Announcements

- Homework 2 Solutions online
- No homework this week
- Exam 1 on Friday

Homework Note

- TA piazza post:
  - Proofread answers
  - Select all solution pages on gradescope (and try to avoid splitting pages)
  - Make sure that your solutions are legible
  - High level description of algorithm
  - Remember: ALL homeworks require justification unless stated otherwise

DFS/explore

Remember:

- Explore
  - Finds all vertices reachable from a single source
  - DOES NOT visit any other vertices
  - Does not compute pre/post orders for other vertices
- DFS
  - Visits ALL vertices of the graph
  - Cannot tell what is reachable from what without extra work
  - Can compute pre/post orders

Last Time

- Divide and Conquer
- Schoolboy Multiplication

Divide & Conquer (Ch 2)

- General Technique
- Master Theorem
- Karatsuba Multiplication
- Strassen’s Algorithm
- Merge Sort
- Order Statistics
- Binary Search
- Closest Pair of Points

Divide and Conquer

This is the first of our three major algorithmic techniques.

1. Break problem into pieces
2. Solve pieces recursively
3. Recombine pieces to get answer

Example: Integer Multiplication

Problem: Given two n-bit numbers find their product.

Naïve Algorithm: Schoolboy multiplication. The binary version of the technique that you probably learned in elementary school.

Runtime: $O(n^2)$

Schoolboy Multiplication

$$
\begin{array}{cccccccc}
 & & & a_1 & a_2 & \dots & a_{n-1} & a_n \\
 & & \text{x} & b_1 & b_2 & \dots & b_{n-1} & b_n \\
\cline{4-8}
 & & a_1b_n & a_2b_n & a_3b_n & \dots & a_{n-1}b_n & a_nb_n \\
 & a_1b_{n-1} & a_2b_{n-1} & a_3b_{n-1} & a_4b_{n-1} & \dots & a_nb_{n-1} & 0 \\
 & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\
+a_1b_n & a_2b_n & a_3b_n & \dots & a_nb_n & 0 & 0 & 0 \\
\hline
\end{array}
$$

ANSWER

Today

- Karatsuba multiplication
- Master Theorem
- Strassen’s algorithm

Two Digit Multiplication

```
    a       b
x   c       d
    -----------
```

Two Digit Multiplication

```
            a     b
        x   c     d
            -----------
            ad    bd
+  ac       bc     0
   ----------------------
```

Two Digit Multiplication

```
          a       b
        x c       d
          -----------
          ad     bd
 + ac     bc      0
   -----------------
   ac  ad+bc     bd
```

Two-Digit Multiplication

$(ab) \cdot (cd) = [ac][bc+ad][bd]$

Requires 4 one-digit multiplications and one addition.

Two-Digit Multiplication

$(ab) \cdot (cd) = [ac][bc+ad][bd]$

Requires 4 one-digit multiplications and one addition.

<u>**Trick:**</u> Compute ac, bd, $(a+b)(c+d)$.
Note that $bc+ad = (a+b)(c+d) - ac - bd$.

Requires 3 one-digit multiplications and 4 addition/subtractions.

Two-Digit Multiplication

$(ab) \cdot (cd) = [ac][bc+ad][bd]$

Requires 4 one-digit multiplications and one addition.

Trick: Compute ac, bd, (a+b)(c+d).
Note that $bc+ad = (a+b)(c+d) – ac – bd$.

Requires 3 one-digit multiplications and 4 addition/subtractions.

How can we apply this to larger problems?

Larger Base

$$a_1 \ \ a_2 \ \ a_3 \ldots a_{n/2} \ \ a_{n/2+1} \ldots a_n$$

$$\times \ \ b_1 \ \ b_2 \ \ b_3 \ldots b_{n/2} \ \ b_{n/2+1} \ldots b_n$$

Larger Base

$$\overbrace{a_1\ \ a_2\ \ a_3 \ldots a_{n/2}}^{A}\ \ \overbrace{a_{n/2+1} \ldots a_n}^{B}$$

$$x\ \ \overbrace{b_1\ \ b_2\ \ b_3 \ldots b_{n/2}}^{C}\ \ \overbrace{b_{n/2+1} \ldots b_n}^{D}$$

Larger Base

$$\overbrace{a_1 \quad a_2 \quad a_3 \ldots a_{n/2}}^{A} \quad \overbrace{a_{n/2+1} \ldots a_n}^{B}$$

$$x \quad \overbrace{b_1 \quad b_2 \quad b_3 \ldots b_{n/2}}^{C} \quad \overbrace{b_{n/2+1} \ldots b_n}^{D}$$

$$AC \qquad\qquad AD+BC \qquad\qquad BD$$

Formally

Want to multiply N and M:

Formally

Want to multiply N and M:

1. Let $X \approx \sqrt{(N+M)}$ be a power of 2.

Formally

Want to multiply N and M:

1. Let $X \approx \sqrt{(N+M)}$ be a power of 2.
2. Write $N = AX+B$, $M = CX+D$
   - This can be done by just taking the high and low bits.

Formally

Want to multiply N and M:

1. Let $X \approx \sqrt{(N+M)}$ be a power of 2.
2. Write $N = AX+B$, $M = CX+D$
   – This can be done by just taking the high and low bits.
3. $N \cdot M = AC \cdot X^2+(AD+BC)X+BD$

   $= AC \cdot X^2+[(A+B)(C+D)-AC-BD]X + BD$

   – The multiplications by X are just bit shifts.

