CS3000: Algorithms & Data
Paul Hand

Lecture 3:

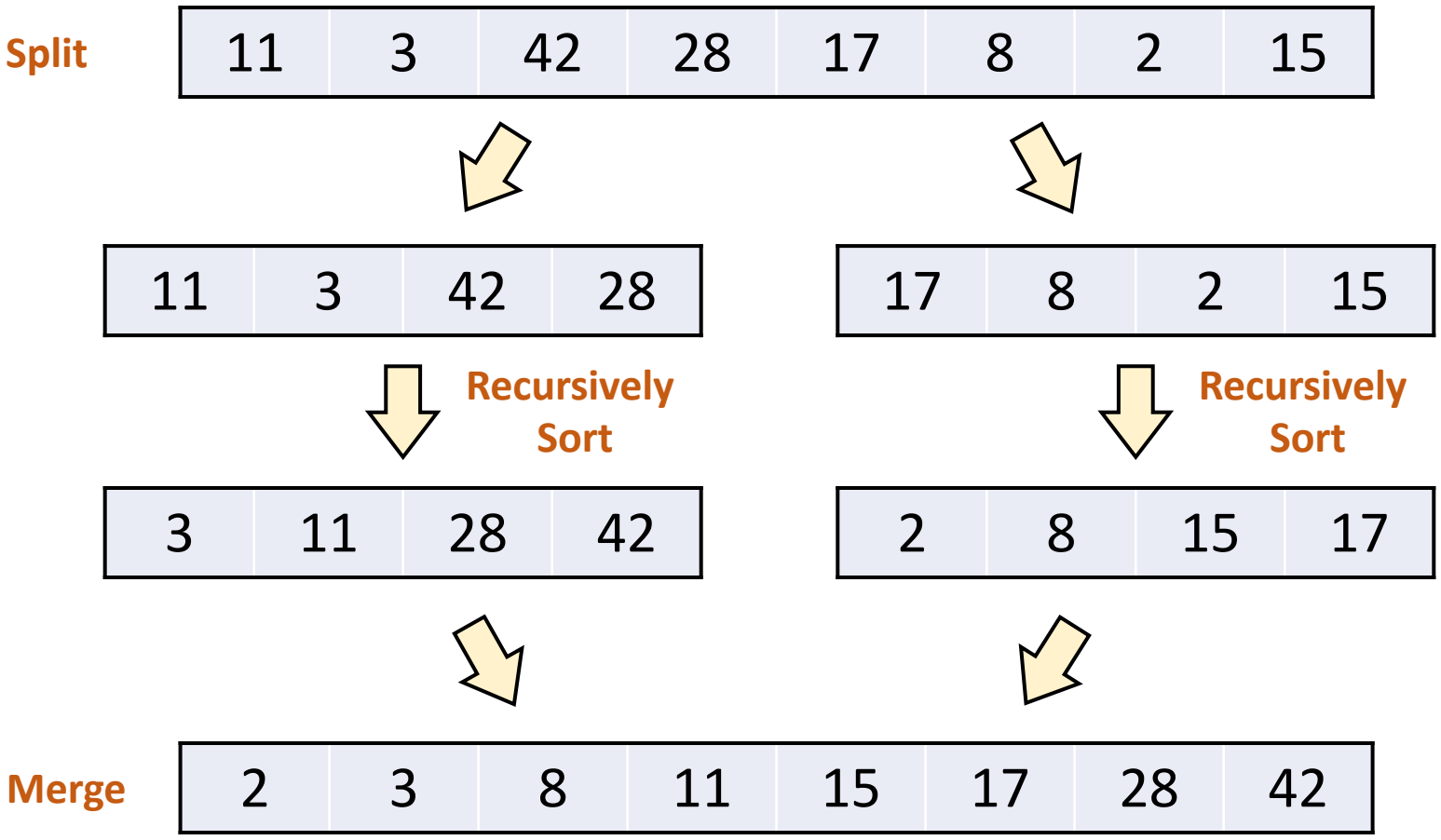
- Asymptotic Analysis
- Divide and Conquer: Mergesort

Jan 23, 2019

Divide and Conquer Algorithms

- Split your problem into smaller subproblems
- Recursively solve each subproblem
- Combine the solutions to the subprobelms

