

Asymptotic Analysis

Why Analyze Running Time?

- Suppose you've created a new algorithm
- How do you convince people to use it?
- Does it...
 - Run faster?
 - Use fewer resources?

Why Analyze Running Time?

- Suppose I've invented a new sorting algorithm, **AsifSort**
- How do I show that it beats existing methods?

Why Analyze Running Time?

- Suppose I've invented a new sorting algorithm, **AsifSort**
How do I show that it beats existing methods?
- Easiest way:
 - Make up a bunch of test cases
 - Run AsifSort and other sort algorithms on these test cases
 - Which finishes first?



Why not?

Why Analyze Running Time?

- Suppose I've invented a new sorting algorithm, **AsifSort**
 - How do I show that it beats existing methods?
 - Easiest way:
 - Make up a bunch of test cases
 - Run AsifSort and other sort algorithms on those test cases
 - Which finishes first?
- How do you know they're the 'right' test cases?
- Computers get faster; data gets bigger... will the ordering change next year?

Why Analyze Running Time?

- Data sizes are doubling every year!
- Moore's Law: density of transistors doubles every 18-24 months: more processing speed!
- If your program takes 2 minutes to run today...how long will it take next year?

Key Observations

- Run-time analysis should be:
 - Independent of the **platform**
 - Independent of the **programmer's skill**
 - Independent of **specific test cases** (content and size!)
 - **Theoretically rigorous**

So How Can We Do Better?

- **Theoretical analysis of algorithms** is used to estimate the run-time of an algorithm as a function of the size of the input
- This run-time is given in terms of the number of **primitive operations** required (e.g., arithmetic operations)

$T(n)$

$m(n)$

Let's do an example

n

```
function sumArray(arr):
```

```
    t_product = 0
```

```
    t_product_product = 0
```

```
    for idx in range(len(arr)):
```

```
        t_product *= arr[idx]
```

```
    for idx1 in range(len(arr)):
```

```
        for idx2 in range(len(arr)):
```

```
            t_product_product += arr[idx1] * arr[idx2]
```

```
    return t_product, t_product_product
```

$$2 + 2n + 3n^2$$

Let's do an example

```
function sumArray(arr):  
    t_product = 0  
    t_product_product = 0  
  
    for idx in range(len(arr)):  
        t_product *= arr[idx]  
  
    for idx1 in range(len(arr)):  
        for idx2 in range(len(arr)):  
            t_product_product += arr[idx1] * arr[idx2]  
  
    return t_product, t_product_product
```

$$T(n) = 1 + 1 + n + 3 \times n^2$$

$$T_1(n) = 9 \times n^4 + 68n + 323$$

$$T_2(n) = 2^n + 3$$

n	$T_1(n)$	$T_2(n)$
1	400	5
2	603	7
3	1256	11
4	2899	19
5	6288	35
6	12395	67
7	22408	131
8	37731	259
9	59984	515
10	91003	1027
11	132840	2051
12	187763	4099
13	258256	8195
14	347019	16387
15	456968	32771
16	591235	65539
17	753168	131075
18	946331	262147
19	1174504	524291
20	1441683	1048579
21	1752080	2097155
22	2110123	4194307
23	2520456	8388611
24	2987939	16777219
25	3517648	33554435
26	4114875	67108867

Classes of functions

$$1 < \log n < \sqrt{n} < n < n \log n < n^2 < n^3 < n^4 < \dots < 2^n < 3^n < \dots < n^n$$