Improved Multiplication

```
ImprovedMult(N,M)
  Let X be a power of 2⌊log(N+M)/2⌋
  Write N = AX + B, M = CX + D
  P1 ← Product(A,C)
  P2 ← Product(B,D)
  P3 ← Product(A+B,C+D)
  Return P1X^2 + [P3-P1-P2]X + P2
```

Let X be a power of $2^{\lfloor \log(N+M)/2 \rfloor}$

$P_1 \leftarrow \text{Product}(A,C)$, $P_2 \leftarrow \text{Product}(B,D)$, $P_3 \leftarrow \text{Product}(A+B,C+D)$

Return $P_1X^2 + [P_3-P_1-P_2]X + P_2$

Improved Multiplication

O(n)

```
ImprovedMult(N,M)
  Let X be a power of 2⌊log(N+M)/2⌋
  Write N = AX + B, M = CX + D
  P1 ← Product(A,C)
  P2 ← Product(B,D)
  P3 ← Product(A+B,C+D)
  Return P1X^2 + [P3-P1-P2]X + P2
```

$$
\begin{aligned}
&\texttt{ImprovedMult(N,M)}\\
&\quad \left.\begin{aligned}&\text{Let } X \text{ be a power of } 2^{\lfloor \log(N+M)/2 \rfloor}\\ &\text{Write } N = AX + B,\ M = CX + D\end{aligned}\right\}\ O(n)\\
&\quad P_1 \leftarrow \text{Product}(A,C)\\
&\quad P_2 \leftarrow \text{Product}(B,D)\\
&\quad P_3 \leftarrow \text{Product}(A+B,C+D)\\
&\quad \text{Return } P_1X^2 + [P_3-P_1-P_2]X + P_2
\end{aligned}
$$

Improved Multiplication

```
ImprovedMult(N,M)
   Let X be a power of 2⌊log(N+M)/2⌋
   Write N = AX + B, M = CX + D
   P1 ← Product(A,C)
   P2 ← Product(B,D)
   P3 ← Product(A+B,C+D)
   Return P1X² + [P3-P1-P2]X + P2
```

$O(n)$

$O(n^2)$

Improved Multiplication

```
ImprovedMult(N,M)
  Let X be a power of 2⌊log(N+M)/2⌋
  Write N = AX + B, M = CX + D
  P1 ← Product(A,C)
  P2 ← Product(B,D)
  P3 ← Product(A+B,C+D)
  Return P1X^2 + [P3-P1-P2]X + P2
```

$$\underbrace{\text{Let } X \text{ be a power of } 2^{\lfloor \log(N+M)/2 \rfloor},\ \text{Write } N = AX + B,\ M = CX + D}_{O(n)}$$

$$\underbrace{P_1 \leftarrow \text{Product}(A,C),\ P_2 \leftarrow \text{Product}(B,D),\ P_3 \leftarrow \text{Product}(A+B,C+D)}_{O(n^2)}$$

$$\underbrace{\text{Return } P_1X^2 + [P_3-P_1-P_2]X + P_2}_{O(n)}$$

Improved Multiplication

```
ImprovedMult(N,M)
   Let X be a power of 2^⌊log(N+M)/2⌋
   Write N = AX + B, M = CX + D
   P₁ ← Product(A,C)
   P₂ ← Product(B,D)
   P₃ ← Product(A+B,C+D)
   Return P₁X² + [P₃-P₁-P₂]X + P₂
```

O(n)

$O(n^2)$

O(n)

Runtime: $O(n^2)$.

Improved Multiplication

```
ImprovedMult(N,M)
  Let X be a power of 2⌊log(N+M)/2⌋
  Write N = AX + B, M = CX + D
  P1 ← Product(A,C)
  P2 ← Product(B,D)
  P3 ← Product(A+B,C+D)
  Return P1X² + [P3-P1-P2]X + P2
```

$O(n)$

$O(n^2)$

$O(n)$

Runtime: $O(n^2)$. No asymptotic improvement!

More Detailed Analysis

This algorithm shows no *asymptotic* improvement, but it is better.

More Detailed Analysis

This algorithm shows no *asymptotic* improvement, but it is better.

To analyze this, lets suppose that computing the product of two n-bit numbers using the schoolboy algorithm takes n^2 time.

Improved Multiplication

```
ImprovedMult(N,M)
  Let X be a power of 2⌊log(N+M)/2⌋
  Write N = AX + B, M = CX + D
  P1 ← Product(A,C)
  P2 ← Product(B,D)
  P3 ← Product(A+B,C+D)
  Return P1X^2 + [P3-P1-P2]X + P2
```

$$\text{Let } X \text{ be a power of } 2^{\lfloor \log(N+M)/2 \rfloor}$$

$$N = AX + B,\quad M = CX + D$$

$$P_1 \leftarrow \mathrm{Product}(A,C),\quad P_2 \leftarrow \mathrm{Product}(B,D),\quad P_3 \leftarrow \mathrm{Product}(A+B,C+D)$$

$$\text{Return } P_1X^2 + [P_3-P_1-P_2]X + P_2$$

Improved Multiplication

O(n)

ImprovedMult(N,M)

Let X be a power of $2^{\lfloor \log(N+M)/2 \rfloor}$

Write $N = AX + B$, $M = CX + D$

$P_1 \leftarrow$ Product(A,C)

$P_2 \leftarrow$ Product(B,D)

$P_3 \leftarrow$ Product(A+B,C+D)

Return $P_1X^2 + [P_3-P_1-P_2]X + P_2$

Improved Multiplication

```
ImprovedMult(N,M)
   Let X be a power of 2⌊log(N+M)/2⌋
   Write N = AX + B, M = CX + D
   P₁ ← Product(A,C)
   P₂ ← Product(B,D)
   P₃ ← Product(A+B,C+D)
   Return P₁X² + [P₃-P₁-P₂]X + P₂
```

$O(n)$

$O(n)$

Improved Multiplication

ImprovedMult(N,M)

