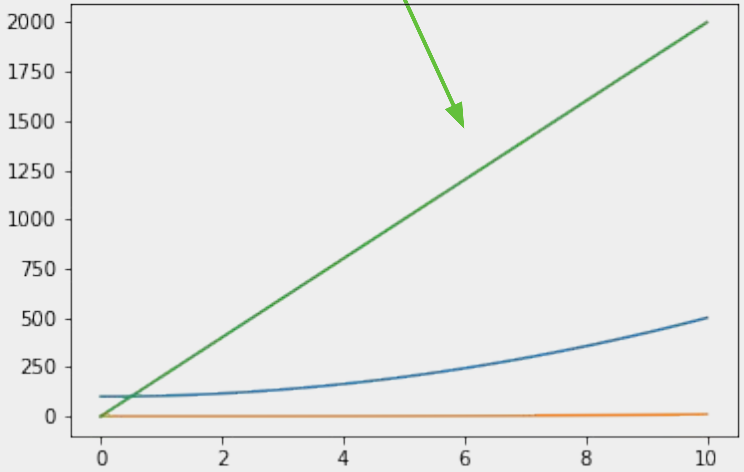
Which one looks better to you?

- $4n^2 + 100$
- $0.01n^3$
- $200n$

Can We Do Better?

Which one looks better to you?

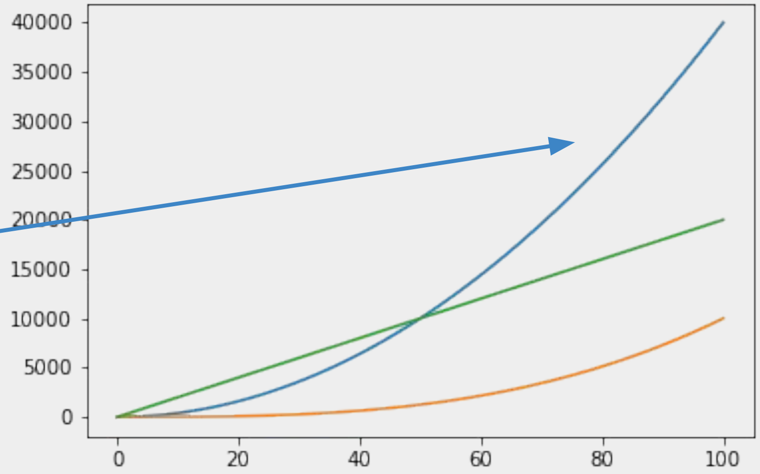
- $4n^2 + 100$
- $0.01n^3$
- $200n$

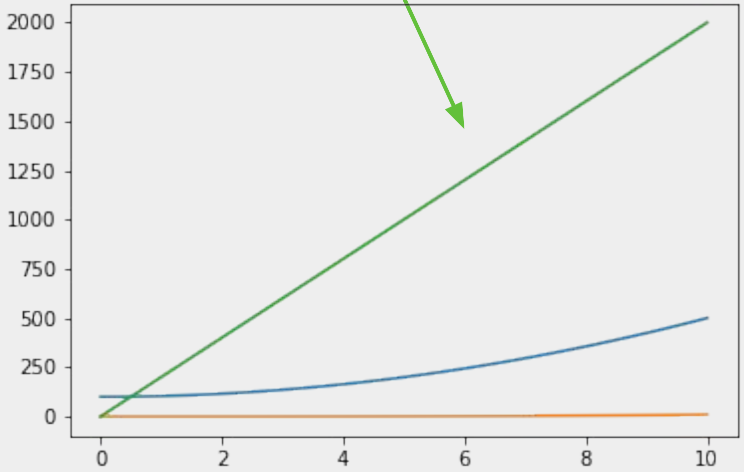


Can We Do Better?

Which one looks better to you?

