Master Method

https://cs.pomona.edu/classes/cs140/

Notes on checkpoint

- The checkpoint will be administered through gradescope
- You should come to class
- You have these options (maybe more?)
  - Write answers directly on gradescope (bring your laptop)
    - Should I try to reserve a lab?
  - Write answers on paper and upload to gradescope
  - Write answers on a tablet, export to image or PDF, and upload to gradescope
- You will have plenty of time, so you might want to bring something to read during the down time

Outline

Topics and Learning Objectives

- Learn about the master method for solving recurrences
- Understand how to draw general recursion trees

Exercise

- Applying the Master Method

Extra Resources

- Chapter 4 (sections 4-6) in CLRS
- Algorithms Illuminated: Part 1: 4Chapter 4
- Master Method

Master Method

- For “solving” recurrences

T(n) = the # of operations required to complete algorithm

T(n) = 2T(n/2) + 7n

Base Case: T(1) ≤ base-case-work

Recurrence: T(n) ≤ recursive-work + combine-work

Recurrence Equation

- When an algorithm contains a recursive call to itself
- We usually specify its running time by a recurrence equation
- We also sometimes just call this a “recurrence”
- A recurrence equation describes the overall running time on a problem of size n in terms of the running time on smaller inputs (some fraction of n)

```
T(n)      FUNCTION MergeSort(array)
  O(1)      n = array.length
  O(1)      IF n == 1
  O(1)          RETURN array

  T(n/2)    left_sorted = MergeSort(array[0 ..< n//2])
  T(n/2)    right_sorted = MergeSort(array[n//2 ..< n])

  O(n)      array_sorted = Merge(left_sorted, right_sorted)

  O(1)      RETURN array_sorted
```

Recurrence Equation

$$T(n) = 2\,T(n/2) + O(n)$$

Master Method

“Black Box” for solving recurrences

Assumes all subproblems are of equal size (most algorithms do this)

- The same amount of data is given to each recursive call

An algorithm that splits the subproblems into 1/3 and 2/3 (or an algorithm that splits data randomly) must be *solved* in a different manner. We’ll look at other methods later

Master Method Recurrence Equation

$$T(n) \leq a\, T\left({}^{n}/_{b}\right) + O(n^{d})$$

T(n) : total amount of operations

a : recursive calls (# of subproblems), always >= 1

b : fraction of input size (shrinkage), always > 1

d : extra work needed to combine, always >= 0

What does zero mean for d?

Master Method Cases

$$T(n) \leq a\,T\left(n/_b\right) + O(n^d)$$

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

Master Method Cases

$$T(n) \le a\, T\left({}^{n}/_{b}\right) + O(n^{d})$$

$$T(n) = \begin{cases} O(n^{d} \lg n), & a = b^{d} \quad \text{Case 1} \\ O(n^{d}), & a < b^{d} \quad \text{Case 2} \\ O(n^{\log_b a}), & a > b^{d} \quad \text{Case 3} \end{cases}$$

Exercise

Merge sort

$$T(n) \leq a\,T\left({}^{n}/_{b}\right) + O(n^d)$$

T(n) : total amount of operations

a : recursive calls (# of subproblems), always >= 1

b : fraction of input size (shrinkage), always > 1

d : extra work needed to combine, always >= 0

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

Exercise

Binary search

$$T(n) \leq a\,T\left({}^{n}/_{b}\right) + O(n^{d})$$

T(n) : total amount of operations

a : recursive calls (# of subproblems), always >= 1

b : fraction of input size (shrinkage), always > 1

d : extra work needed to combine, always >= 0

$$T(n) = \begin{cases} O(n^{d} \lg n), & a = b^{d} \\ O(n^{d}), & a < b^{d} \\ O(n^{log_{b} a}), & a > b^{d} \end{cases}$$

Exercise

Closest pair

$$T(n) \leq a\,T\left({}^{n}/_{b}\right) + O(n^{d})$$

T(n) : total amount of operations

a : recursive calls (# of subproblems), always >= 1

b : fraction of input size (shrinkage), always > 1

d : extra work needed to combine, always >= 0

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

Exercise

$T(n) \leq 2\ T(n/2) + O(n^2)$

$$T(n) \leq a\, T\left({}^{n}/_{b}\right) + O(n^d)$$

T(n) : total amount of operations

a : recursive calls (# of subproblems), always >= 1

b : fraction of input size (shrinkage), always > 1

d : extra work needed to combine, always >= 0

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

Integer Multiplication

Input: Two n-digit nonnegative integers, x and y.