$O(n)$

Let X be a power of $2^{\lfloor \log(N+M)/2 \rfloor}$

Write N = AX + B, M = CX + D

$P_1 \leftarrow$ Product(A,C)

$P_2 \leftarrow$ Product(B,D)

$3(n/2)^2$

$P_3 \leftarrow$ Product(A+B,C+D)

$O(n)$

Return $P_1X^2 + [P_3-P_1-P_2]X + P_2$

Improved Multiplication

```
ImprovedMult(N,M)
  Let X be a power of 2⌊log(N+M)/2⌋
  Write N = AX + B, M = CX + D
  P1 ← Product(A,C)
  P2 ← Product(B,D)
  P3 ← Product(A+B,C+D)
  Return P1X^2 + [P3-P1-P2]X + P2
```

$O(n)$

$3(n/2)^2$

$O(n)$

Runtime: $(3/4)n^2+O(n)$.

Improved Multiplication

```
ImprovedMult(N,M)
   Let X be a power of 2⌊log(N+M)/2⌋
   Write N = AX + B, M = CX + D
   P1 ← Product(A,C)
   P2 ← Product(B,D)
   P3 ← Product(A+B,C+D)
   Return P1X² + [P3-P1-P2]X + P2
```

O(n)

$3(n/2)^2$

O(n)

Runtime: $(3/4)n^2+O(n)$. Better than n^2!

Further Improvements

So this trick does help. Saving a multiplication at the cost of a few extra additions is a big deal, when multiplications are $O(n^2)$ and additions are $O(n)$.

Can we do better?

Further Improvements

So this trick does help. Saving a multiplication at the cost of a few extra additions is a big deal, when multiplications are $O(n^2)$ and additions are $O(n)$.

Can we do better?

Yes. Our algorithm is still using schoolboy multiplication to do the smaller multiplications. We can instead use our faster algorithm.

Further Improvement

```
KaratsubaMult(N,M)


  Let X be a power of 2^⌊log(N+M)/2⌋
  Write N = AX + B, M = CX + D
  P_1 ← Product(A,C)
  P_2 ← Product(B,D)
  P_3 ← Product(A+B,C+D)
  Return P_1X^2 + [P_3-P_1-P_2]X + P_2
```

Further Improvement

```
KaratsubaMult(N,M)
  If N+M<99, Return Product(N,M)
  Let X be a power of 2⌊log(N+M)/2⌋
  Write N = AX + B, M = CX + D
  P1 ← Product(A,C)
  P2 ← Product(B,D)
  P3 ← Product(A+B,C+D)
  Return P1X^2 + [P3-P1-P2]X + P2
```

$$X = 2^{\lfloor \log(N+M)/2 \rfloor}$$

$$P_1 \leftarrow \text{Product}(A,C),\quad P_2 \leftarrow \text{Product}(B,D),\quad P_3 \leftarrow \text{Product}(A+B,C+D)$$

$$\text{Return } P_1X^2 + [P_3-P_1-P_2]X + P_2$$

Further Improvement

```
KaratsubaMult(N,M)
  If N+M<99, Return Product(N,M)
  Let X be a power of 2⌊log(N+M)/2⌋
  Write N = AX + B, M = CX + D
  P1 ← KaratsubaMult(A,C)
  P2 ← KaratsubaMult(B,D)
  P3 ← KaratsubaMult(A+B,C+D)
  Return P1X2 + [P3-P1-P2]X + P2
```

$$
\begin{aligned}
&\texttt{KaratsubaMult(N,M)}\\
&\quad \texttt{If N+M<99, Return Product(N,M)}\\
&\quad \texttt{Let X be a power of } 2^{\lfloor \log(N+M)/2 \rfloor}\\
&\quad \texttt{Write N = AX + B, M = CX + D}\\
&\quad P_1 \leftarrow \texttt{KaratsubaMult(A,C)}\\
&\quad P_2 \leftarrow \texttt{KaratsubaMult(B,D)}\\
&\quad P_3 \leftarrow \texttt{KaratsubaMult(A+B,C+D)}\\
&\quad \texttt{Return } P_1X^2 + [P_3-P_1-P_2]X + P_2
\end{aligned}
$$

Further Improvement

```
KaratsubaMult(N,M)
  If N+M<99, Return Product(N,M)
  Let X be a power of 2⌊log(N+M)/2⌋
  Write N = AX + B, M = CX + D
  P1 ← KaratsubaMult(A,C)
  P2 ← KaratsubaMult(B,D)
  P3 ← KaratsubaMult(A+B,C+D)
  Return P1X^2 + [P3-P1-P2]X + P2
```

Let X be a power of $2^{\lfloor \log(N+M)/2 \rfloor}$

$P_1 \leftarrow$ KaratsubaMult(A,C), $P_2 \leftarrow$ KaratsubaMult(B,D), $P_3 \leftarrow$ KaratsubaMult(A+B,C+D)

Return $P_1X^2 + [P_3-P_1-P_2]X + P_2$

Further Improvement

```
KaratsubaMult(N,M)
  If N+M<99, Return Product(N,M)
  Let X be a power of 2⌊log(N+M)/2⌋
  Write N = AX + B, M = CX + D
  P1 ← KaratsubaMult(A,C)
  P2 ← KaratsubaMult(B,D)
  P3 ← KaratsubaMult(A+B,C+D)
  Return P1X2 + [P3-P1-P2]X + P2
```

$$\text{Let } X \text{ be a power of } 2^{\lfloor \log(N+M)/2 \rfloor}$$

$$P_1 \leftarrow \text{KaratsubaMult}(A,C)$$

$$P_2 \leftarrow \text{KaratsubaMult}(B,D)$$

$$P_3 \leftarrow \text{KaratsubaMult}(A+B,C+D)$$

$$\text{Return } P_1X^2 + [P_3-P_1-P_2]X + P_2$$

O(n)