Divide and Conquer: Mergesort



Merging two sorted lists

```
Merge(L,R):
  Let n ← len(L) + len(R)
  Let A be an array of length n
  j ← 1, k ← 1,

  For i = 1,…,n:
    If (j > len(L)):              // L is empty
      A[i] ← R[k], k ← k+1
    ElseIf (k > len(R)):          // R is empty
      A[i] ← L[j], j ← j+1
    ElseIf (L[j] <= R[k]):        // L is smallest
      A[i] ← L[j], j ← j+1
    Else:                         // R is smallest
      A[i] ← R[k], k ← k+1

  Return A
```

Consider A[i]

WTS $A[i] \le A[\ell]\ \forall \ell \ge i$

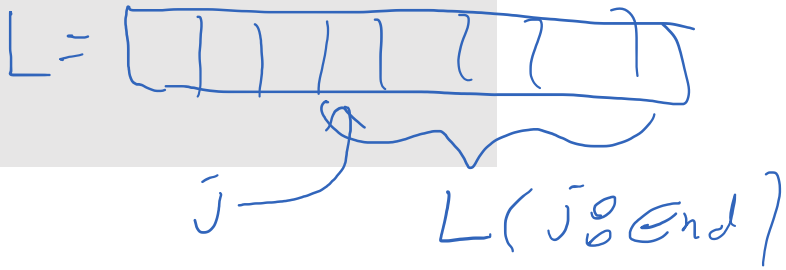
At i^{th} iteration

$A[i] = \min(L[j], R[k])$

$A[i] \le$ all entries $L[j:end]$

$A[i] \le$ all entries of $R[k:end]$

So $A[i] \le A[\ell]\ \forall \ell \ge i$



Merging two sorted lists

```
Merge(L,R):
  Let n ← len(L) + len(R)
  Let A be an array of length n
  j ← 1, k ← 1,

  For i = 1,…,n:
    If (j > len(L)):              // L is empty
      A[i] ← R[k], k ← k+1
    ElseIf (k > len(R)):          // R is empty
      A[i] ← L[j], j ← j+1
    ElseIf (L[j] <= R[k]):        // L is smallest
      A[i] ← L[j], j ← j+1
    Else:                         // R is smallest
      A[i] ← R[k], k ← k+1

  Return A
```

- **Prove:** If L and R are sorted from smallest to largest, then A is sorted from smallest to largest.

$A[i] \leq A[i+1]$ (WTS)

First thing added is $\min(L[1], R[1])$

Because L, R sorted

$L[1] = \min(L)$

$R[1] = \min(R)$

$\Rightarrow \min(L[1], R[1]) = \min(L \cup R)$

General fact

MergeSort Algorithm

```
MergeSort(A):
  If (len(A) = 1): Return A     // Base Case

  Let m ← ⌈len(A)/2⌉             // Split
  Let L ← A[1:m], R ← A[m+1:n]

  Let L ← MergeSort(L)          // Recurse
  Let R ← MergeSort(R)

  Let A ← Merge(L,R)            // Merge

  Return A
```

3

Correctness of Mergesort

- **Claim:** The algorithm **`Mergesort`** is correct

```
MergeSort(A):
  If (len(A) = 1): Return A          // Base Case

  Let m ← ⌈len(A)/2⌉                 // Split
  Let L ← A[1:m], R ← A[m+1:n]

  Let L ← MergeSort(L)               // Recurse
  Let R ← MergeSort(R)

  Let A ← Merge(L,R)                 // Merge

  Return A
```

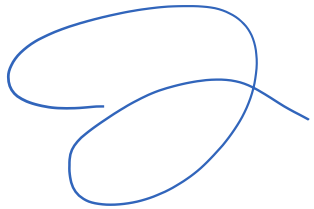
For all

$\forall n \in \mathbb{N}$ $\forall$ list A with n numbers Mergesort returns A in sorted order

Inductive Hypothesis: $H(n) = \forall$ A of size n MergeSort is correct

Base Case: $H(1)$ is true, obviously

Inductive Step: Assume $H(1), \ldots, H(n)$ are all true. We'll prove $H(n+1)$.