For all $n > n_0$

Classes of functions

n	logn	n×logn	n^2	n^3	n^4	2^n
1	0	0	1	1	1	2
2	1	2	4	8	16	4
3	1.5849625	4.7548875	9	27	81	8
4	2	8	16	64	256	16
5	2.32192809	11.6096405	25	125	625	32
6	2.5849625	15.509775	36	216	1296	64
7	2.80735492	19.6514845	49	343	2401	128
8	3	24	64	512	4096	256
9	3.169925	28.529325	81	729	6561	512
10	3.32192809	33.2192809	100	1000	10000	1024
11	3.45943162	38.0537478	121	1331	14641	2048
12	3.5849625	43.01955	144	1728	20736	4096
13	3.70043972	48.1057163	169	2197	28561	8192
14	3.80735492	53.3029689	196	2744	38416	16384
15	3.9068906	58.6033589	225	3375	50625	32768
16	4	64	256	4096	65536	65536
17	4.08746284	69.4868683	289	4913	83521	131072
18	4.169925	75.05865	324	5832	104976	262144
19	4.24792751	80.7106228	361	6859	130321	524288
20	4.32192809	86.4385619	400	8000	160000	1048576
21	4.39231742	92.2386659	441	9261	194481	2097152
22	4.45943162	98.1074956	484	10648	234256	4194304
23	4.52356196	104.041925	529	12167	279841	8388608
24	4.5849625	110.0391	576	13824	331776	16777216
25	4.64385619	116.096405	625	15625	390625	33554432