Further Improvement

```
KaratsubaMult(N,M)
  If N+M<99, Return Product(N,M)
  Let X be a power of 2⌊log(N+M)/2⌋
  Write N = AX + B, M = CX + D
  P1 ← KaratsubaMult(A,C)
  P2 ← KaratsubaMult(B,D)
  P3 ← KaratsubaMult(A+B,C+D)
  Return P1X^2 + [P3-P1-P2]X + P2
```

O(n)

O(n)

Further Improvement

```
KaratsubaMult(N,M)
  If N+M<99, Return Product(N,M)
  Let X be a power of 2⌊log(N+M)/2⌋
  Write N = AX + B, M = CX + D
  P1 ← KaratsubaMult(A,C)
  P2 ← KaratsubaMult(B,D)
  P3 ← KaratsubaMult(A+B,C+D)
  Return P1X^2 + [P3-P1-P2]X + P2
```

$X = 2^{\lfloor \log(N+M)/2 \rfloor}$, $P_1 \leftarrow$ KaratsubaMult(A,C), $P_2 \leftarrow$ KaratsubaMult(B,D), $P_3 \leftarrow$ KaratsubaMult(A+B,C+D), Return $P_1X^2 + [P_3-P_1-P_2]X + P_2$

O(n)

???

O(n)

Runtime Recurrence

Karatsuba multiplication on inputs of size n spends O(n) time, and then makes three recursive calls to problems of (approximately) half the size.

Runtime Recurrence

Karatsuba multiplication on inputs of size n spends O(n) time, and then makes three recursive calls to problems of (approximately) half the size.

If T(n) is the runtime for n-bit inputs, we have the recursion:

$$T(n) = \begin{cases} O(1) & \text{if } n = O(1) \\ 3T(n/2 + O(1)) + O(n) & \text{otherwise} \end{cases}$$

Runtime Recurrence

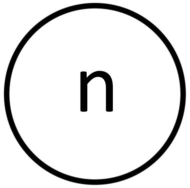
Karatsuba multiplication on inputs of size n spends O(n) time, and then makes three recursive calls to problems of (approximately) half the size.

If T(n) is the runtime for n-bit inputs, we have the recursion:

$$T(n) = \begin{cases} O(1) & \text{if } n = O(1) \\ 3T(n/2 + O(1)) + O(n) & \text{otherwise} \end{cases}$$

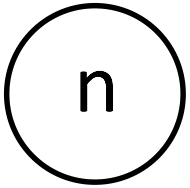
How do we solve this recursion?

Recursive Calls



Recursive Calls

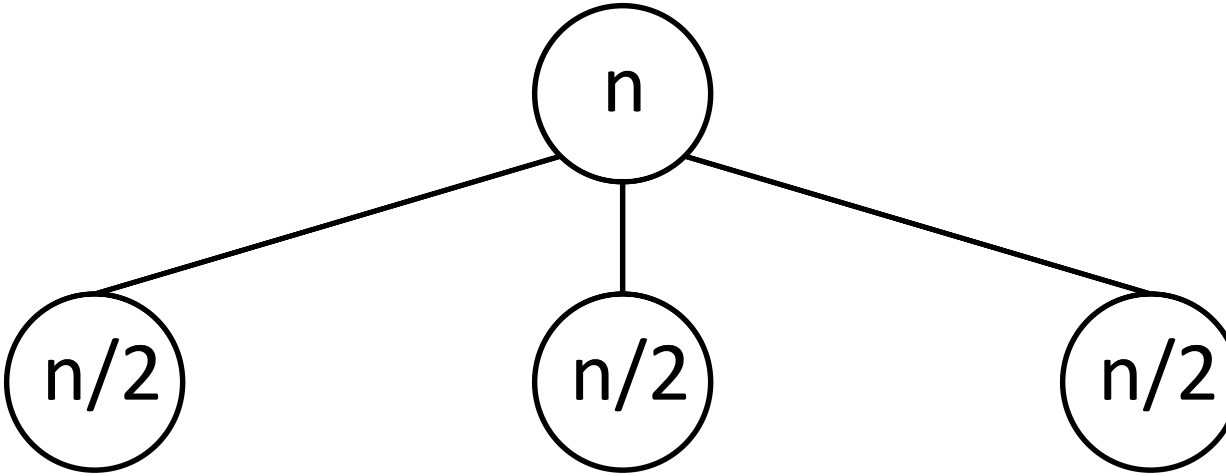
1(n)



Recursive Calls

1(n)

3(n/2)