- $4n^2 + 100$
- $0.01n^3$
- $200n$

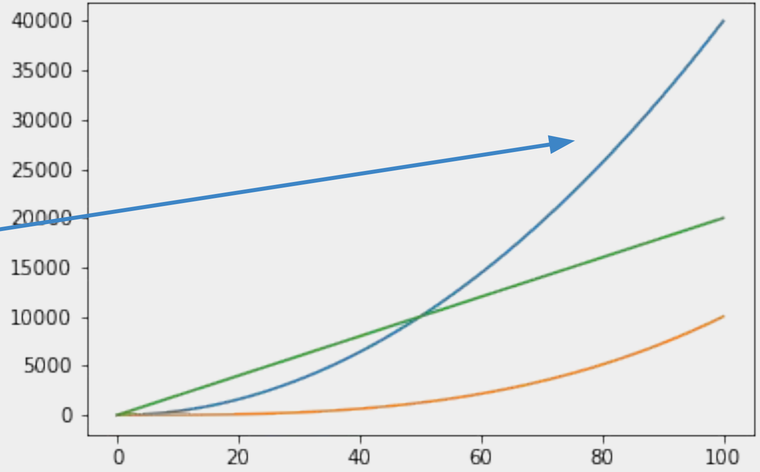


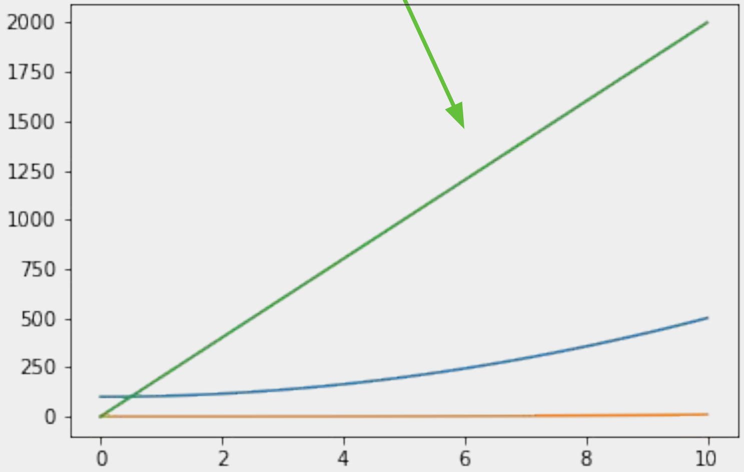


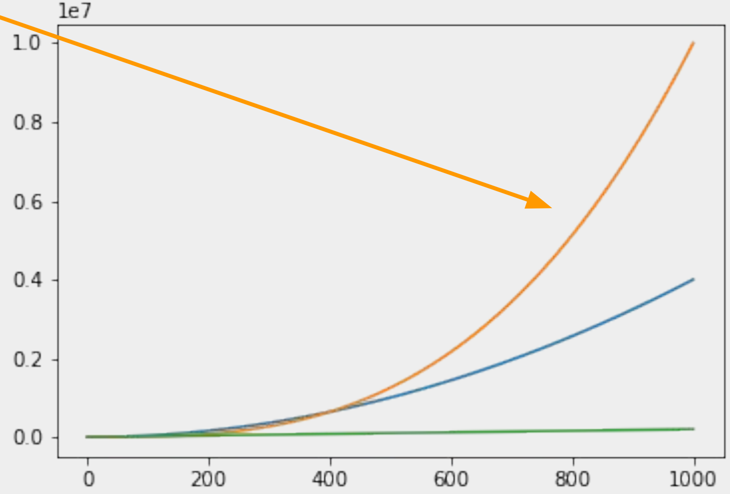
Can We Do Better?

Which one looks better to you?

- $4n^2 + 100$
- $0.01n^3$
- $200n$







Asymptotic Analysis

Asymptotic Analysis

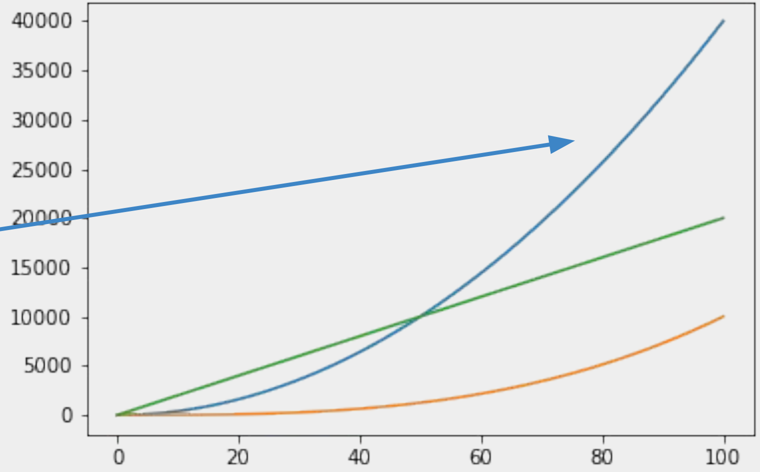
We want to measure how algorithms performance/running time/number of operations grows with the growth of the input size. And we want that measure to be **independent** of hardware, programming language, cpu optimizations, etc.

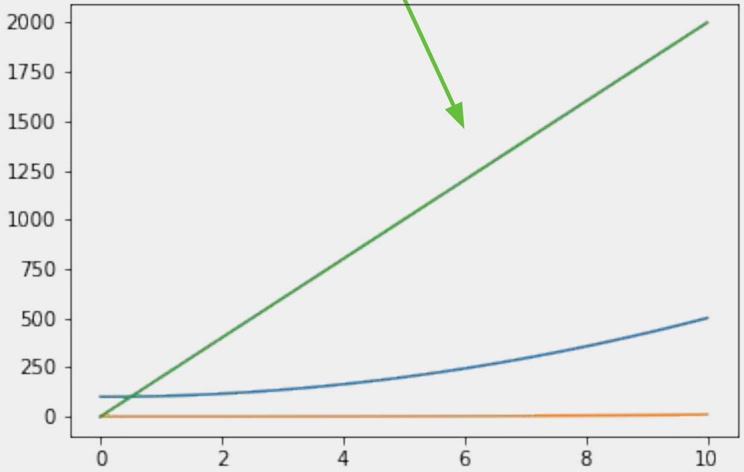
Asymptotic Analysis: Big-O Notation

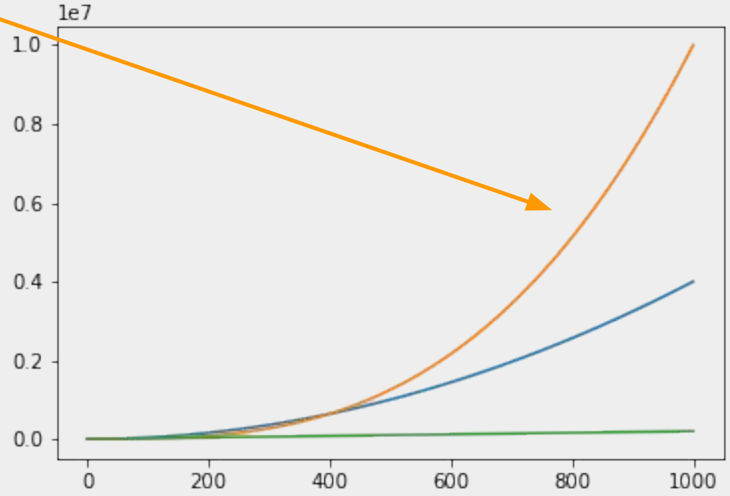
- We'll use $O(\cdot)$ notation.
  - We'll define this $O(\cdot)$ mathematically in the following lectures.
  - We'll say that elementary school multiplication algorithm runs in $O(n^2)$ time.
  - Informally if function is $O(n^2)$ it means it "grows like" n^2.
  - It ignores **constant factors** and **lower order terms**.