Correctness of Mergesort

- **Claim:** The algorithm **`Mergesort`** is correct

```
MergeSort(A):
  If (len(A) = 1): Return A      // Base Case

  Let m ← ⌈len(A)/2⌉             // Split
  Let L ← A[1:m], R ← A[m+1:n]
  Let L ← MergeSort(L)           // Recurse
  Let R ← MergeSort(R)

  Let A ← Merge(L,R)             // Merge

  Return A
```

Inductive Step:

Assume: MergeSort is correct for all A of size $\leq n$

Want to show: MergeSort is correct for all A of size $n+1$

Consider an A of size $n+1$.

① $\lceil \frac{n+1}{2} \rceil$ & $n - \lceil \frac{n+1}{2} \rceil \leq n$

m — size of L — size of R

② L, R both correctly sorted by inductive hypothesis

③ L, R sorted $\Rightarrow$ A sorted.

May want to prove

Running Time of Mergesort

```
MergeSort(A):
  If (n = 1): Return A

  Let m ← ⌈n/2⌉
  Let   L ← A[1:m]
        R ← A[m+1:n]

  Let L ← MergeSort(L)
  Let R ← MergeSort(R)
  Let A ← Merge(L,R)

  Return A
```

1 op for T(1) case

O(n)

$T(\frac{n}{2})$

$T(\frac{n}{2})$

cn

T(n) = time to sort list of size n

T(1) = c

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

So what is T(n)?

