CS 341: ALGORITHMS

Lecture 2: divide & conquer I

Readings: see website

Trevor Brown

https://student.cs.uwaterloo.ca/~cs341

trevor.brown@uwaterloo.ca

1

DIVIDE AND CONQUER

Notable algorithms: mergesort, quicksort, binary search, …

2

DIVIDE-AND-CONQUER DESIGN STRATEGY

- **divide:** Given a problem instance I, construct one or more smaller problem instances $I_1, \dots, I_a$
  - These are called **subproblems**
  - Usually, want subproblems to be small compared to the size of I (e.g., half the size)
- **conquer:** For $1 \le j \le a$, solve instance I_j **recursively**, obtaining solutions $S_1, \dots, Sa$
- **combine:** Given solutions $S_1, \dots, Sa$, use an appropriate combining function to find the solution S to the problem instance I
  - i.e., S = Combine($S_1, \dots, Sa$).

3

D&C PROTO-ALGORITHM

```
DnC_template(I)
    if BaseCase(I) return Result(I)
    subproblems = [I_1, I_2, ..., I_a]
    subsolutions = []
    for j = 1..a
        subsolutions[j] = DnC_template(I_j)
    return Combine(subsolutions)
```

4

CORRECTNESS

```
DnC_template(I)
    if BaseCase(I) return Result(I)
    subproblems = [I_1, I_2, ..., I_a]
    subsolutions = []
    for j = 1..a
        subsolutions[j] = DnC_template(I_j)
    return Combine(subsolutions)
```

- Prove base cases are correct
- Inductively assume subproblems are solved correctly
- Show they are correctly assembled into a solution

5

RUNTIME/SPACE COMPLEXITY?

```
DnC_template(I)
    if BaseCase(I) return Result(I)
    subproblems = [I_1, I_2, ..., I_a]
    subsolutions = []
    for j = 1..a
        subsolutions[j] = DnC_template(I_j)
    return Combine(subsolutions)
```

- Techniques covered in this lecture
  - Model complexities using recurrence relations
  - Solve with substitution, master theorem, etc.

6

WORKED EXAMPLE: DESIGN OF MERGESORT

Here, a problem instance consists of an array A of n integers, which we want to sort in increasing order. The size of the problem instance is n.

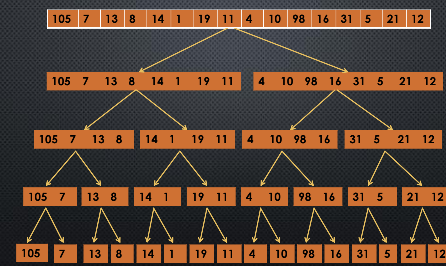
divide: Split A into two subarrays: A_L consists of the first $\lceil \frac{n}{2} \rceil$ elements in A and A_R consists of the last $\lfloor \frac{n}{2} \rfloor$ elements in A.

conquer: Run *Mergesort* on A_L and A_R.

combine: After A_L and A_R have been sorted, use a function *Merge* to merge A_L and A_R into a single sorted array. Recall that this can be done in time $\Theta(n)$ with a single pass through A_L and A_R. We simply keep track of the "current" element of A_L and A_R, always copying the smaller one into the sorted array.

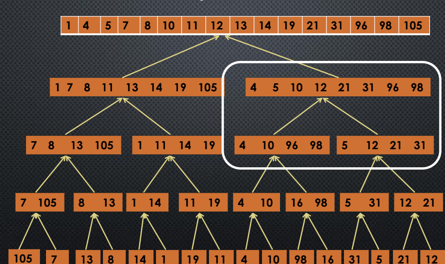
7

DIVIDE



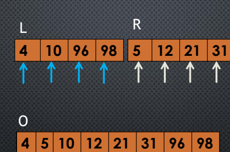
8

MERGE: CONQUER AND COMBINE



9

MERGE SIMULATION



10

PSEUDOCODE FOR MERGESORT

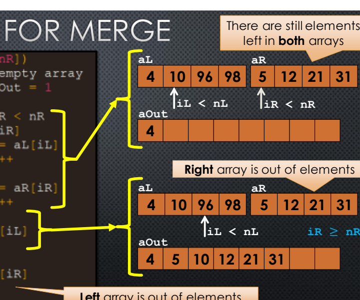
```
Mergesort(A[1..n])
      if n == 1 then return A
      nL = ceil(n/2)
      aL = A[1..nL]
      aR = A[(nL+1)..n]
      sL = Mergesort(aL)
      sR = Mergesort(aR)
      return Merge(sL, sR)
```