Can We Do Better?

- $4n^2 + 100$
- $0.01n^3$
- $200n$



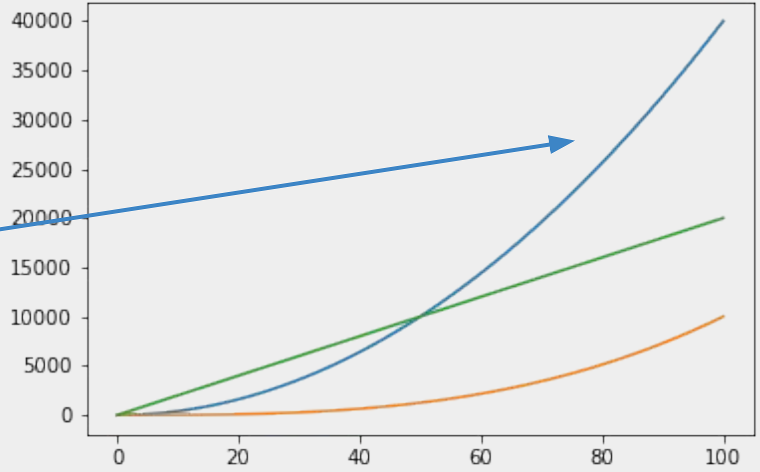


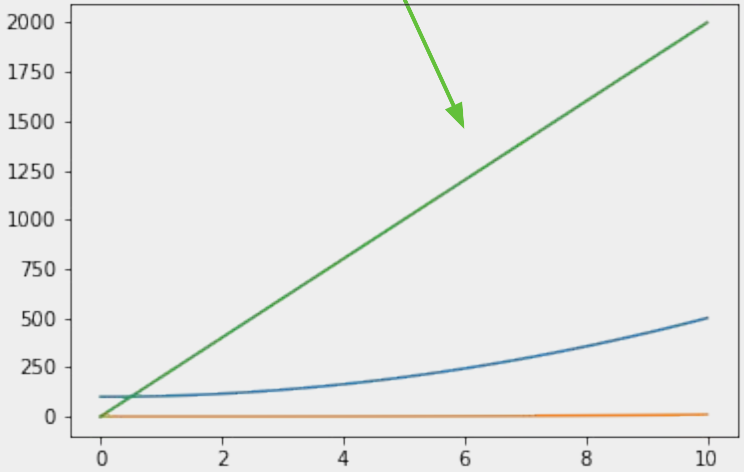


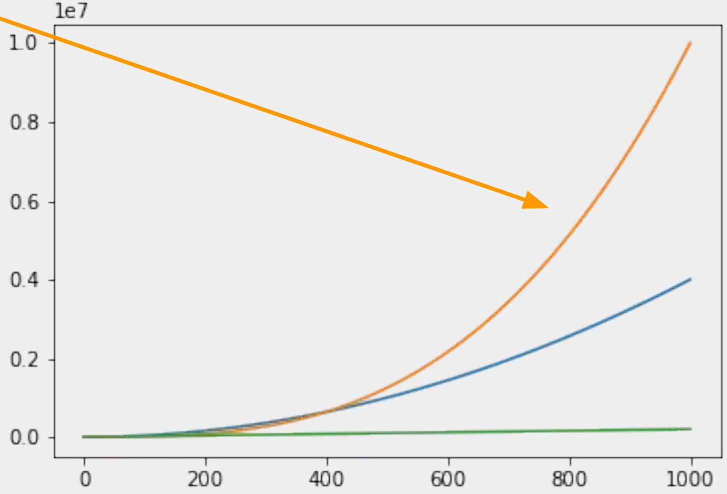
Can We Do Better?

$O(n^2)$
$O(n^3)$
$O(n)$

- $4n^2 + 100$
- $0.01n^3$
- $200n$







So Can We Multiply **Asymptotically** Faster?

Divide and Conquer!

Divide and Conquer

Our first algorithm design paradigm. The main idea:

1. Break up the problem into several similar smaller subproblems
2. Solve them recursively
3. Combine the results

Divide and Conquer: Multiplication Example

Original problem: multiply two 4 digit numbers

Subproblems:

$$1234 * 5678 =$$

$$(12 * 100 + 34) * (56 * 100 + 78) =$$

$$10000 * 12 * 56 + 100 * (12 * 78 + 34 * 56) + 34 * 78$$

Divide and Conquer: Multiplication Example

Original problem: multiply two 4 digit numbers

Subproblems:

$$1234 * 5678 =$$

$$(12 * 100 + 34) * (56 * 100 + 78) =$$

$$10000 * 12 * 56 + 100 * (12 * 78 + 34 * 56) + 34 * 78$$

Divide and Conquer: Multiplication Example

Original problem: multiply two 4 digit numbers

Subproblems:

$$[x_1, x_2, ...x_n] * [y_1, y_2, ...y_n] =$$

$$(a * 10^{n/2} + b) * (c * 10^{n/2} + d) =$$

$$10^n * a * c + 10^{n/2} * (a * d + b * c) + b * d$$

where:

$a = [x_1, x_2, ... x_{n/2}]$ $c = [y_1, y_2, ... y_{n/2}]$

$b = [x_{n/2+1}, x_{n/2+2}, ... x_n]$ $d = [y_{n/2+1}, y_{n/2+2}, ... x_n]$

Divide and Conquer: Multiplication Example

Original problem: multiply two 4 digit numbers

Subproblems:

$$[x_1, x_2, \ldots x_n] * [y_1, y_2, \ldots y_n] =$$

$$(a * 10^{n/2} + b) * (c * 10^{n/2} + d) =$$

$$10^n * a * c + 10^{n/2} * (a * d + b * c) + b * d$$

So we have **four n/2 digit** problems instead of **one n digit** problem.

where:

$a = [x_1, x_2, \ldots x_{n/2}]$ $\quad$ $c = [y_1, y_2, \ldots y_{n/2}]$

$b = [x_{n/2+1}, x_{n/2+2}, \ldots x_n]$ $\quad$ $d = [y_{n/2+1}, y_{n/2+2}, \ldots x_n]$

Divide and Conquer: Multiplication Pseudocode

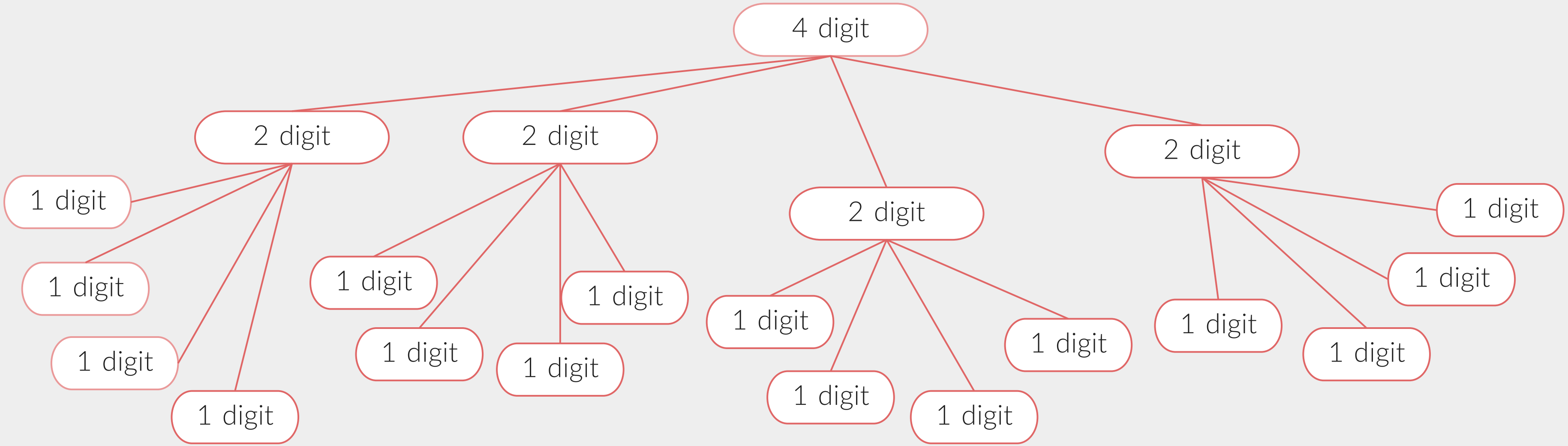
```
multiply(x, y):
    n = length of x
    if n == 1:
        return x * y
    a, b = split x
    c, d = split y
    ad = multiply(a, d)
    ac = multiply(a, c)
    bc = multiply(b, c)
    bd = multiply(b, d)

    return  10^n * ac + 10^(n/2) * (ad + bc) + bd
```

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

*For simplicity reasons here we assume that the length n is a power of 2.

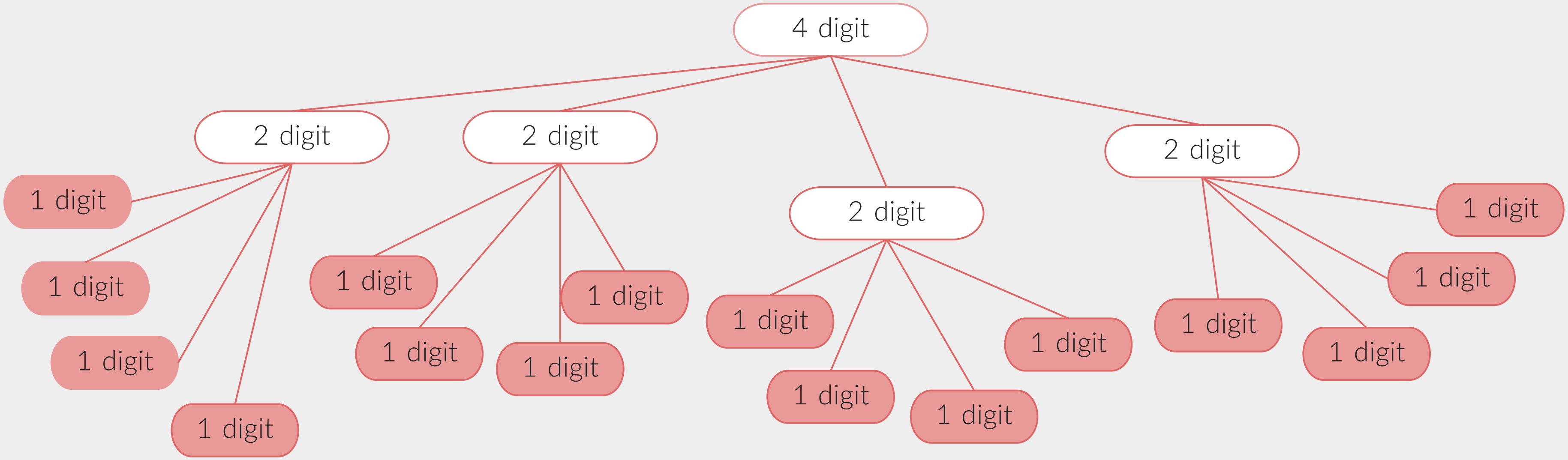
Divide and Conquer: Analysis

- How many single digit multiplications does this algorithm perform?
  - Recursion tree! (first for two 4-digit numbers)