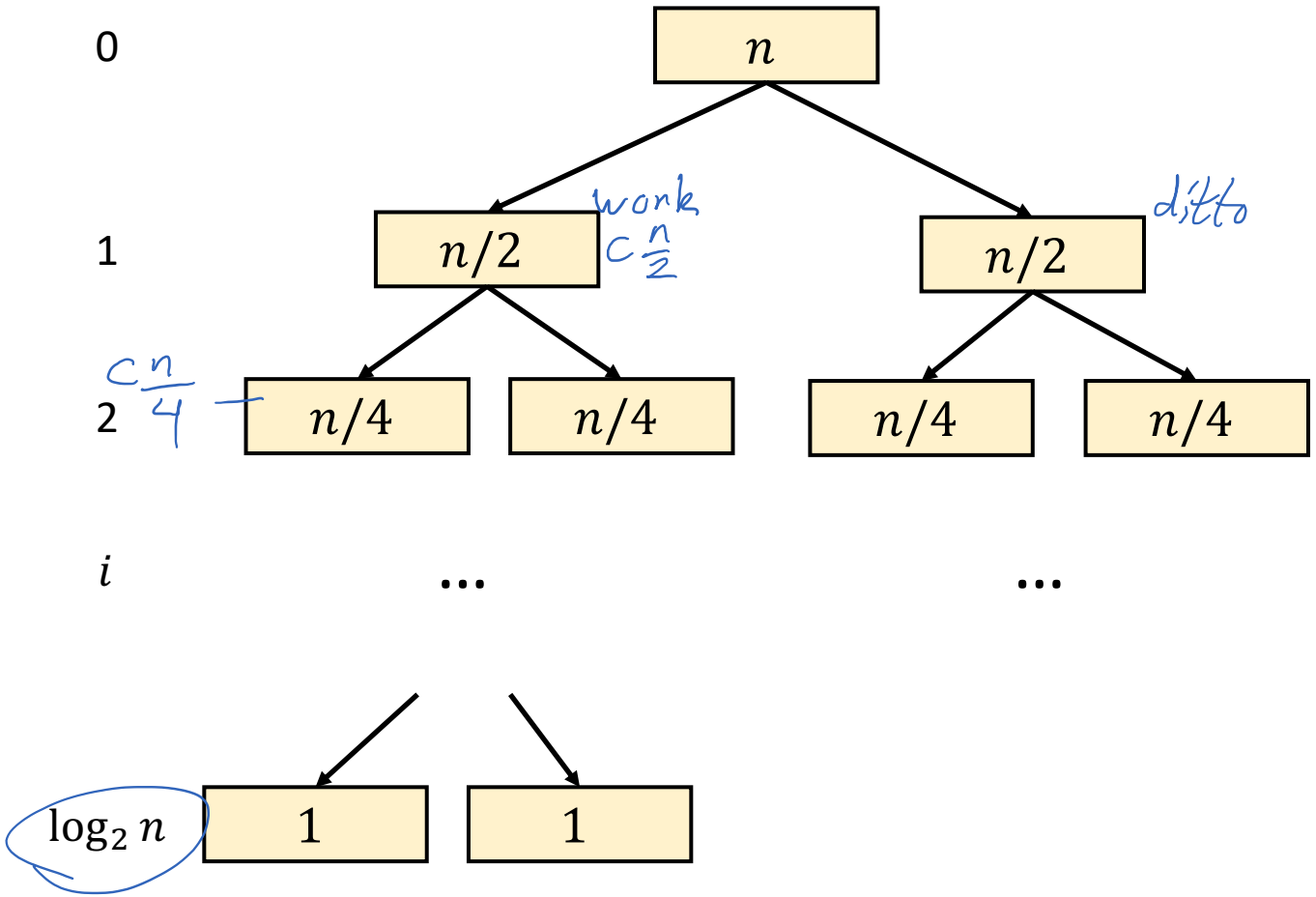
recurrence formula

Find Runtime using Recursion Trees and Summation

$$T(n) = 2 \cdot T(n/2) + Cn$$
$$T(1) = C$$

↗ problem How does T scale for large n?

Level | Input Size at this Level | Work at this Level



$= \Theta(n \log n)$

$= Cn \log_2(2n)$

Total Work $Cn + Cn + \ldots + Cn = (\log_2 n + 1)Cn$

Proof by Induction

$$T(n) = 2 \cdot T(n/2) + Cn$$
$$T(1) = C$$

H(n):

- **Claim:** $T(n) = Cn \log_2 2n$

Proof:

Base case: $n=1$. Trivial, as $T(1) = C$

General case: Assume $H(1), H(2), \ldots H(n-1)$ is true.
We want to show $H(n)$ is true.

$$T(n) = 2T(n/2) + Cn$$
$$= 2\left(C\frac{n}{2}\log_2 2\frac{n}{2}\right) + Cn \quad \text{— by inductive hypothesis}$$
$$= 2\left(C\frac{n}{2}\log_2 n\right) + Cn$$
$$= Cn\log_2 n + Cn = Cn\log_2 2n.$$

Mergesort Summary

much faster than n^2 time

- Sort a list of n numbers in $\Theta(n \log_2 n)$ time
  - Can actually sort anything that allows comparisons
  - No comparison based algorithm can be (much) faster
- Divide-and-conquer
  - Break the list into two halves, sort each one and merge
  - Key Fact: Merging sorted lists is easier than sorting
- Proof of correctness
  - Proof by induction, summation & recursion trees
- Analysis of running time
  - Recurrences
    - Can prove using recursion tree and summing work at each level
    - Can prove using induction
    - Can use "Master Theorem" for recurrences

Solving Recurrences:
“The Master Theorem”

Activity

Θ or O or Ω

- Suppose $r > 0$.
- For large ℓ, how does $S = \sum_{i=0}^{\ell} r^i$ behave?

- What tools do we have to help you? What can you claim?

Cases I $r < 1$ $\sum_{i=0}^{\infty} r^i = \frac{1}{1-r}$ $S = \Theta(1)$

II $r = 1$ $S = \ell$ $\Theta(\ell)$

III $r > 1$ $S \leq \ell \cdot r^\ell$ $O(\ell r^\ell)$?

The "Master Theorem"

problem of size n

- Generic divide-and-conquer algorithm:
  - Split into $\boldsymbol{a}$ pieces of size $\frac{\boldsymbol{n}}{\boldsymbol{b}}$ and merge in time $O(n^{\boldsymbol{d}})$
- Recipe for recurrences of the form:
  - $T(n) = \boldsymbol{a} \cdot T(n/\boldsymbol{b}) + Cn^{\boldsymbol{d}}$

How do we solve?