11

PSEUDOCODE FOR MERGE

```
Merge(aL[1..nL], aR[1..nR])
     aOut[1..(nL+nR)] = empty array
     iL = 1 ; iR = 1 ; iOut = 1

     while iL < nL and iR < nR
          if aL[iL] < aR[iR]
               aOut[iOut] = aL[iL]
               iL++ ; iOut++
          else
               aOut[iOut] = aR[iR]
               iR++ ; iOut++
     while iL < nL
          aOut[iOut] = aL[iL]
          iL++ ; iOut++
     while iR < nR
          aOut[iOut] = aR[iR]
          iR++ ; iOut++
     return aOut
```



12

ANALYSIS OF MERGESORT

```
Mergesort(A[1..n])
    if n == 1 then return A
    nL = ceil(n/2)
    aL = A[1..nL]
    aR = A[(nL+1)..n]
    sL = Mergesort(aL)
    sR = Mergesort(aR)
    return Merge(sL, sR)
```

Annotations: O(1) (line 1); O(1) (nL = ceil(n/2)); O(n) or O(1) (aL, aR); ??? (sL, sR); O(n) (return Merge)

So, MergeSort(A) takes **O(n)** time, **plus** the time for its **two recursive calls**!

How can we **analyze** this **recursive** program structure?

13

$Hulk(n) = Face - Chin + Hulk(n-1)$

RECURRENCE RELATIONS

A crucial analysis tool for recursive algorithms

14

RECURRENCE RELATIONS

Suppose $a_1, a_2, \ldots,$ is an infinite sequence of real numbers.

A recurrence relation is a formula that expresses a general term a_n in terms of one or more previous terms $a_1, \ldots, a_{n-1}$.

A recurrence relation will also specify one or more initial values starting at a_1.

Solving a recurrence relation means finding a formula for a_n that does not involve any previous terms $a_1, \ldots, a_{n-1}$.

There are many methods of solving recurrence relations. Two important methods are guess-and-check and the recursion tree method.

15

MATHEMATICALLY EXPRESSING THE COMPLEXITY OF MERGESORT

Let $T(n)$ denote the time to run *Mergesort* on an array of length n.

divide takes time $\Theta(1)$

conquer takes time $T\left(\lceil \frac{n}{2} \rceil\right) + T\left(\lfloor \frac{n}{2} \rfloor\right)$

combine takes time $\Theta(n)$

$T(n)$ is a function of $T(\ldots)$ so T is a **recurrence relation**

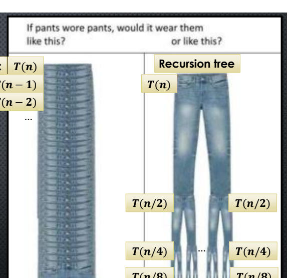
How can we compute/solve for $T(n)$?

Recurrence relation:

$$T(n) = \begin{cases} T\left(\lceil \frac{n}{2} \rceil\right) + T\left(\lfloor \frac{n}{2} \rfloor\right) + \Theta(n) & \text{if } n > 1 \\ \Theta(1) & \text{if } n = 1. \end{cases}$$

To make this easier, assume $n = 2^k$, which lets us ignore floors/ceilings

16

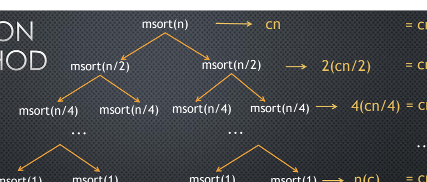


RECURSION TREE METHOD

Evaluating recurrences with $T(n/c)$ terms

17

RECURSION TREE METHOD



Level	# of nodes	runtime per node	total runtime for level
0	1	cn	cn
1	2	$c(n/2)$	$2c(n/2) = cn$
2	4	$c(n/4)$	$4c(n/4) = cn$
…	…	…	…
$\log n$	n	$c(n/n) = c$	$nc(n/n) = cn$

Can also compute using a table…

Total = cn * #levels

Total = $cn \log_2(n)$

So, mergesort has runtime $O(n \log n)$

18

RECURSION TREE METHOD FORMALIZED

Sample recurrence for **two** recursive calls on problem size $n/2$

$$T(n) = \begin{cases} 2T\left(\frac{n}{2}\right) + cn & \text{if } n > 1 \text{ is a power of 2} \\ d & \text{if } n = 1, \end{cases}$$

where c and d are constants.

We can solve this recurrence relation when n is a power of two, by constructing a recursion tree, as follows:

Step 1 Start with a one-node tree, say N, having the value $T(n)$.