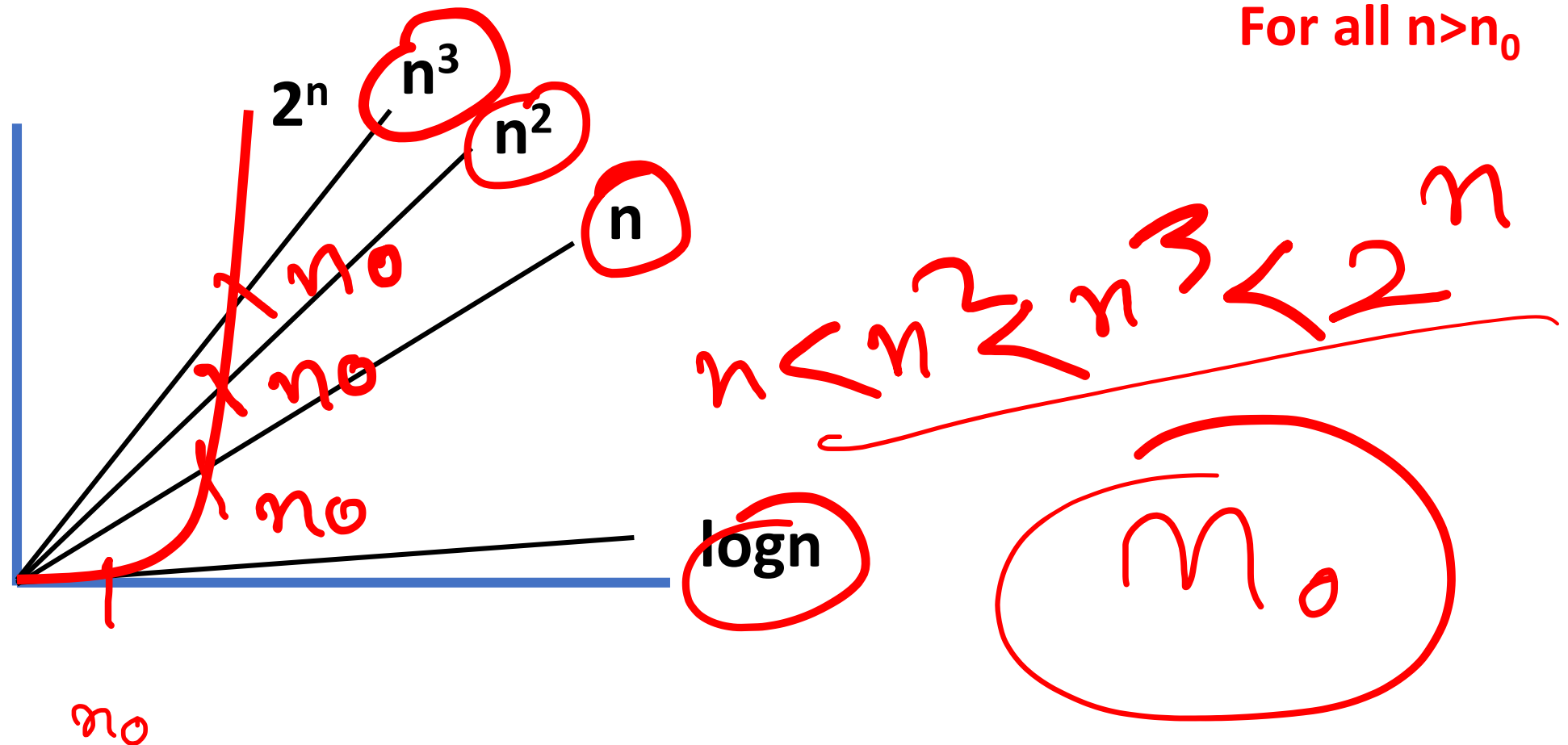
$n \leq 16$
 $2^n \geq n^4$

$n^4 < 2^n$

$n^k < 2^n$
 n^0

Classes of functions

$$1 < \log n < \sqrt{n} < n < n \log n < n^2 < n^3 < n^4 < \dots < 2^n < 3^n < \dots < n^n$$



Asymptotic notation

$O(g)$ at most within const of g for large n Upper Bound

$\Omega(g)$ at least within const of g for large n Lower Bound

$\Theta(g)$ within const of g for large n Average Bound

$1 < \log n < \sqrt{n} < n < n \log n < n^2 < n^3 < n^4 < \dots < 2^n < 3^n < \dots < n^n$

Asymptotic notation

$T(n)$
 $f(n)$

Set of functions

$f(n) = O(g(n))$ at most within cost of g for large n

$\left\{ \begin{array}{l} f(n) = O(g(n)) \text{ if there exist positive constants } c, n_0 \text{ such that} \\ \text{for all } n > n_0, 0 \leq f(n) \leq c \times g(n) \end{array} \right\}$

Important note:

$O(g(n))$ is actually a set!

When we say " $f(n) = O(g(n))$ ", we are actually
Saying that $f(n) \in O(g(n))$

Big-O Notation: Definition

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

- There exists:
 - some positive constant c
 - some minimum n -value n_0
- Such that for all $n > n_0$:
 - $f(n) \leq c \cdot g(n)$

**$f(n)=O(g(n))$ if there exist positive constants c, n_0 such that
for all $n>n_0, 0 \leq f(n) \leq c \times g(n)$**

Big-O Notation: Definition



$f(n)=O(g(n))$ if there exist positive constants c, n_0 such that for all $n>n_0$, $0 \leq f(n) \leq c \times g(n)$

$$f(n)=2n+3$$


$$f(n)=2n+3 \leq 10n \text{ for all } n \geq 1$$

$f(n)$ c $g(n)$ n_0


$$g(n)=n$$

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

Big-O Notation: Definition

 $f(n)=O(g(n))$ if there exist positive constants c, n_0 such that
for all $n>n_0$, $0 \leq f(n) \leq c \times g(n)$

$$f(n)=2n+3$$



$$f(n)=2n+3 \leq 2n + 3n$$

$$f(n)=2n+3 \leq 5n$$

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

for all $n \geq 1$

Big-O Notation: Definition

$1 < \log n < \sqrt{n} < \mathbf{n} < n \log n < n^2 < n^3 < n^4 < \dots < 2^n < 3^n < \dots < n^n$

$f(n) = O(g(n))$ if there exist positive constants c, n_0 such that
for all $n > n_0$, $0 \leq f(n) \leq c \times g(n)$

$$f(n) = 2n + 3$$

$$f(n) = 2n + 3 \leq 2n^2 + 3n^2 \quad \text{for all } n \geq 1$$

$$f(n) = 2n + 3 \leq 5n^2$$

$$f(n) = O(n^2)$$

- Not a tight upper bound
- In algorithm analysis We want tight bounds

Big-O Notation: Notation Conventions

- $O(g(n))$ represents a set, so the $=$ sign doesn't mean what it normally means
- We use " $=$ " to represent **set membership**
- This means that "equality" is **not symmetric!**

We can say $f(n) = O(g(n))$, but not $O(g(n)) = f(n)$!