Merge Sort
$a=2$
$b=2$
$d=1$

Geometric Series

- Claim: $S = \sum_{i=0}^{\ell} r^i = \frac{r^{\ell+1} - 1}{r - 1}$
- Why?

case $r \neq 1$

$$S = 1 + r + r^2 + \cdots + r^{\ell}$$

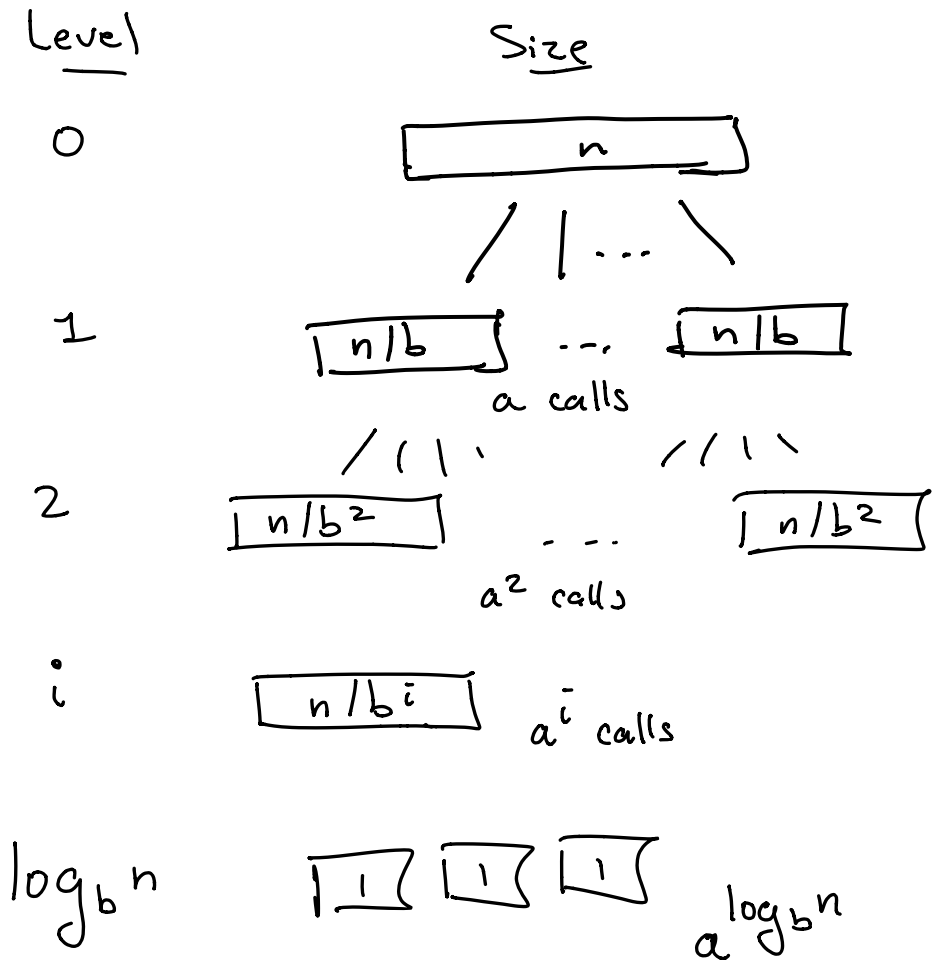
$$rS = \quad r + r^2 + \cdots + r^{\ell} + r^{\ell+1}$$

$rS - S = r^{\ell+1} - 1$ $\qquad S = \frac{r^{\ell+1} - 1}{r - 1}$

- Solution $S = \frac{1 - r^{\ell+1}}{1 - r} = \frac{r^{\ell+1} - 1}{r - 1}$

Recursion Tree

- $T(n) = \boldsymbol{a}T(n/\boldsymbol{b}) + n^{\boldsymbol{d}}$



Activity 9

Where is the most work happening?

which level

If $\frac{a}{b^d} < 1$, level 0!

If $\frac{a}{b^d} > 1$, level $\log_b n$

If $\frac{a}{b^d} = 1$, all comparable

Recursion Tree

- $T(n) = \boldsymbol{a}T(n/\boldsymbol{b}) + n^{\boldsymbol{d}}$
- $\left(\frac{\boldsymbol{a}}{\boldsymbol{b}^{\boldsymbol{d}}}\right) > 1$

$$\sum_{i=0}^{\log_b n} \left(\frac{a}{b^d}\right)^i n^d = n^d \frac{\left(\frac{a}{b^d}\right)^{\log_b n + 1} - 1}{\left(\frac{a}{b^d}\right) - 1} = \Theta\left(\frac{a}{b^d}^{\log_b n} \cdot n^d\right)$$

apply geometric summation

$$= \Theta\left(n^{\log_b a - d} \cdot n^d\right)$$

$$= \boxed{\Theta(n^{\log_b a})}$$

most expensive level of recursion tree is the last

Use $\left(\frac{a}{b^d}\right)^{\log_b n} = n^{\log_b\left(\frac{a}{b^d}\right)}$

$= n^{\log_b a - d} =$

Fact:

$x^{\log_b n} = n^{\log_b x}$

Why?