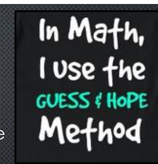
Step 2 Grow two children of N. These children, say N_1 and N_2, have the value $T(n/2)$, and the value of N is replaced by cn.

Step 3 Repeat this process recursively, terminating when a node receives the value $T(1) = d$.

Step 4 Sum the values on each level of the tree, and then compute the sum of all these sums; the result is $T(n)$.

19

GUESS-AND-CHECK METHOD



- Suppose we have the following recurrence
  $T(0) = 4$; $T(n) = T(n-1) + 6n - 5$
- **Guess** the form of the solution **any** way you like
- My approach: **the substitution method**
  - Recursively substitute the formula into itself
  - Try to identify patterns to **guess** the final closed form
- **Prove** that the guess was correct

20

SUBSTITUTION METHOD: WORKED EXAMPLE

Recurrence: $T(0) = 4$; $T(n) = T(n-1) + 6n - 5$

- $T(n-1) = T((n-1)-1) + 6(n-1) - 5$ — Compare: new terms? +(6n-5) -6
- $T(n) = (T(n-2) + 6(n-1) - 5) + 6n - 5$ **(substitute)**
- $= T(n-2) + 2(6n-5) - 6$ (try to preserve structure)
- $= (T(n-3) + 6(n-2) - 5) + 2(6n-5) - 6$ **(substitute)**
- $= T(n-3) + 3(6n-5) - 6(1+2)$ ← new terms? +(6n-5) -2(6)
- … identify patterns and **guess** what happens in the limit
- $= T(0) + n(6n-5) - 6(1+2+3+\cdots+(n-1)) = guess(n)$

21

- $guess(n) = T(0) + n(6n-5) - 6(1+2+3+\cdots+(n-1))$
- Use $1+2+\cdots+(n-1) = \frac{n(n-1)}{2}$
- $guess(n) = 4 + 6n^2 - 5n - 6n(n-1)/2$ **(simplify)**
- $= 3n^2 - 2n + 4$
- Are we done?
- The form of $guess(n)$ was an **educated guess**.
- To be sure, we must **prove** it correct using **induction**

22

PROOF

- **Recall:** $T(0) = 4$; $T(n) = T(n-1) + 6n - 5$; $guess(n) = 3n^2 - 2n + 4$
- **Want to prove:** $guess(n) = T(n)$ **for all** n
- Base case: $guess(0) = 3(0)^2 - 2(0) + 4 = T(0)$

23

PROOF

- **Recall:** $T(0) = 4$; $T(n) = T(n-1) + 6n - 5$; $guess(n) = 3n^2 - 2n + 4$
- **Want to prove:** $guess(n) = T(n)$ **for all** n
- Inductive case: **suppose** $guess(n) = T(n)$ for $n \geq 0$, **show** $guess(n+1) = T(n+1)$.
- $T(n+1) = T(n) + 6(n+1) - 5$ (by definition)
- $= guess(n) + 6(n+1) - 5$ (by inductive hypothesis)
- $= 3n^2 - 2n + 4 + 6(n+1) - 5$ (substitute)
- $= 3n^2 + 4n + 5$ (simplify)
- $guess(n+1) = 3(n+1)^2 - 2(n+1) + 4$ (by definition)
- $= 3n^2 + 4n + 5 = T(n+1)$ (simplify)

24

ANOTHER APPROACH

- Suppose you look for a while at the previous recurrence:
  - $T(0) = 4\,; T(n) = T(n-1) + 6n - 5$
- With some experience, you might just **guess** it's **quadratic**
- If you're right, it should have the form:
  - $an^2 + bn + c$ for some **unknown** constants **a, b, c**
- So, just carry the unknown constants **into the proof**!
  - You can then determine what the constants **must be** for the proof to work out

25

- $T(0) = 4\,; T(n) = T(n-1) + 6n - 5\,; \boldsymbol{guess(n) = an^2 + bn + c}$
- **Want to prove:** $\boldsymbol{guess(n) = T(n)}$ **for all** $\boldsymbol{n}$
- Base case: $guess(0) = a(0)^2 + b(0) + c = T(0) = 4$
- this holds **iff** $\boldsymbol{c = 4}$ $\quad (a, b$ are not constrained)
- Inductive case: **suppose** $guess(n) = T(n)$ for $n \geq 0$, **show** $guess(n+1) = T(n+1)$.
- $\boldsymbol{T(n+1)} = T(n) + 6(n+1) - 5$ (by definition)
- $= \boldsymbol{guess(n)} + 6(n+1) - 5$ (by inductive hypothesis)
- $= an^2 + bn + \boldsymbol{4} + 6(n+1) - 5$ (substitute)
- $= an^2 + (b+6)n + 5$ (simplify)