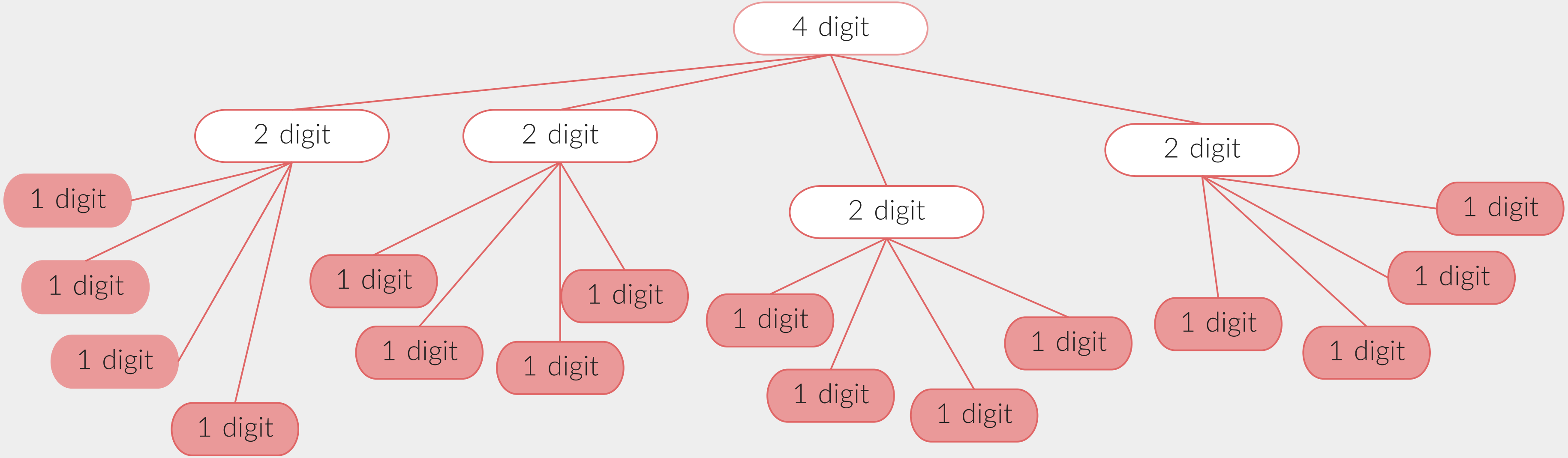
Divide and Conquer: Analysis

- How many single digit multiplications does this algorithm perform?
  - Recursion tree! (first for two 4-digit numbers)



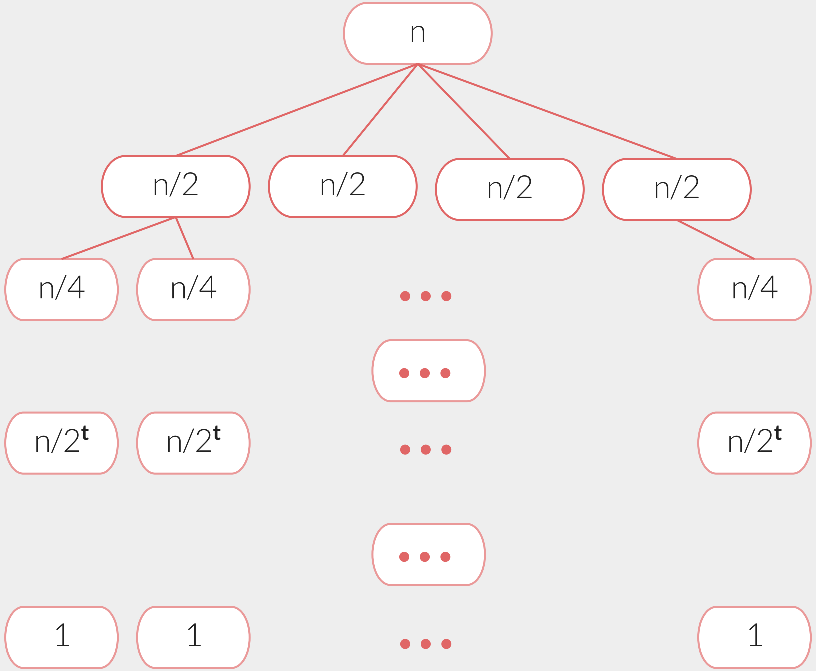
Divide and Conquer: Analysis

- How many single digit multiplications does this algorithm perform?
  - Recursion tree! (first for two 4-digit numbers)
  - 16!