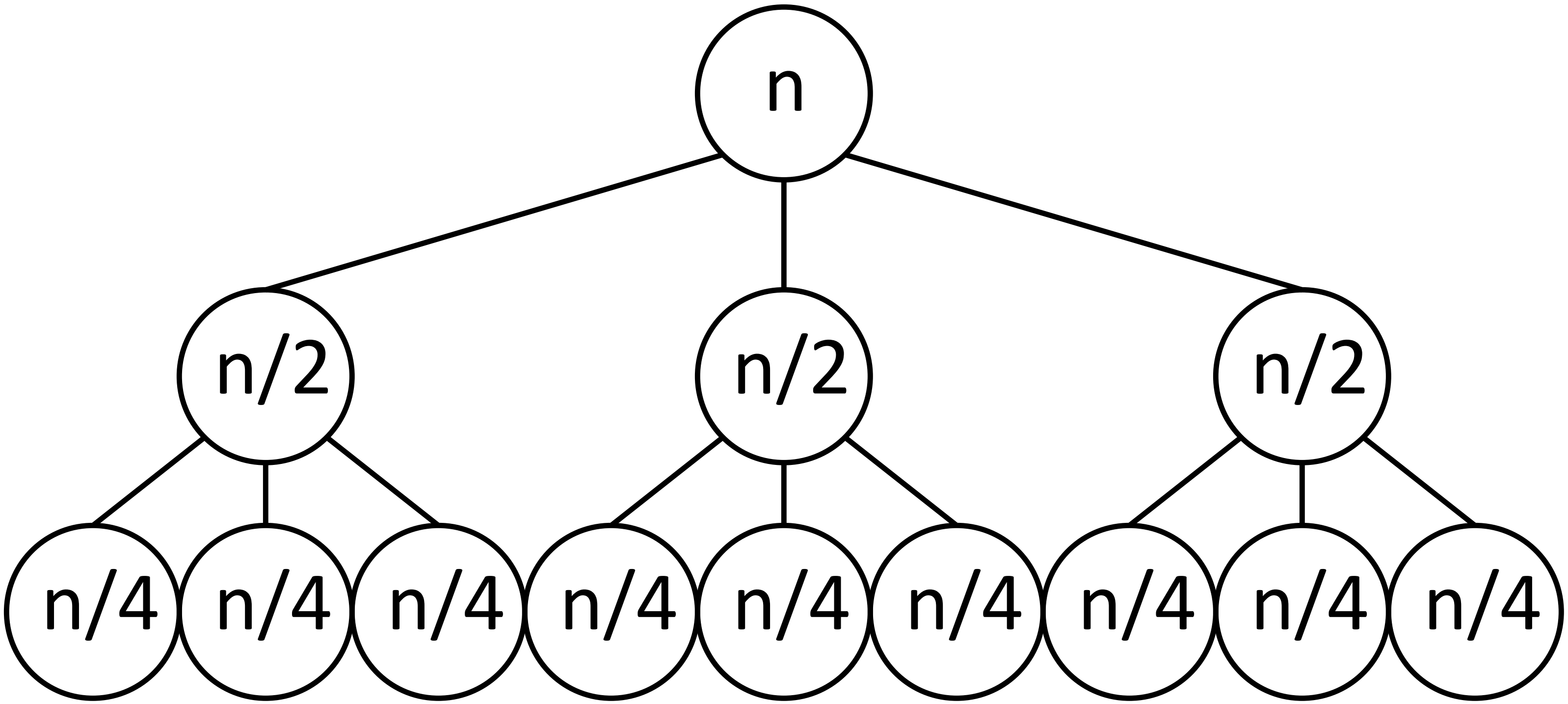


Recursive Calls

1(n)

3(n/2)

9(n/4)