Output: The product $x \cdot y$

Assumptions: n is a power of 2

What is the time complexity using the "grade-school" algorithm.

```
  123456789
x 987654321
-----------
```

Multiplication

Constant

Constant

Linear

```
FUNCTION RecursiveIntMult(x, y)
    n = NumDigits(x)
    IF n == 1, RETURN x * y

    a, b = SplitIntIntoHalves(x)
    c, d = SplitIntIntoHalves(y)




    ac = RecursiveIntMult(a, c)
    ad = RecursiveIntMult(a, d)
    bc = RecursiveIntMult(b, c)
    bd = RecursiveIntMult(b, d)



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

What is the recurrence?

Multiplication

T(n) ≤ 4 T(n/2) + O(n)

$$T(n) \le a\, T\left({}^{n}/_{b}\right) + O(n^d)$$

T(n) : total amount of operations

a : recursive calls (# of subproblems), always >= 1

b : fraction of input size (shrinkage), always > 1

d : extra work needed to combine, always >= 0

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

Karatsuba

```
FUNCTION RecursiveIntMult(x, y)
    n = NumDigits(x)
    IF n == 1, RETURN x * y

    a, b = SplitIntIntoHalves(x)
    c, d = SplitIntIntoHalves(y)




    ac = RecursiveIntMult(a, c)
    ad = RecursiveIntMult(a, d)
    bc = RecursiveIntMult(b, c)
    bd = RecursiveIntMult(b, d)



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

```
FUNCTION Karatsuba(x, y)
    n = NumDigits(x)
    IF n == 1, RETURN x * y

    a, b = SplitIntIntoHalves(x)
    c, d = SplitIntIntoHalves(y)

    p = a + b
    q = c + d

    ac = Karatsuba(a, c)
    bd = Karatsuba(b, d)

    pq = Karatsuba(p, q)

    adbc = pq - ac - bd

    RETURN 10^n * ac
         + 10^(n/2) * adbc
         + bd
```

Karatsuba

```
FUNCTION RecursiveIntMult(x, y)
    n = NumDigits(x)
    IF n == 1, RETURN x * y

    a, b = SplitIntIntoHalves(x)
    c, d = SplitIntIntoHalves(y)




    ac = RecursiveIntMult(a, c)
    ad = RecursiveIntMult(a, d)
    bc = RecursiveIntMult(b, c)
    bd = RecursiveIntMult(b, d)



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

```
FUNCTION Karatsuba(x, y)
    n = NumDigits(x)
    IF n == 1, RETURN x * y

    a, b = SplitIntIntoHalves(x)
    c, d = SplitIntIntoHalves(y)

    p = a + b
    q = c + d

    ac = Karatsuba(a, c)
    bd = Karatsuba(b, d)

    pq = Karatsuba(p, q)

    adbc = pq - ac - bd

    RETURN 10^n * ac
         + 10^(n/2) * adbc
         + bd
```

Karatsuba

What is the recurrence?

```
FUNCTION Karatsuba(x, y)
    n = NumDigits(x)
    IF n == 1, RETURN x * y

    a, b = SplitIntIntoHalves(x)
    c, d = SplitIntIntoHalves(y)

    p = a + b
    q = c + d

    ac = Karatsuba(a, c)
    bd = Karatsuba(b, d)

    pq = Karatsuba(p, q)

    adbc = pq - ac - bd

    RETURN 10^n * ac
          + 10^(n/2) * adbc
          + bd
```

Constant

Constant

Linear

Linear

Linear

Karatsuba

T(n) ≤ 3 T(n/3) + O(n)

$$T(n) \le a\, T\left({}^{n}/_{b}\right) + O(n^{d})$$

T(n) : total amount of operations

a : recursive calls (# of subproblems), always >= 1

b : fraction of input size (shrinkage), always > 1

d : extra work needed to combine, always >= 0

$$T(n) = \begin{cases} O(n^{d} \lg n), & a = b^{d} \\ O(n^{d}), & a < b^{d} \\ O(n^{log_{b} a}), & a > b^{d} \end{cases}$$

Iterative Matrix Multiplication

$$z_{ij} = \sum_{k=1}^{n} x_{ik} y_{kj}$$

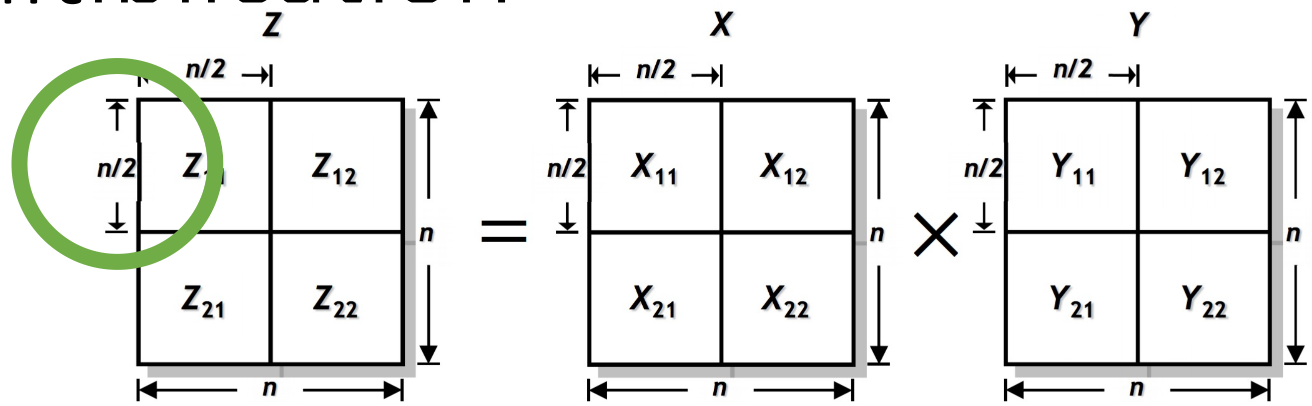
```
FUNCTION IMM(X, Y)
    Z = create_new_matrix(X.size, X.size)