Let's do an example

```
function sumArray(arr):
```

```
    t_product = 0
```

```
    t_product_product = 0
```

```
    for idx in range(len(arr)):
```

```
        t_product *= arr[idx]
```

```
    for idx1 in range(len(arr)):
```

```
        for idx2 in range(len(arr)):
```

```
            t_product_product += arr[idx1] * arr[idx2]
```

```
    return t_product, t_product_product
```

$T(n) = 1 + 1 + n + 3 \times n^2$

$T(n) = O(n^2)$

? $= O(n^2)$

$$1 + 1 + n + 3n^2$$

$$2 + n + 3n^2 \leq 2n^2 + n^2 + 3n^2$$

$$2 + n + 3n^2 \leq 6n^2$$

Omega Notation

**$f(n) = \Omega(g(n))$ if there exist positive constants c, n_0 such that
for all $n > n_0$, $c \times g(n) \leq f(n)$**

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

- There exists:
 - some positive constant c
 - some minimum n -value n_0
- Such that for all $n > n_0$:
 - $f(n) \geq c * g(n)$

Omega Notation

$f(n) = \Omega(g(n))$ if there exist positive constants c, n_0 such that
for all $n > n_0$, $c \times g(n) \leq f(n)$

$$f(n) = 2n + 3$$

$$f(n) = 2n + 3 \geq 1 \times n \quad \text{for all } n \geq 1$$

$f(n)$

c

$g(n)$

$= n$

n_0

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

Omega Notation

$f(n) = \Omega(g(n))$ if there exist positive constants c, n_0 such that
for all $n > n_0$, $c \times g(n) \leq f(n)$

①

$$f(n) = 2n + 3$$

$$f(n) = 2n + 3 \geq \log n$$

for all $n \geq 1$

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

$$f(n) = \Omega(1)$$
$$f(n) = \Omega(\sqrt{n})$$

$$1 < \log n < \sqrt{n} < n < n \log n < n^2 < n^3 < n^4 < \dots < 2^n < 3^n < \dots < n^n$$

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

Theta notation

$\Theta(g)$ *For all of our algorithms we want to proof the theta bound*

$f(n)=\Theta(g(n))$ if there exist positive constants c_1 , c_2 and n_0

such that $0 \leq c_1 \times g(n) \leq f(n) \leq c_2 \times g(n)$ for all $n \geq n_0$

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

- There exists:
 - some positive constants c_1 and c_2
 - some minimum n -value n_0 *same*
- Such that for all $n > n_0$:
 - $0 \leq c_1 \times g(n) \leq f(n) \leq c_2 \times g(n)$

Theta notation

$f(n) = \Theta(g(n))$ if there exist positive constants c_1 , c_2 and n_0 such that
 $0 \leq c_1 \times g(n) \leq f(n) \leq c_2 \times g(n)$ for all $n \geq n_0$

$$f(n) = 2n + 3$$

$$f(n) = 2n + 3 \leq 2n^2 + 3n^2 \quad \text{for all } n \geq 1$$

$$f(n) = 2n + 3 \leq 5n$$

$$1 + 1 \times n \leq f(n) = 2n + 3 \leq 5 \times n$$

$c_1 g(n)$

$f(n)$

c_2

$g(n)$

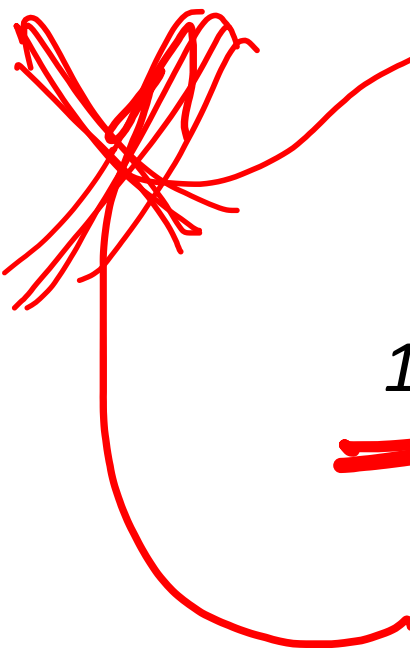
Can we put n^2 here?