$x^{\log_b n} = b^{\log_b(x^{\log_b n})} = b^{\log_b n \log_b x} = (b^{\log_b n})^{\log_b x} = n^{\log_b x}$

$$\boxed{T(n) = \Theta(n^{\log_b a})}$$

Recursion Tree

- $T(n) = \boldsymbol{a}T(n/\boldsymbol{b}) + n^{\boldsymbol{d}}$
- $\left(\frac{\boldsymbol{a}}{\boldsymbol{b}^{\boldsymbol{d}}}\right) = 1$

$$\sum_{i=0}^{\log_b n} \left(\frac{a}{b^d}\right)^i n^d = \sum_{i=0}^{\log_b n} n^d = \Theta(\log_b n \cdot n^d)$$

$$= \Theta(n^d \log_b n)$$

all levels of tree are equally expensive

Recursion Tree

- $T(n) = \boldsymbol{a}T(n/\boldsymbol{b}) + n^{\boldsymbol{d}}$
- $\left(\frac{\boldsymbol{a}}{\boldsymbol{b}^{\boldsymbol{d}}}\right) < 1$

$$\sum_{i=0}^{\log_b n} \left(\frac{a}{b^d}\right)^i n^d = n^d \frac{\left(\frac{a}{b^d}\right)^{\log_b n + 1} - 1}{\left(\frac{a}{b^d}\right) - 1} = \Theta(n^d)$$

apply geometric summation

because $\left(\frac{a}{b^d}\right) < 1$,

$\left(\frac{a}{b^d}\right)^{\log_b n} \to 0$ as $n \to \infty$

expensive part is at the 0^{th} level of recursion tree

The “Master Theorem”

- Recipe for recurrences of the form:
  - $T(n) = \boldsymbol{a} \cdot T(n/\boldsymbol{b}) + Cn^{\boldsymbol{d}}$
- Three cases:
  - $\left(\frac{\boldsymbol{a}}{\boldsymbol{b}^{\boldsymbol{d}}}\right) > 1 : \; T(n) = \Theta\left(n^{\log_{\boldsymbol{b}} \boldsymbol{a}}\right)$
  - $\left(\frac{\boldsymbol{a}}{\boldsymbol{b}^{\boldsymbol{d}}}\right) = 1 : \; T(n) = \Theta\left(n^{\boldsymbol{d}} \log n\right)$
  - $\left(\frac{\boldsymbol{a}}{\boldsymbol{b}^{\boldsymbol{d}}}\right) < 1 : \; T(n) = \Theta\left(n^{\boldsymbol{d}}\right)$

Practice

- Use the Master Theorem to Solve:
  - $T(n) = 16 \cdot T\left(\frac{n}{4}\right) + n^2$
  - $T(n) = 21 \cdot T\left(\frac{n}{5}\right) + n^2$
  - $T(n) = 2 \cdot T\left(\frac{n}{2}\right) + 1$
  - $T(n) = 1 \cdot T\left(\frac{n}{2}\right) + 1$

- Recipe for recurrences of the form:
  - $T(n) = \boldsymbol{a} \cdot T(n/\boldsymbol{b}) + Cn^{\boldsymbol{d}}$
- Three cases:
  - $\left(\frac{\boldsymbol{a}}{\boldsymbol{b}^{\boldsymbol{d}}}\right) > 1 : \; T(n) = \Theta\left(n^{\log_{\boldsymbol{b}} \boldsymbol{a}}\right)$
  - $\left(\frac{\boldsymbol{a}}{\boldsymbol{b}^{\boldsymbol{d}}}\right) = 1 : \; T(n) = \Theta\left(n^{\boldsymbol{d}} \log n\right)$
  - $\left(\frac{\boldsymbol{a}}{\boldsymbol{b}^{\boldsymbol{d}}}\right) < 1 : \; T(n) = \Theta\left(n^{\boldsymbol{d}}\right)$