    FOR i IN [0 ..< X.size]
        FOR j IN [0 ..< X.size]
            FOR k IN [0 .. < X.size]
                Z[i][j] += X[i][k] * Y[k][j]

    RETURN Z
```

What are “a”, “b”, and “d”?

Recursive Matrix Multiplication




```
FUNCTION RMM(X, Y)
    IF X.size == 1
        RETURN X * Y

    Z = create_new_matrix(X.size, X.size)

    Z(1,1) = RMM(X(1,1), Y(1,1)) + RMM(X(1,2), Y(2,1)) # Upper left
    Z(1,2) = RMM(X(1,1), Y(1,2)) + RMM(X(1,2), Y(2,2)) # Upper right
    Z(2,1) = RMM(X(2,1), Y(1,1)) + RMM(X(2,2), Y(2,1)) # Lower left
    Z(2,2) = RMM(X(2,1), Y(1,2)) + RMM(X(2,2), Y(2,2)) # Lower right

    RETURN Z
```

Quadratic

Element-wise addition of matrices

What are “a”, “b”, and “d”?

Matrix Multiplication

Recursive matrix multiplication

$$T(n) \leq a\, T\left({}^{n}/_{b}\right) + O(n^{d})$$

T(n) : total amount of operations

a : recursive calls (# of subproblems), always >= 1

b : fraction of input size (shrinkage), always > 1

d : extra work needed to <u>combine</u>, always >= 0

$$T(n) = \begin{cases} O(n^{d} \lg n), & a = b^{d} \\ O(n^{d}), & a < b^{d} \\ O(n^{\log_{b} a}), & a > b^{d} \end{cases}$$

Strassen's Matrix Multiplication

```
FUNCTION SMM(X, Y)
    IF X.size == 1
        RETURN X * Y

    PA = SMM(X(1,1), Y(1,2) − Y(2,2))
    PB = SMM(X(1,1) + X(1,2), Y(2,2))
    PC = SMM(X(2,1) + X(2,2), Y(1,1))
    PD = SMM(X(2,2), Y(2,1) − Y(1,1))
    PE = SMM(X(1,1) + X(2,2), Y(1,1) + Y(2,2))
    PF = SMM(X(1,2) − X(2,2), Y(2,1) + Y(2,2))
    PG = SMM(X(1,1) − X(2,1), Y(1,1) + Y(1,2))
    Z(1,1) = PE + PD − PB + PF
    Z(1,2) = PA + PB
    Z(2,1) = PC + PD
    Z(2,2) = PA + PE − PC − PG