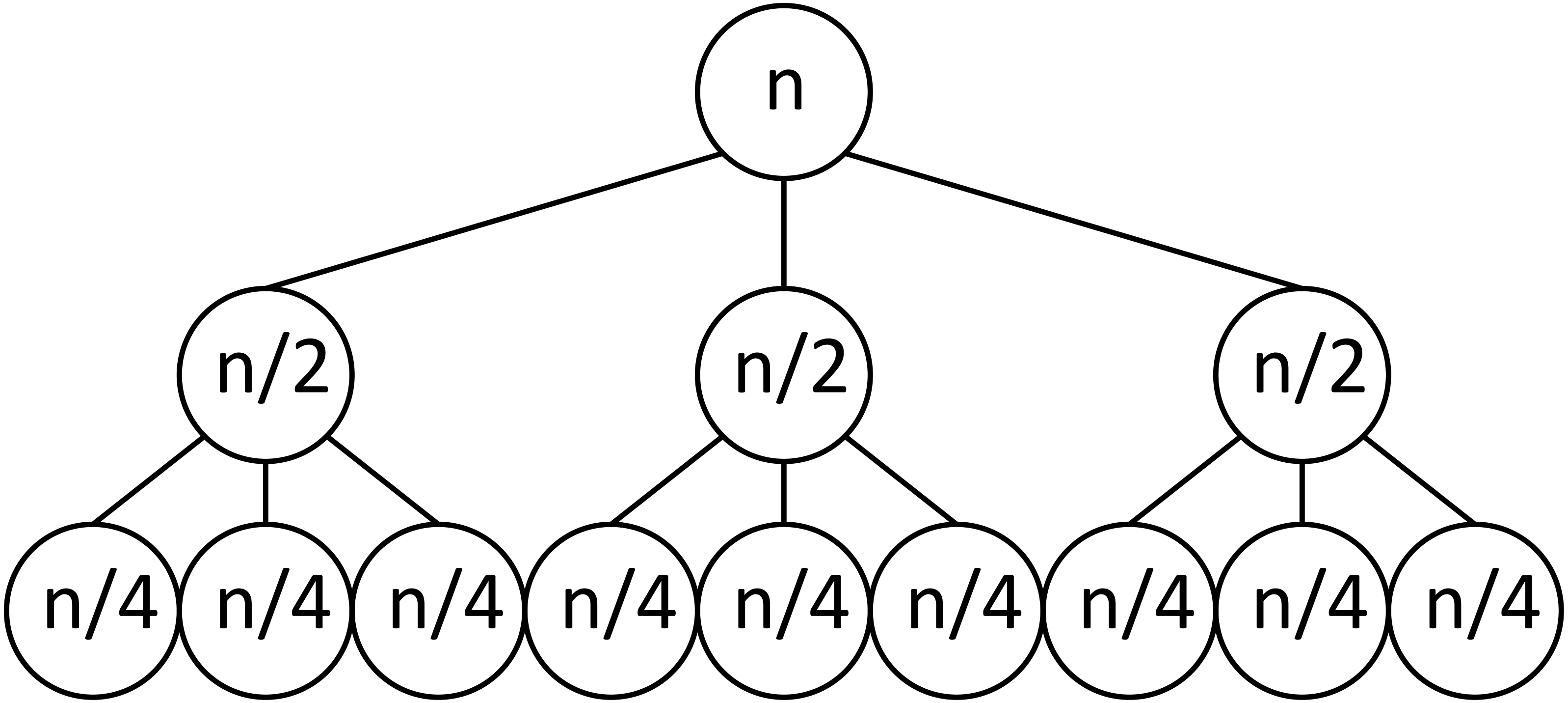
Recursive Calls

$1(n)$

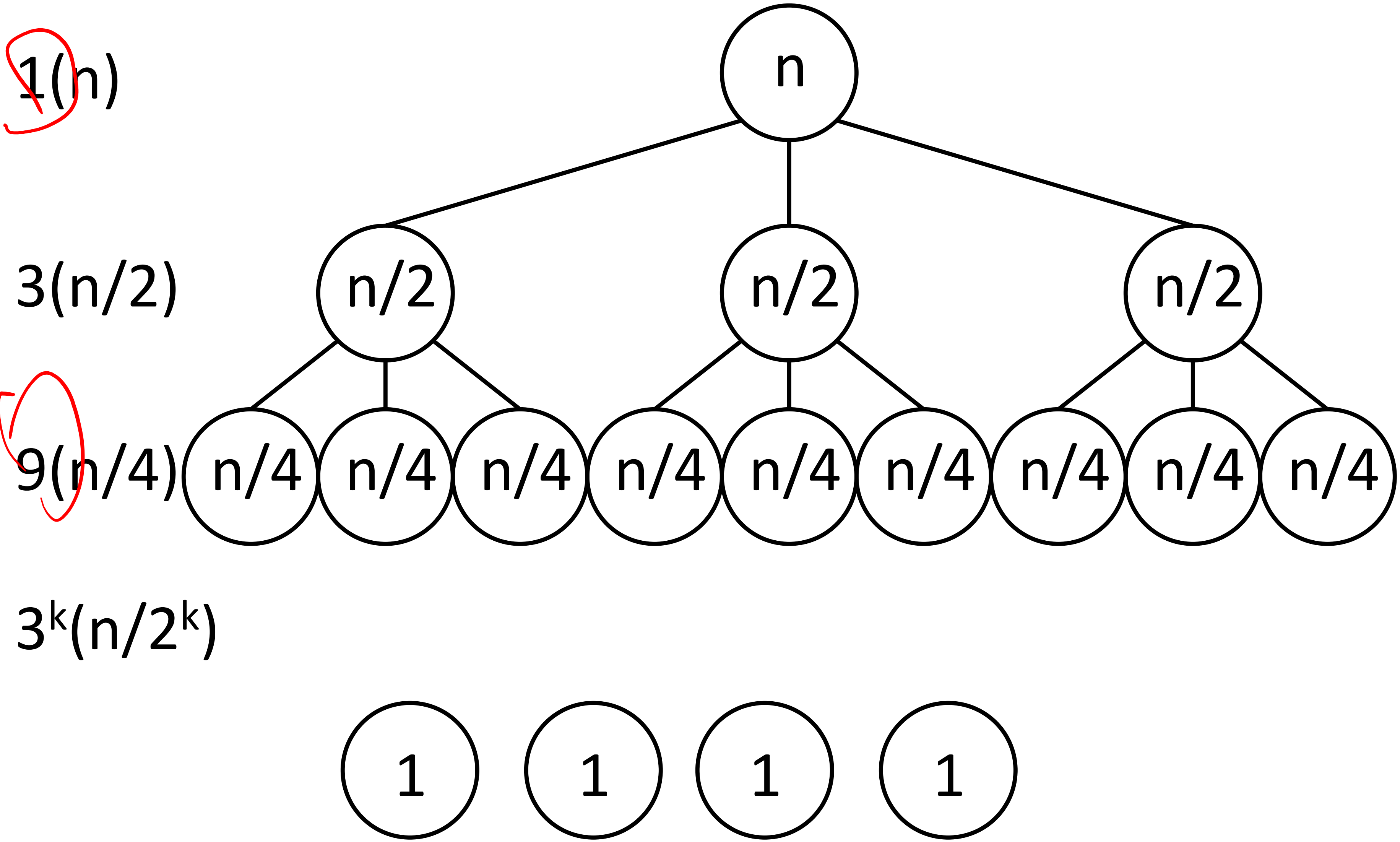
$3(n/2)$

$9(n/4)$

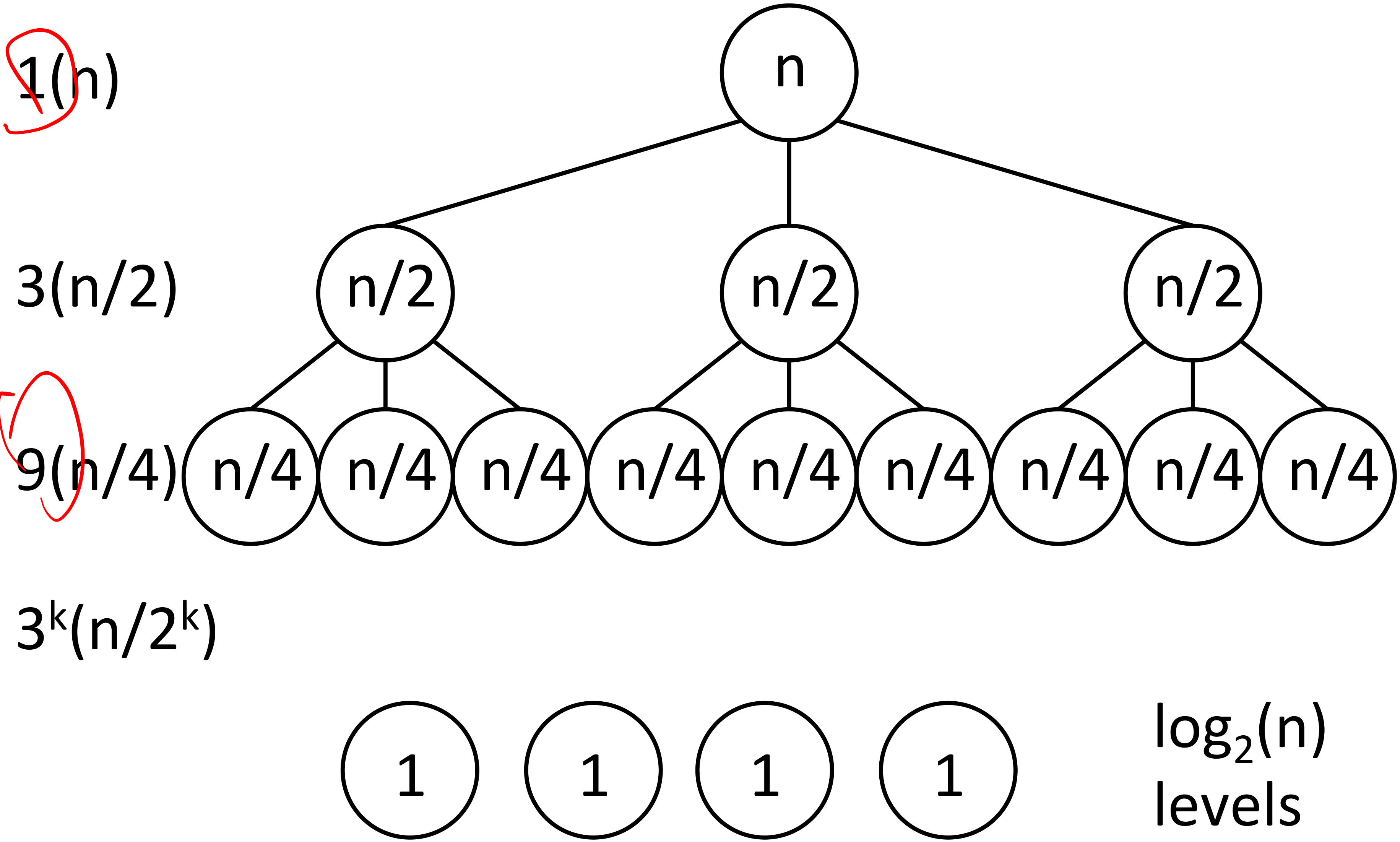
$3^k(n/2^k)$



Recursive Calls



Recursive Calls



Recursive Calls

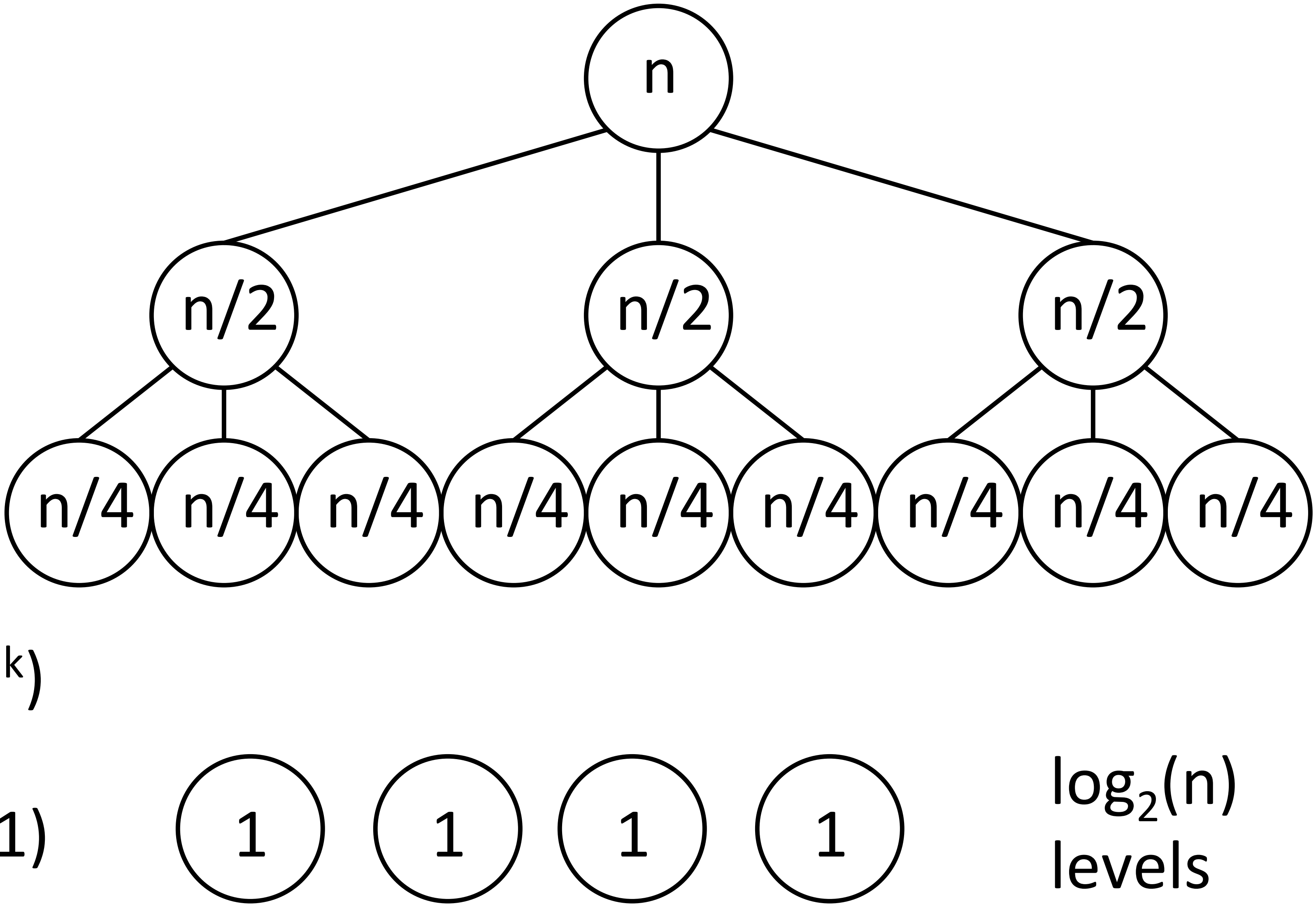
$1(n)$

$3(n/2)$

$9(n/4)$

$3^k(n/2^k)$

$3^{\log(n)}(1)$



$\log_2(n)$ levels

Total Runtime

$$\text{Total Runtime} = \sum_{k=0}^{\log_2(n)} 3^k O(n/2^k)$$

$$= O(n) \sum_{k=0}^{\log_2(n)} (3/2)^k$$

const!

$$= O(n)((3/2)^{\log_2(n)+1} \cancel{-} 1)/\cancel{(3/2 - 1)}$$

$n = 2^{\log_2 n}$

$$= O(n)(3/2)^{\log_2(n)}$$

$$= O(3^{\log_2(n)})$$

$$= O(2^{\log_2(3)\log_2(n)})$$

$$= O(n^{\log_2(3)})$$

$$= O(n^{1.585\ldots}).$$

Divide and Conquer

This is our first example of this general technique:

1. Break problem into pieces.
   - Compute AC, BD, (A+B)(C+D)
2. Recursively solve pieces.
3. Recombine to get answer.
   - $NM=ACX^2+[(A+B)(C+D)-AC-BD]X+BD$

Generalization

We will often get runtime recurrences with D&C looking something like this:

$T(n) = O(1)$ for $n = O(1)$

$T(n) = a\,T(n/b + O(1)) + O(n^d)$ otherwise.

Generalization

We will often get runtime recurrences with D&C looking something like this:

$T(n) = O(1)$ for $n = O(1)$

$T(n) = a\,T(n/b + O(1)) + O(n^d)$ otherwise.