$$g(n) = n$$

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

Theta notation

$f(n)=\Theta(g(n))$ if there exist positive constants c_1, c_2 and n_0 such that
 $0 \leq c_1 \times g(n) \leq f(n) \leq c_2 \times g(n)$ for all $n \geq n_0$



$f(n)=2n+3$

$f(n)=2n+3 \leq 2n+3n^2$ for all $n \geq 1$

$f(n)=2n+3 \leq 5n^2$

$1 + 1 \times n \leq f(n)=2n+3 \leq 5 \times n^2$ ← Can we put n^2 here?

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

$f(n) \neq \Theta(n^2)$

$$1 < \log n < \sqrt{n} < n < n \log n < n^2 < n^3 < n^4 < \dots < 2^n < 3^n < \dots < n^n$$

$$f(n) = 2n^2 + 3n + 4$$

$$2n^2 + 3n + 4 < 2n^2 + 3n^2 + 4n^2$$

$$2n^2 + 3n + 4 < 9n^2$$

$$1 \times n^2 < 2n^2 + 3n + 4$$

$$n_0 = 1 \quad C = 1$$

$$f(n) = O(n^2)$$

$$f(n) = \Omega(n^2)$$

$$f(n) = \Theta(n^2)$$

n	$2n^2 + 3n + 4$	$9n^2$
1	9	9
2	18	36
3	31	81
4	48	144
5	69	225
6	94	324
7	123	441
8	156	576
9	193	729
10	234	900

$$1 < \log n < \sqrt{n} < n < n \log n < n^2 < n^3 < n^4 < \dots < 2^n < 3^n < \dots < n^n$$

$$f(n) = n^2 \log n + n$$

$$n^2 \log n + n < 10 \times n^2 \log n$$

$$1 \times n^2 \log n < n^2 \log n + n$$

$$f(n) \neq O(n^2 \log n)$$

$$f(n) = \Omega(n^2 \log n)$$

$$f(n) = \Theta(n^2 \log n)$$

~~$n \geq 1$~~
 $n \geq 2$

Big-O Notation: In Practice

- Use these **simplification rules**:
 - Only pay attention to the dominant terms (x^3 more important than x^2)
 - Don't include constants in your big-O expression

$$1 < \log n < \sqrt{n} < n < n \log n < n^2 < n^3 < n^4 < \dots < 2^n < 3^n < \dots < n^n$$

$$f(n) = n! = n \times (n-1) \times (n-2) \times \dots \times 3 \times 2 \times 1$$

$$1 \times 2 \times 3 \times \dots \times (n-2) \times (n-1) \times n$$

$$< n \times n \times n \times \dots \times n$$

$$< n^n$$

$$(1 \times 1 \times 1 \times \dots \times 1)$$

$$\Omega(1)$$

$$f(n) = O(n^n)$$

$$1 \leq n! \leq n^n$$

$$f(n) = \Omega(1)$$

$$g = 1$$

$$g = n^n$$

$$O(n^n)$$

$$1 < \log n < \sqrt{n} < n < n \log n < n^2 < n^3 < n^4 < \dots < 2^n < 3^n < \dots < n^n$$