    RETURN Z
```

Quadratic

What are "a", "b", and "d"?

Exercise

Strassen's matrix multiplication

$$T(n) \leq a\, T\left({}^{n}/_{b}\right) + O(n^d)$$

T(n) : total amount of operations

a : recursive calls (# of subproblems), always >= 1

b : fraction of input size (shrinkage), always > 1

d : extra work needed to combine, always >= 0

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

Master Method Proof

Assume

- $T(1) = O(1)$ (this is our base-case)
- $T(n) \leq a\, T(n/b) + cn^d$
- n is a power of b (not necessary, but makes the math easier)

- How did we analyze the running time of merge sort?

Generalizing the Recursion Tree Analysis

For merge sort

- What was the # number of subproblems for a given level L?
- What was the size of each of the subproblems at level L?
- How many total levels were there?

Merge Sort Exercise

1. How many sub-problems are there at level ‘L’? (Note: the top level is ‘Level 0’, the second level is ‘Level 1’, and the bottom level is ‘Level $\log_2(n)$’)

   Answer: 2^L

2. How many elements are there for a given sub-problem found in level ‘L’?

   Answer: $n/2^L$

3. How many computations are performed at a given level? (Note the cost of a ‘merge’ operation was 21m)

   Answer: $2^L \, 21(n/2^L) \rightarrow 21n$