We will need to know how to solve these.

Tracking Recursive Calls

We have:

- 1 recursive call of size n

Tracking Recursive Calls

We have:

- 1 recursive call of size n
- a recursive calls of size n/b+O(1)

Tracking Recursive Calls

We have:

- 1 recursive call of size n
- a recursive calls of size $n/b+O(1)$
- a^2 recursive calls of size $n/b^2+O(1)$

Tracking Recursive Calls

We have:

- 1 recursive call of size n
- a recursive calls of size $n/b+O(1)$
- a^2 recursive calls of size $n/b^2+O(1)$
- …
- a^k recursive calls of size $n/b^k+O(1)$

Tracking Recursive Calls

We have:

- 1 recursive call of size n
- a recursive calls of size $n/b+O(1)$
- a^2 recursive calls of size $n/b^2+O(1)$
- …
- a^k recursive calls of size $n/b^k+O(1)$

Bottoms out when $k = \log_b(n)$.

Runtime

Combining the runtimes from each level of the recursion we get:

$$\text{Total Runtime} = \sum_{k=0}^{\log_b(n)} a^k O((n/b^k)^d)$$

$$= O(n^d) \sum_{k=0}^{\log_b(n)} (a/b^d)^k.$$

Runtime

Combining the runtimes from each level of the recursion we get:

$$\text{Total Runtime} = \sum_{k=0}^{\log_b(n)} a^k O((n/b^k)^d)$$

$$= O(n^d) \sum_{k=0}^{\log_b(n)} (a/b^d)^k.$$

The asymptotics will depend on whether a/b^d is bigger than 1.

Case 1: $a > b^d$