$$f(n) = \log n!$$

$$\log(1 \times 2 \times 3 \times \dots \times n)$$

$$< \log(\underbrace{n \times n \times n \times \dots \times n}_n)$$

$$\log(1 \times 1 \times 1 \times \dots \times 1) \leftarrow$$

$$O(\log n^n)$$

$$O(n \log n)$$

$$1 \leq \log n! \leq \log n^n$$

~~$$1 \leq \log n! \leq n \log n$$~~

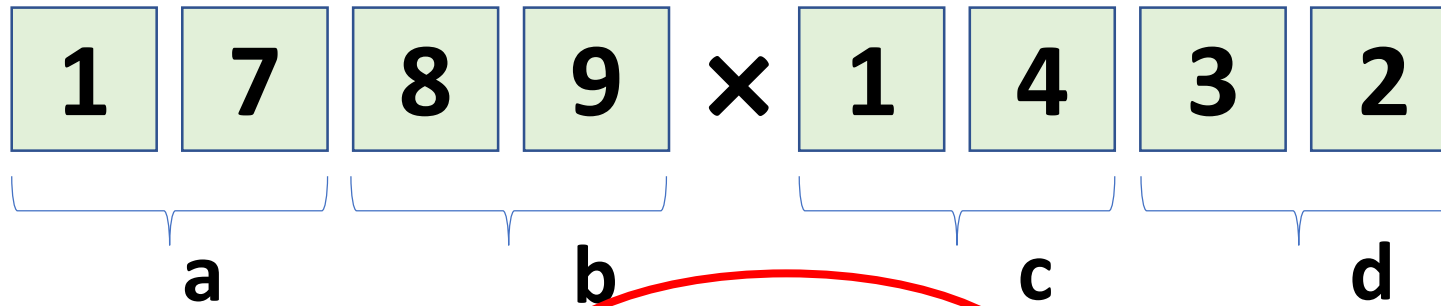
$$\Omega(\log(1))$$

$$f(n) = O(n \log n)$$

$$f(n) = \Omega(1)$$

Review

Karatsuba



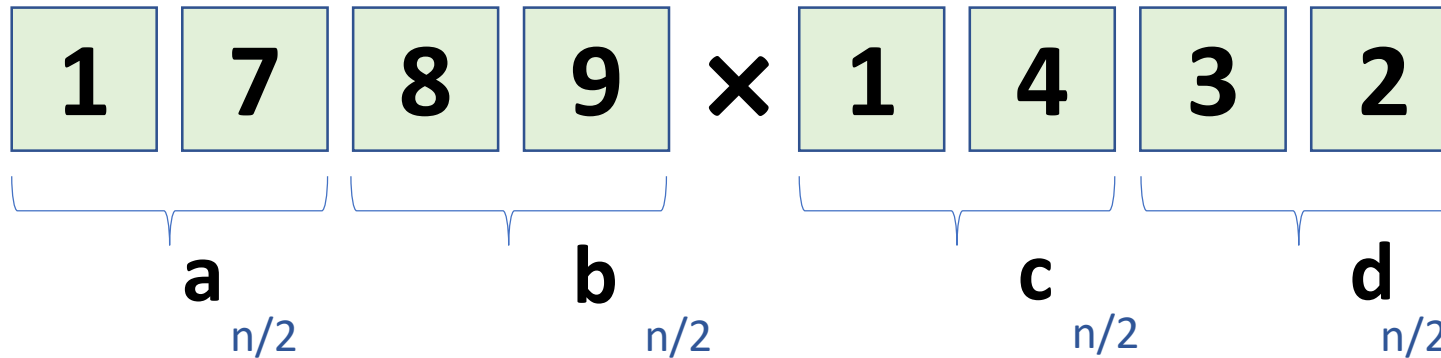
$$(a \times c)(100^2) + (a \times d + b \times c)(100) + b \times d$$

$$\Rightarrow (a+b)(c+d) = ac + ad + bc + bd$$

$$ad + bc = (a+b)(c+d) - ac - bd$$

Karatsuba

$$(a \times c)(100^2) + (a \times d + b \times c)(100) + b \times d$$



Recursively compute:

1. $ac, bd, (a+b)(c+d)$

2. $ad+bc = (a+b)(c+d) - ac - bd$

3. $ac \times 100^2 + (ad+bc) \times 100 + bd$

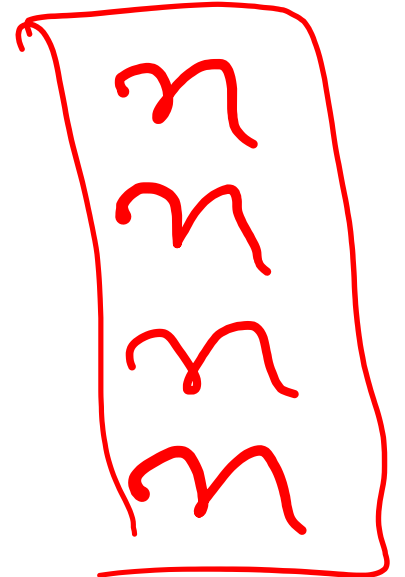
$3T\left(\frac{n}{2}\right)$

2 addition,

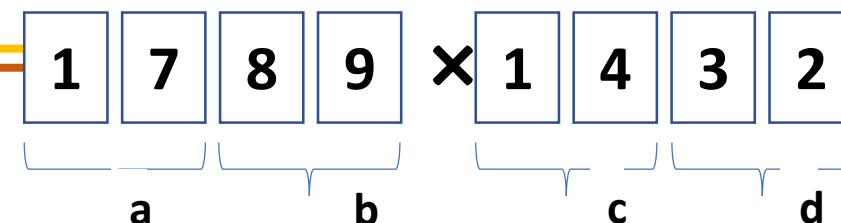
4n subtraction

4n addition

Approximately,
Not exactly



Karatsuba (ab,cd)



BASE CASE:

return $b \times d$ if inputs are 1 digit

ELSE:

Compute $ac = \text{karatsuba}(a, c)$ → $T(\frac{n}{2})$

Compute $bd = \text{karatsuba}(b, d)$ → $T(\frac{n}{2})$

Compute $t = \text{karatsuba}((a+b), (c+d))$ → $T(\frac{n}{2}) + 2n$

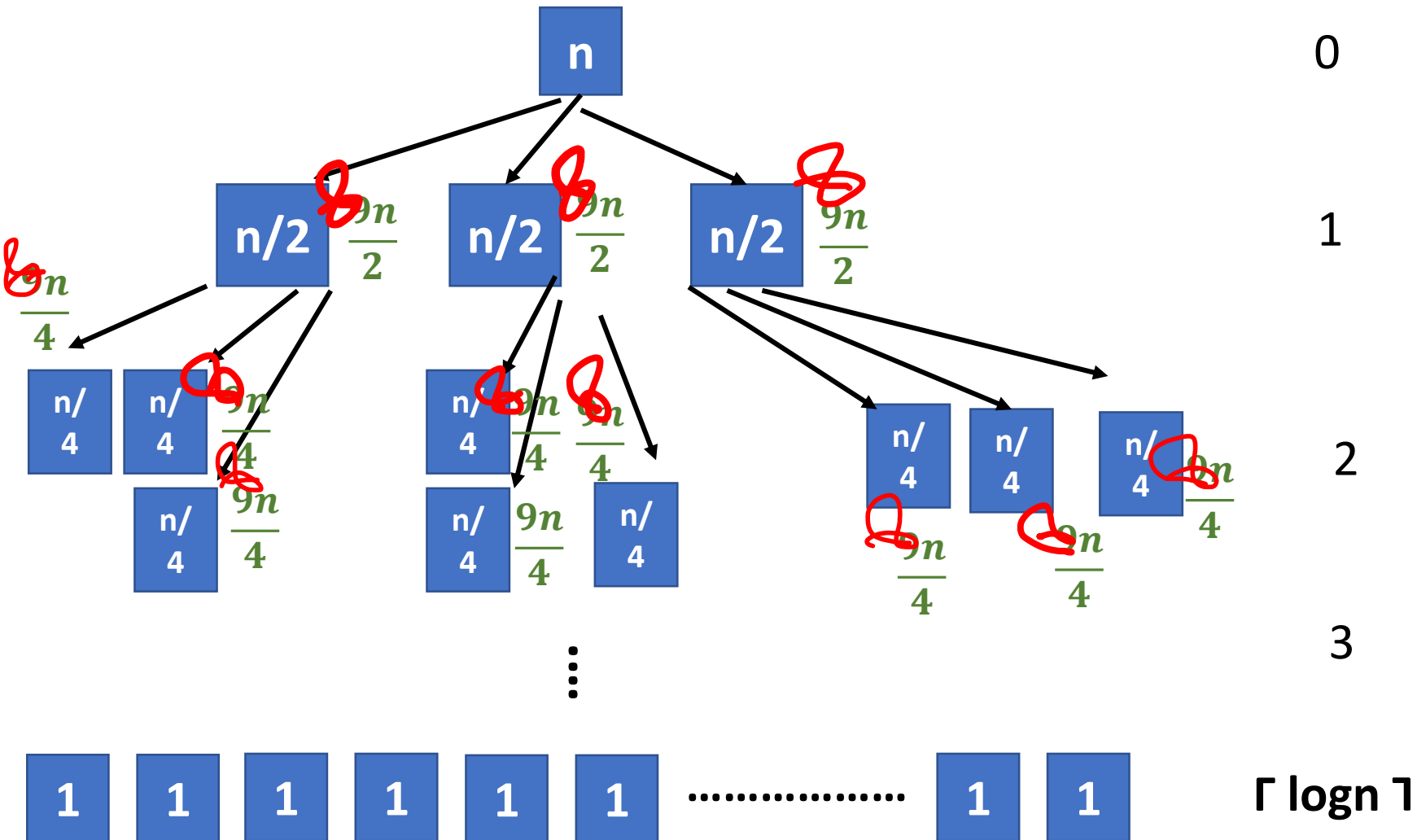
$\text{mid} = t - ac - bd$ → $3n$

$T(n) = 3T(\frac{n}{2}) + 9n$
Ignoring issue of carries

Return $ac \times (10)^{(\text{number of digit of } a)} + \text{mid} \times (10)^{(\text{number of digit of } a)} + bd$

4n steps

$$T(n) = 3T\left(\frac{n}{2}\right) + 9n$$



~~$9n$~~

$$3 \times \frac{9n}{2} = \left(\frac{3}{2}\right)^1 \times \del{9n}$$

$$9 \times \frac{9n}{4} = \left(\frac{3}{2}\right)^2 \times \del{9n}$$

$$27 \times \frac{9n}{8} = \left(\frac{3}{2}\right)^3 \times \del{9n}$$

$$\left(\frac{3}{2}\right)^{\Gamma \log n 1} \times \del{9n}$$

Calculations:

$$(1+a+a^2+\dots+a^L) = \frac{a^{L+1}-1}{a-1}$$

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

$$T(n) = 9n + \frac{3}{2} \times 9n + \left(\frac{3}{2}\right)^2 \times 9n + \dots + \left(\frac{3}{2}\right)^{\lceil \log n \rceil} \times 9n$$

$$= 9n \times \left(1 + \frac{3}{2} + \left(\frac{3}{2}\right)^2 + \dots + \left(\frac{3}{2}\right)^{\lceil \log n \rceil}\right)$$

$$= 9n \times \left[\frac{\left(\frac{3}{2}\right)^{\lceil \log n \rceil + 1} - 1}{\frac{3}{2} - 1} \right] = 9n \times (2) \times \left[\left(\frac{3}{2}\right)^{\lceil \log n \rceil + 1} - 1 \right]$$

$$= 2 \times 9n \left[2^{\log_2 \frac{3}{2}} \right]^{\log n + 1} - 18n = 18n \left[2^{(\log_2 3 - 1)} \right]^{\log n + 1} - 18n$$

$$= 18n \left[2^{((\log_2 n)^{\log_2 3} - \log_2 n + \log_2 3 - 1)} \right] - 18n = 18n \left[\frac{n^{\log_2 3} \times 2^{\log_2 3 - 1}}{n} \right] - 18n$$

$$= 18 \times 2^{(\log_2 3 - 1)} \times n^{\log_2 3} - 18n = O(n^{\log_2 3})$$