4. What is the total computational cost of merge sort?

   Answer: $21n\,(\log_2(n) + 1)$

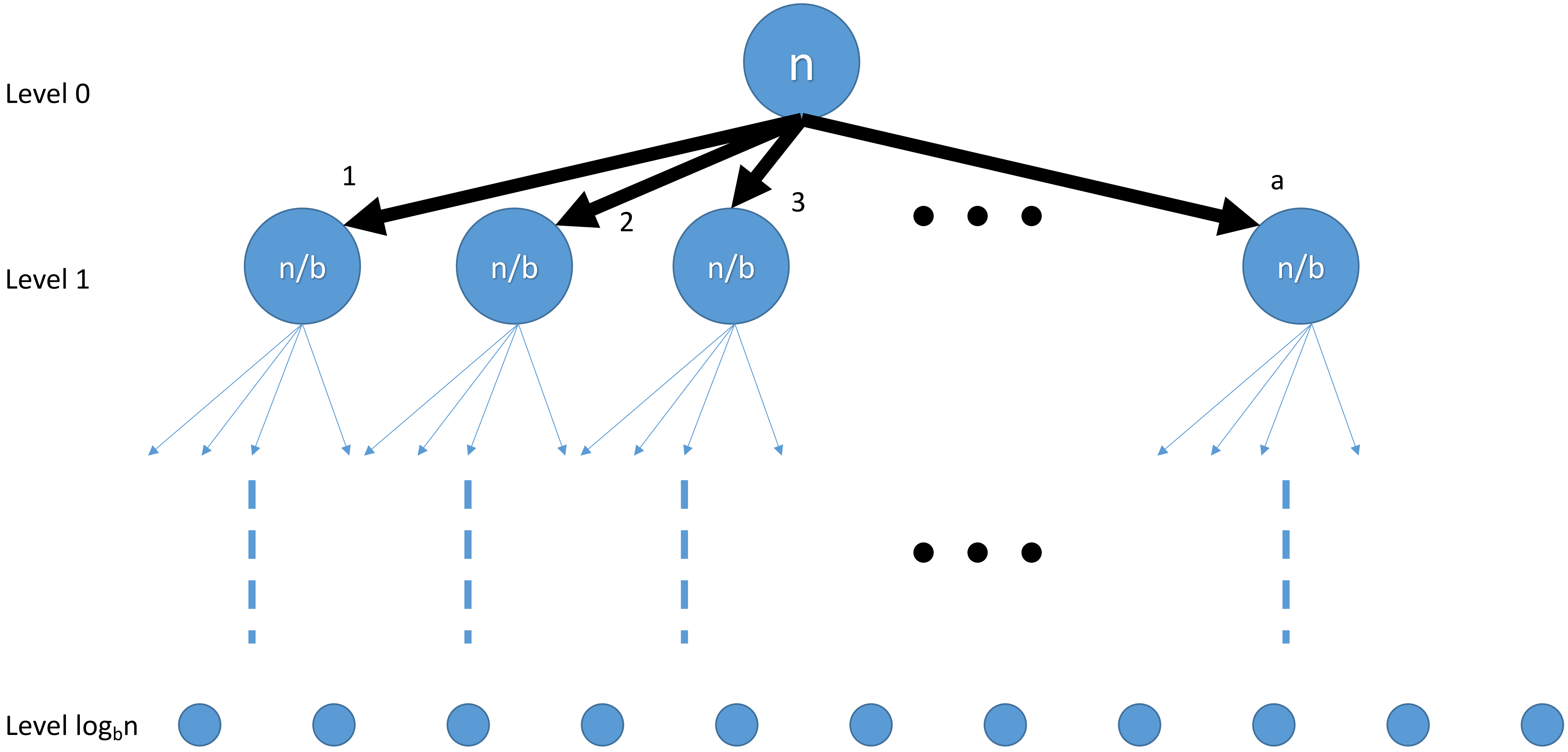
Generalizing the Recursion Tree Analysis

For merge sort

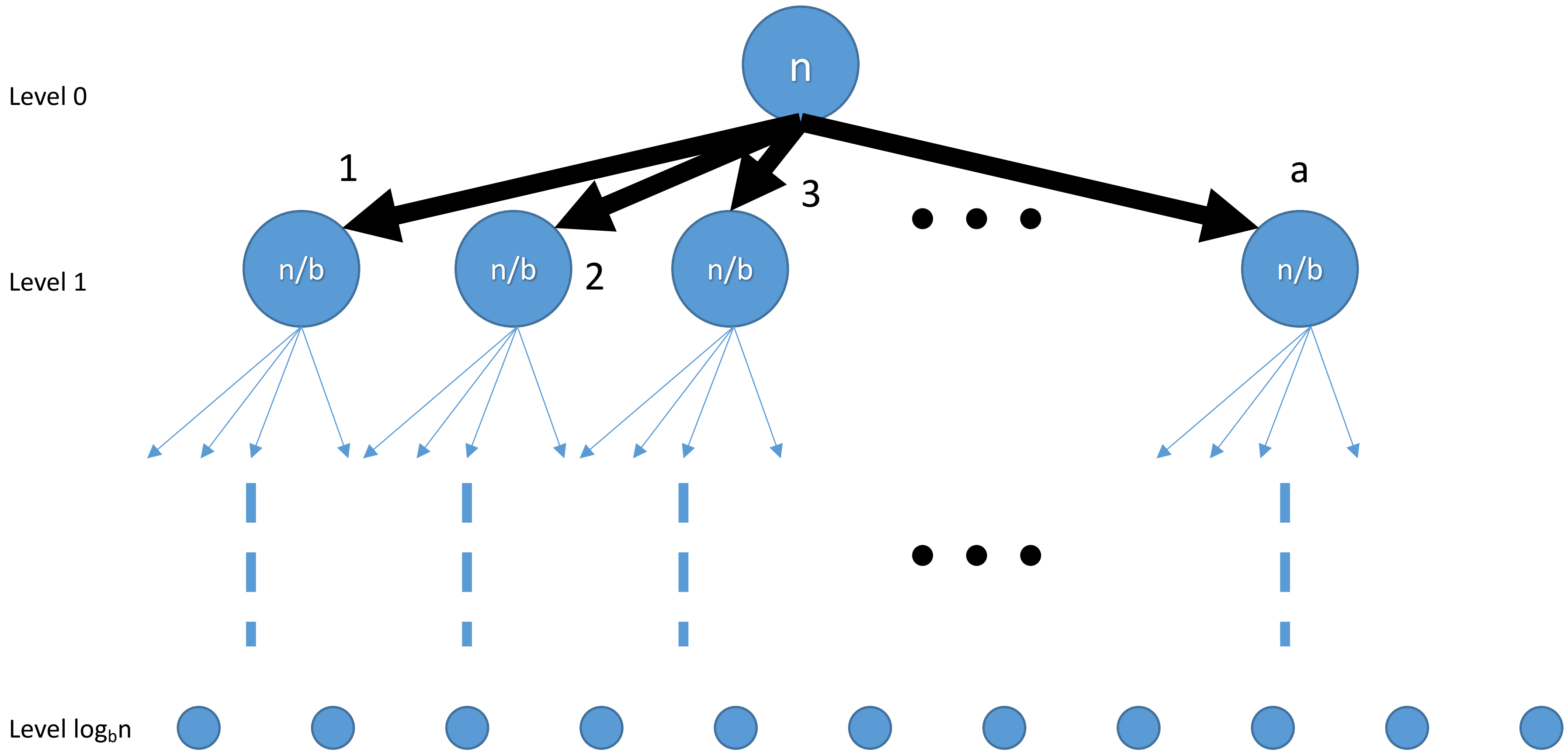
- What was the # number of subproblems for a given level L?
- What was the size of each of the subproblems at level L?
- How many total levels were there?

In the general case

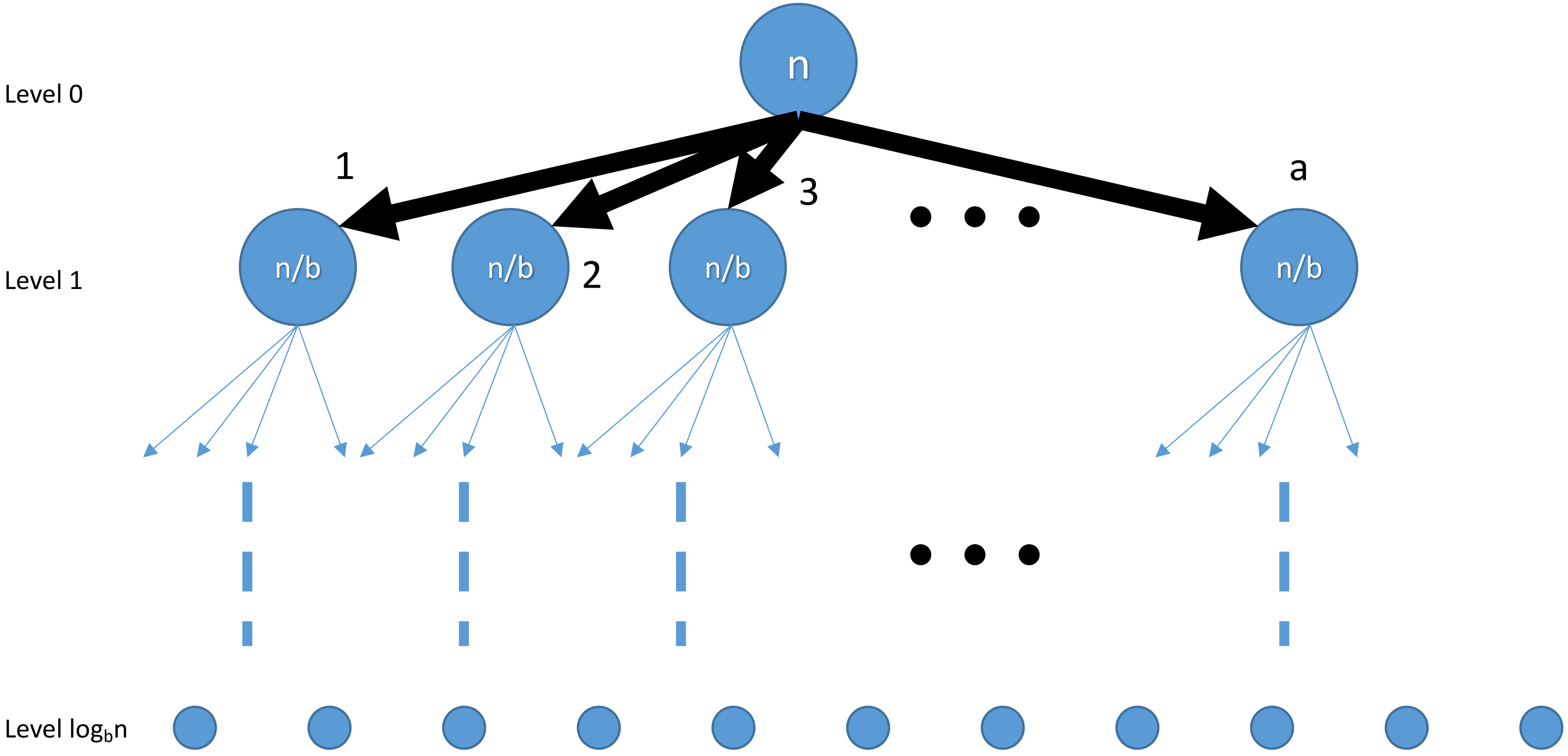
- What is the # number of subproblems for a given level L?
- What is the size of each of the subproblems at level L?
- How many total levels are there?


Level 0
Level 1
Level $\log_b n$
n
n/b
n/b
n/b
n/b
1
2
3
a

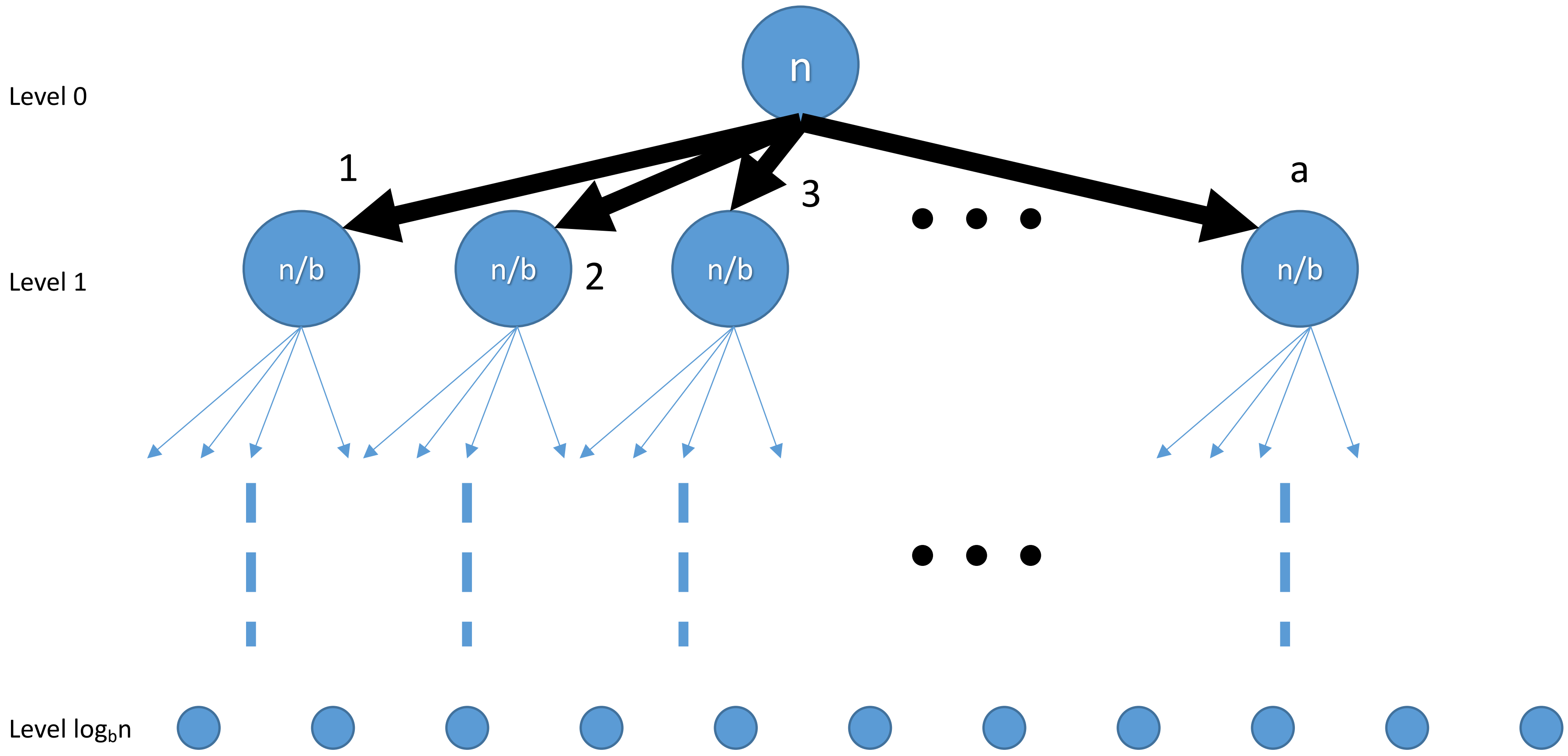
How many sub-problems at level L?



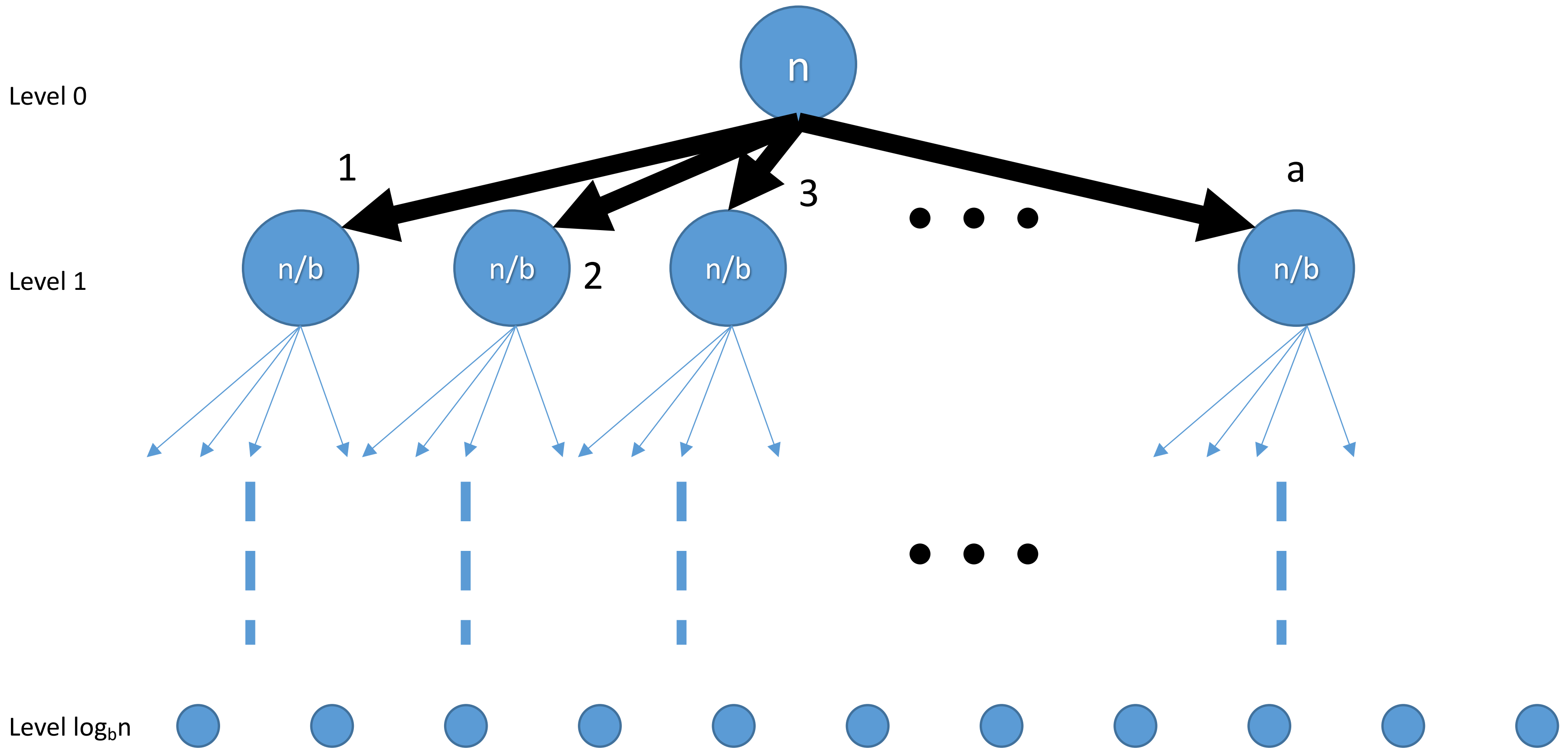
How many elements for each problem at level L?



How much work is done outside of recursion?



What is the total work done at level L?



What is the total work done at level L?

Work at Level L

$$a^L c({}^{n}/_{b^L})^d$$

What is the total work done at level L?

Work at Level L

$$a^L c\left({}^{n}\!/_{b^L}\right)^d$$

Rewrite to group together terms dependent on level

$$cn^d\left({}^{a}\!/_{b^d}\right)^L$$

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

What is the total work done for the tree?