26

- **Recall:** $\boldsymbol{guess(n) = an^2 + bn + c}$ where $\boldsymbol{c = 4}$
- Inductive case: **suppose** $guess(n) = T(n)$ for $n \geq 0$, **show** $guess(n+1) = T(n+1)$.
- $\boldsymbol{T(n+1)} = an^2 + (b+6)n + 5$ (continue previous slide)
- $\boldsymbol{guess(n+1)} = a(n+1)^2 + b(n+1) + \boldsymbol{4}$ (by definition and $\boldsymbol{c = 4}$)
- $= a(n^2 + 2n + 1) + bn + b + 4$ (simplify, and...)
- $= an^2 + (2a+b)n + (a+b+4)$ (rearrange polynomial)
- We want this to be equal to $\boldsymbol{T(n+1)}$
  - $an^2 + (2a+b)n + (a+b+4) = an^2 + (b+6)n + 5$
  - equivalent to $(2a+b) = (b+6)$ and $(a+b+4) = 5$
  - first implies $\boldsymbol{a = 3}$ $\quad$ plug a into second to get $b = 5 - 4 - 3 = -2$

So, inductive hypothesis is **correct** for **a = 3, b = −2, c = 4**

27

MASTER THEOREM FOR RECURRENCES

- Provides a formula for solving many recurrence relations
- We start with a **simplified version**
- Consider recurrence: $T(1) = d \quad ; \quad T(n) = aT\left(\frac{n}{b}\right) + \Theta(n^y)$ where $a \geq 1, b \geq 2$ and n is a power of b (i.e., $n = b^j$ for integer j)

Example corresponding algorithm

```
    if BaseCase(I) return Result(I)

    subsolutions = []
    for j = 1..a
        let s = subproblem of size n/b
        subsolutions[j] = DnC_algo(s)

    solution = combine in n^y time
    return solution
```

Simplified Master Theorem

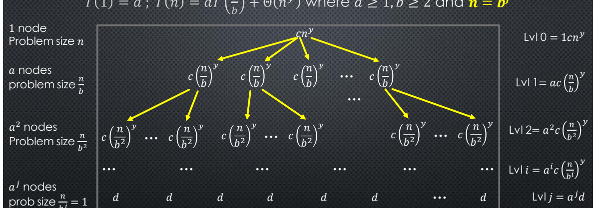
$$T(n) \in \begin{cases} \Theta(n^x) & \text{if } y < x \\ \Theta(n^x \log n) & \text{if } y = x \\ \Theta(n^y) & \text{if } y > x. \end{cases}$$

where $x = \log_b a$.

28

DERIVING THE SIMPLIFIED MASTER THEOREM

$T(1) = d\,; T(n) = aT\left(\frac{n}{b}\right) + \Theta(n^y)$ where $a \geq 1, b \geq 2$ and $\boldsymbol{n = b^j}$



Sum over all levels we get $T(n) = da^j + \sum_{i=0}^{j-1} ca^i \left(\frac{n}{b^i}\right)^y$

Let's rearrange this into a **geometric sequence** and solve

29

REARRANGING

- $T(n) = da^j + \sum_{i=0}^{j-1} ca^i \left(\frac{n}{b^i}\right)^y$
- $= da^j + \sum_{i=0}^{j-1} ca^i \frac{n^y}{(b^i)^y}$
- $= da^j + \sum_{i=0}^{j-1} ca^i \frac{n^y}{(b^y)^i}$
- $= da^j + \sum_{i=0}^{j-1} cn^y \frac{a^i}{(b^y)^i}$
- $= da^j + \sum_{i=0}^{j-1} cn^y \left(\frac{a}{b^y}\right)^i$
- $= da^j + cn^y \sum_{i=0}^{j-1} \left(\frac{a}{b^y}\right)^i$

- Let $x = \log_b a$
- x relates # of subproblems to their size
- Rearranging we have $b^x = a$
- So $T(n) = da^j + cn^y \sum_{i=0}^{j-1} \left(\frac{b^x}{b^y}\right)^i$
- $= da^j + cn^y \sum_{i=0}^{j-1} (b^{x-y})^i$
- Also $da^j = d(b^x)^j = d(b^j)^x$
- Since $n = b^j$ this is just dn^x
- So $\boldsymbol{T(n) = dn^x + cn^y \sum_{i=0}^{j-1} (b^{x-y})^i}$
- **and we can simplify: let** $\boldsymbol{r = b^{x-y}}$