Divide and Conquer: Analysis

- Now let's try to generalize, draw the recursion tree for **n** digit numbers.



level 0: 1 problem of size n

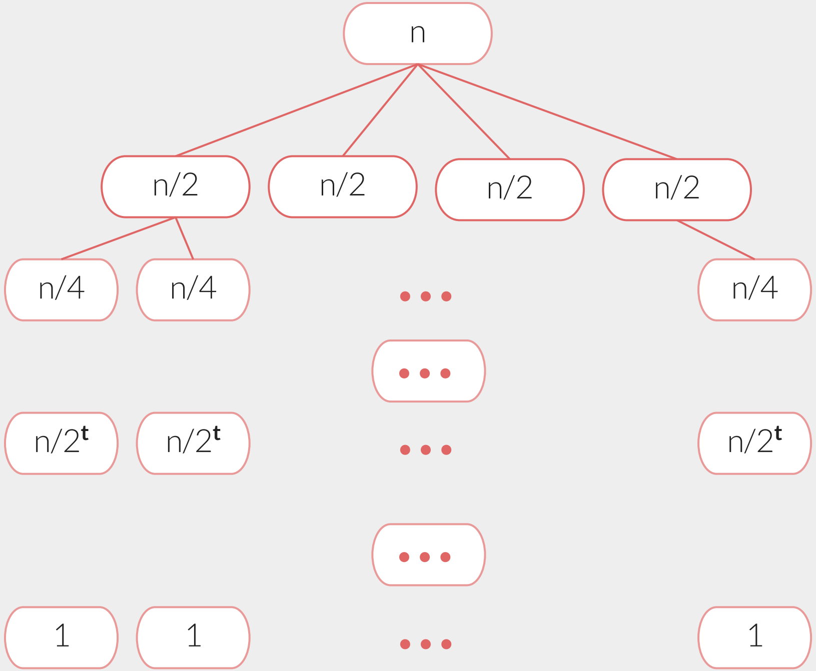
level 1: 4 problems of size n/2

level t: __ problem of size $n/2^t$

level ___ : __ problems of size 1

Divide and Conquer: Analysis

- Now let's try to generalize, draw the recursion tree for **n** digit numbers.



level 0: 1 problem of size n

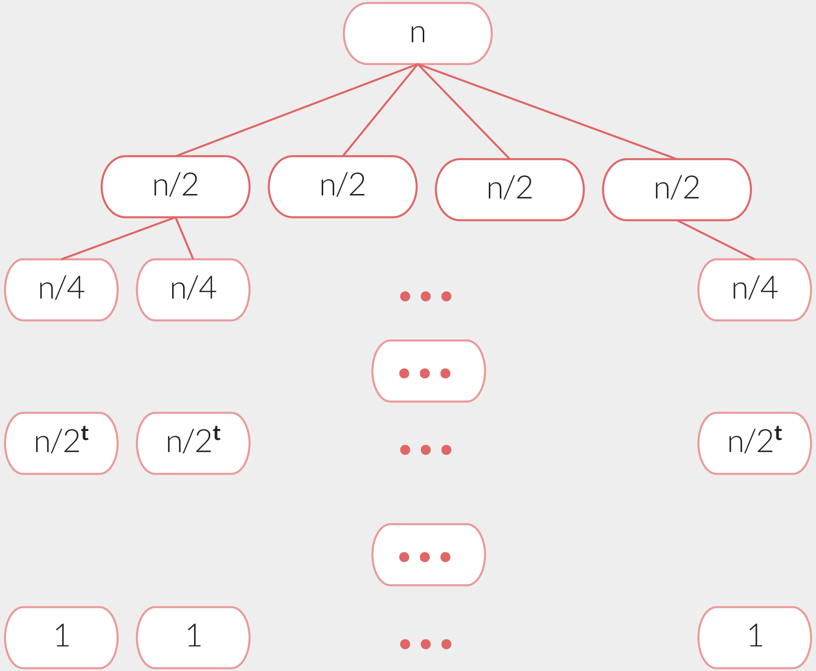
level 1: 4 problems of size n/2

level t: 4^t problem of size $n/2^t$

level ___ : __ problems of size 1

Divide and Conquer: Analysis

- Now let's try to generalize, draw the recursion tree for **n** digit numbers.



level 0: 1 problem of size n

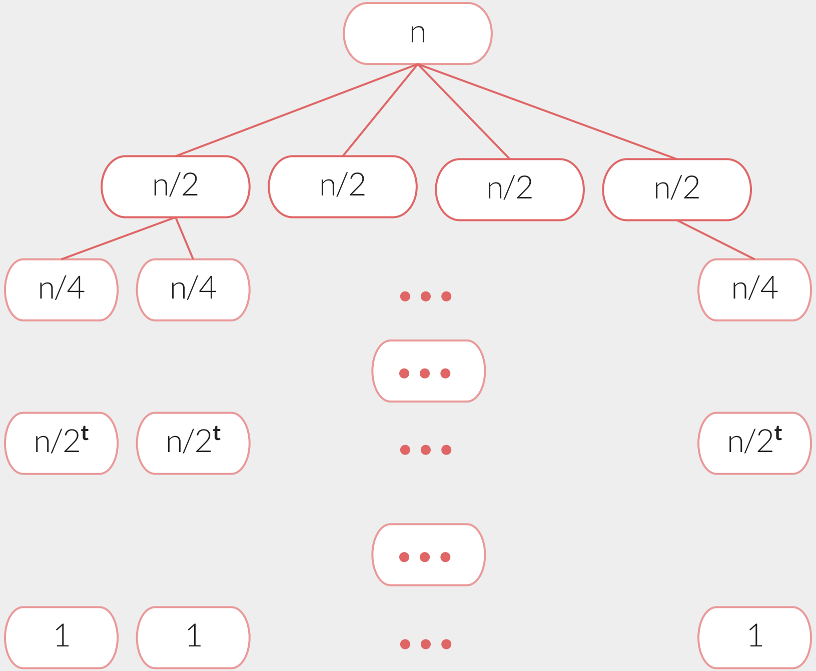
level 1: 4 problems of size n/2

level t: 4^t problem of size $n/2^t$

level log(n): __ problems of size 1

Divide and Conquer: Analysis

- Now let's try to generalize, draw the recursion tree for **n** digit numbers.



level 0: 1 problem of size n

level 1: 4 problems of size n/2

level t: 4^t problem of size $n/2^t$

level log(n): n^2 problems of size 1

Divide and Conquer: Analysis

- So divide and conquer does at least $O(n^2)$ operations, like our elementary-school algorithm did.
- What do we do?