Work at Level L

$$a^L c\left({}^{n}/_{b^L}\right)^d$$

Rewrite to group together terms dependent on level

$$cn^d\left({}^{a}/_{b^d}\right)^L$$

Work done for the entire tree

$$T(n) \leq cn^d \sum_{L=0}^{log_b n} \left({}^{a}/_{b^d}\right)^L$$

Work done by a recursive algorithm

$$T(n) \leq cn^{d} \sum_{L=0}^{log_b n} \left({}^{a}/_{b^d}\right)^{L}$$

Let's look at the cases again

<u>What happens when</u>

$a = b^d$: work stays roughly the same at each level

O(work at each level * number of levels) | $O(n^d \lg n)$

$a < b^d$: work goes down at each level

O(work done at the root) | $O(n^d)$

$a > b^d$: work goes up at each level

O(work done at the leaves) | $O(n^{log_b a})$

Review

- We have three difference cases of trees
  1. Work is similar at each level
  2. Work decreases at each level
  3. Work increases at each level
- These tree lead to our three cases for the Master Method
- What really matters is the ratio between a and b^d

From where do we get the cases?

$$T(n) = \begin{cases} O(n^d \lg n), & a = b^d \quad \text{Case 1} \\ O(n^d), & a < b^d \quad \text{Case 2} \\ O(n^{\log_b a}), & a > b^d \quad \text{Case 3} \end{cases}$$

A few helpers

$$\sum_{i=0}^{k} 1 = k + 1$$

$$\sum_{i=0}^{k} r^i = \frac{r^{k+1} - 1}{r - 1} \text{ when r} > 1$$