Increasing geometric series dominated by *last* term. Runtime is dominated by recursive calls at the bottom level.

Case 1: $a > b^d$

Increasing geometric series dominated by *last* term. Runtime is dominated by recursive calls at the bottom level.

$$
\begin{aligned}
\text{Runtime} &= O(a^{\log_b(n)}) \\
&= O(b^{\log_b(a)\log_b(n)})) \\
&= O(n^{\log_b(a)}).
\end{aligned}
$$

Case 2: $a < b^d$

Decreasing geometric series is dominated by the first term. Runtime is mostly based on the cleanup steps at the top level.

Case 2: $a < b^d$

Decreasing geometric series is dominated by the first term. Runtime is mostly based on the cleanup steps at the top level.

Runtime = $O(n^d)$

Case 3: $a = b^d$

Every level of the recursion does the same amount of work.

Case 3: $a = b^d$

Every level of the recursion does the same amount of work.

$$\text{Runtime} = O(n^d) \sum_{k=0}^{\log_b(n)} (1^k) = O(n^d \log(n)).$$

Master Theorem

Theorem: Let T(n) be given by the recurrence:

$$T(n) = \begin{cases} O(1) & \text{if } n = O(1) \\ aT(n/b + O(1)) + O(n^d) & \text{otherwise} \end{cases}$$

Then we have that

$$T(n) = \begin{cases} O(n^{\log_b(a)}) & \text{if } a > b^d \\ O(n^d \log(n)) & \text{if } a = b^d \\ O(n^d) & \text{if } a < b^d \end{cases}$$