Divide and Conquer: Analysis

- So divide and conquer does at least $O(n^2)$ operations, like our elementary-school algorithm did.
- What do we do?
  - Karatsuba algorithm!!

* photo from his wikipedia article

Divide and Conquer: Karatsuba Trick

Original problem: multiply two 4 digit numbers

Subproblems:

$$[x_1, x_2, ...x_n] * [y_1, y_2, ...y_n] =$$

$$(a * 10^{n/2} + b) * (c * 10^{n/2} + d) =$$

$$10^n * a * c + 10^{n/2} * (a * d + b * c) + b * d$$

Divide and Conquer: Karatsuba Trick

Original problem: multiply two 4 digit numbers

Subproblems:

$$[x_1, x_2, ...x_n] * [y_1, y_2, ...y_n] =$$

$$(a * 10^{n/2} + b) * (c * 10^{n/2} + d) =$$

$$10^n * a * c + 10^{n/2} * (a * d + b * c) + b * d$$

Here we divide ***(a* d + b * c)*** into two subproblems, but we don't actually need ***a * d*** and ***b * c*** separately.

What we can note: ***(a * d + b * c) = (a + b) * (c + d) - a * c - b * d***

As we have ***a * c*** and ***b * d*** computed, we only need ***(a + b) * (c + d)*!**

Divide and Conquer: Karatsuba Trick

So instead of computing these:

ac

ad

bc

bd

It's enough to compute these:

ac	**1**
bd	**2**
(a+b)*(c+d)	**3**

Divide and Conquer: Karatsuba Trick

So instead of computing these:

ac

ad

bc

bd

It's enough to compute these:

ac	**1**
bd	**2**
(a+b)*(c+d)	**3**

$$10^n * \underbrace{a * c}_{1} + 10^{n/2} * \underbrace{(a * d + b * c)}_{3\,-\,2\,-\,1} + \underbrace{b * d}_{2}$$

Divide and Conquer: Karatsuba Trick

So instead of computing these:

ac

ad

bc

bd

* important note (a+b) and (c + d) still have n/2 digits, so it's still a half-sized problem.

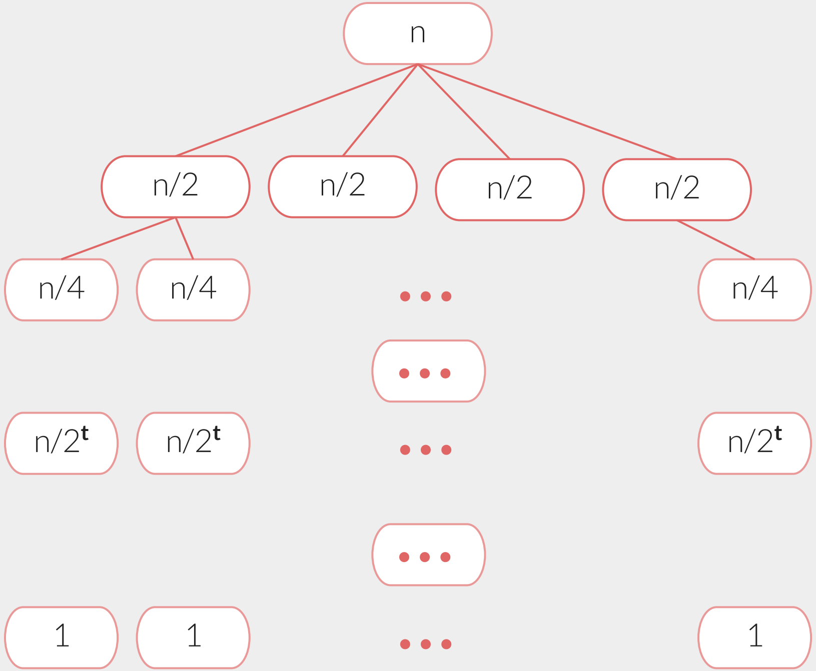
It's enough to compute these:

ac	**1**
bd	**2**
(a+b)*(c+d)	**3**

$$10^n * \underbrace{a * c}_{1} + 10^{n/2} * (\underbrace{a * d + b * c}_{3\,-\,2\,-\,1}) + \underbrace{b * d}_{2}$$

Recursion Tree: First Attempt

This is recursion tree for our first attempt.
For Karatsuba algorithm we will **cut the branching factor from 4 to 3!**