$$\sum_{i=0}^{\infty} r^i = \frac{1}{1 - r} \text{ when } r < 1$$

$$log_a(n) = {log_2(n)}/{log_2(a)} = c\,log_2(n) = O(log_2\,(n))\ for\ all\ values\ of\ a \geq 1$$

Proving the Master Method: Case 1

$T(n) \leq \quad cn^d \sum_{L=0}^{log_b n} \left({}^{a}/_{b^d}\right)^L$

$cn^d \sum_{L=0}^{log_b n} (1)^L$

$cn^d (log_b n + 1)$

1. $a = b^d$
2. $\sum_{i=0}^{k} 1 = k + 1$

Claim: $T(n) = O(n^d \lg n)$

$$cn^{d}(log_{b}n + 1) = O(n^{d} \lg n)$$

Master Method

$$T(n) \leq a\, T\left(n/_b\right) + O(n^d)$$

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

Proving the Master Method: Case 2

$T(n) \leq \quad cn^d \sum_{L=0}^{log_b n} (a/_{b^d})^L$

$cn^d \sum_{L=0}^{log_b n} (a/_{b^d})^L$

$cn^d \frac{(a/_{b^d})^{\log_b n+1} - 1}{(a/_{b^d}) - 1}$

$cn^d \frac{1 - (a/_{b^d})^{\log_b n+1}}{1 - (a/_{b^d})}$

1. $a < b^d$
2. $\sum_{i=0}^{k} r^i = \frac{r^{k+1} - 1}{r - 1}$
3. Multiply top and bottom by -1
4. We can remove the complex term from the numerator and keep the original inequality

Proving the Master Method: Case 2

What can we say about this term?

$$T(n) \le \quad cn^d \frac{1}{1-\left({}^{a}/_{b^d}\right)}$$

${}^{a}/_{b^d}$ is constant with respect to n

$$cn^d c_2$$

Claim: $T(n) = O(n^d)$

$$cn^{d}c_2 = O(n^{d})$$

Master Method

$$T(n) \le a\, T\left(\frac{n}{b}\right) + O(n^d)$$

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

Proving the Master Method: Case 3

$T(n) \leq \quad cn^d \sum_{L=0}^{log_b n} ({}^a/_{b^d})^L$

$cn^d \sum_{L=0}^{log_b n} ({}^a/_{b^d})^L$

$cn^d ({}^a/_{b^d})^{log_b n}$

$ca^{\log_b n}$

Claim: $T(n) = O(n^{log_b a})$

1. $a > b^d$
2. Last term of summation is asymptotically largest: $({}^a/_{b^d})^{log_b n}$
3. Distribute the exponent and simplify

$$c a^{\log_b n} = O(n^{log_b a})$$

Master Method

$$T(n) \leq a\, T\left({}^{n}/_{b}\right) + O(n^d)$$

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

Master Method Summary

1. We analyzed a generalized recursion tree
2. Counted the amount of work done at each level
3. Counted the amount of work done by the tree
4. Found that we have three different types of trees
    1. Same rate throughout (case 1: $a = b^d$)
    2. Root dominates (case 2: $a < b^d$)
    3. Leaves dominate (case 3: $a > b^d$)
5. Saw that these trees relate to the difference master method cases

$$T(n) \leq a\,T\left(n/_b\right) + O(n^d)$$

T(n) : total amount of operations

a : recursive calls (# of subproblems), always >= 1

b : fraction of input size (shrinkage), always > 1

d : extra work needed to combine, always >= 0

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