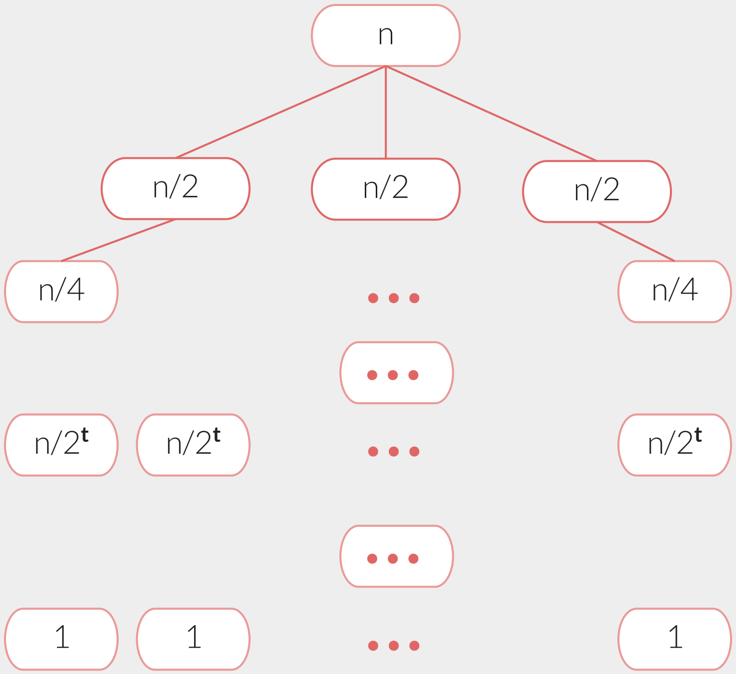
level 0: 1 problem of size n

level 1: 4 problems of size n/2

level t: 4^t problem of size $n/2^t$

level log(n): n^2 problems of size 1

Recursion Tree: Karatsuba Multiplication



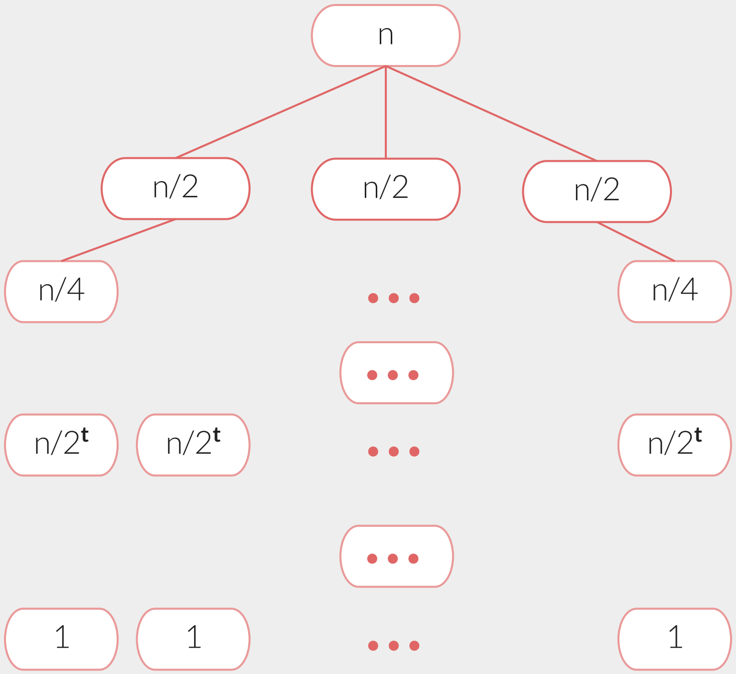
level 0: 1 problem of size n

level 1: 3 problems of size n/2

level t: 3^t problem of size $n/2^t$

level log(n): ____ problems of size 1

Recursion Tree: Karatsuba Multiplication



$$3^{\log(n)} = n^{\log(3)} = n^{1.58496....}$$

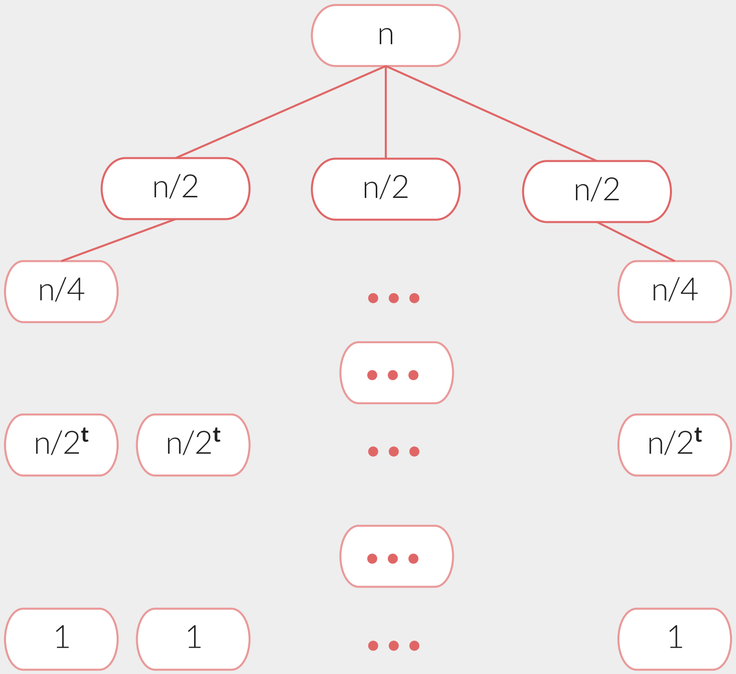
level 0: 1 problem of size n

level 1: 3 problems of size n/2

level t: 3^t problem of size $n/2^t$

level log(n): ____ problems of size 1

Recursion Tree: Karatsuba Multiplication



$3^{\log(n)} = n^{\log(3)} = n^{1.58496....}$

level 0: 1 problem of size n

level 1: 3 problems of size n/2

level t: 3^t problem of size $n/2^t$

level log(n): $n^{\sim 1.6}$ problems of size 1

Recap

- You'll learn how to **analyze, design** and **talk about** algorithms.
- We looked at some Divide and Conquer.
- Karatsuba Algorithm.
- Analyzing algorithm runtimes **asymptotically**.