30

SOLVING THE GEOMETRIC SEQ

- $T(n) = dn^x + cn^y \sum_{i=0}^{j-1} r^i$ where $r = b^{x-y}$
- **Geo. Seq. formula:** $\sum_{i=0}^{j-1} ar^i = \begin{cases} a\frac{r^j-1}{r-1} \in \Theta(r^j) & \text{if } r > 1 \\ ja \in \Theta(j) & \text{if } r = 1 \\ a\frac{1-r^j}{1-r} \in \Theta(1) & \text{if } 0 < r < 1 \end{cases}$
- So different solutions depending on r
  - Case 1: $r = b^{x-y} > 1 \iff x - y > 0 \iff x > y$
  - Case 2: $r = b^{x-y} = 1 \iff x - y = 0 \iff x = y$
  - Case 3: $0 < r = b^{x-y} < 1 \iff x - y < 0 \iff x < y$

31

SOLVING THE GEOMETRIC SEQ

- Formula: $\sum_{i=0}^{j-1} ar^i = \begin{cases} a\frac{r^j-1}{r-1} \in \Theta(r^j) & \text{if } r > 1 \\ ja \in \Theta(j) & \text{if } r = 1 \\ a\frac{1-r^j}{1-r} \in \Theta(1) & \text{if } 0 < r < 1 \end{cases}$
- **Case 1:** $r = b^{x-y} > 1 \iff x - y > 0 \iff x > y$
- $T(n) = dn^x + cn^y \sum_{i=0}^{j-1} r^i \in dn^x + cn^y \Theta(r^j)$
- $T(n) \in \Theta(n^x + n^y r^j) = \Theta(n^x + n^y (b^{x-y})^j) = \Theta\left(n^x + n^y (b^j)^{x-y}\right)$
- Recall $b^j = n$, so $T(n) \in \Theta(n^x + n^y n^{x-y}) = \Theta(n^x + n^{y+x-y})$
- So $T(n) \in \Theta(n^x)$

32

SOLVING THE GEOMETRIC SEQ

- Formula: $\sum_{i=0}^{j-1} ar^i = \begin{cases} a\frac{r^j-1}{r-1} \in \Theta(r^j) & \text{if } r > 1 \\ ja \in \Theta(j) & \text{if } r = 1 \\ a\frac{1-r^j}{1-r} \in \Theta(1) & \text{if } 0 < r < 1 \end{cases}$
- **Case 2:** $r = b^{x-y} = 1 \iff x - y = 0 \iff x = y$
- $T(n) = dn^x + cn^y \sum_{i=0}^{j-1} r^i \in dn^x + cn^y \Theta(j)$
- $T(n) \in \Theta(n^x + jn^y) = \Theta(n^x + jn^x)$ since $x = y$
- Recall $b^j = n$, so $\log_b b^j = \log_b n$. This means $j \in \Theta(\log n)$.
- So $T(n) = \Theta(n^x + n^x \log n) = \Theta(n^x \log n)$

33

SOLVING THE GEOMETRIC SEQ

- Formula: $\sum_{i=0}^{j-1} ar^i = \begin{cases} a\frac{r^j-1}{r-1} \in \Theta(r^j) & \text{if } r > 1 \\ ja \in \Theta(j) & \text{if } r = 1 \\ a\frac{1-r^j}{1-r} \in \Theta(1) & \text{if } 0 < r < 1 \end{cases}$
- **Case 3:** $0 < r = b^{x-y} < 1 \iff x - y < 0 \iff x < y$
- $T(n) = dn^x + cn^y \sum_{i=0}^{j-1} r^i \in dn^x + cn^y \Theta(1)$
- $T(n) \in \Theta(n^x + n^y)$
- Since $x < y$, we simply have $T(n) \in \Theta(n^y)$

34

MASTER THEOREM FOR RECURRENCES

- Simplified version

Consider recurrence:
$T(n) = aT\left(\frac{n}{b}\right) + \Theta(n^y)$ where $a \geq 1, b \geq 2$ and $n = b^j$
And let $x = \log_b a$.

$$T(n) \in \begin{cases} \Theta(n^x) & \text{if } y < x \\ \Theta(n^x \log n) & \text{if } y = x \\ \Theta(n^y) & \text{if } y > x. \end{cases}$$

35

SOME BONUS INTUITION FOR R CASES

Recall: $T(n) = dn^x + cn^y \sum_{i=0}^{j-1} r^i$ where $r = b^{x-y}$
$x = \log_b a$ i.e. $\log_{\text{subproblem size}} |\text{subproblems}|$

case	r	y, x	complexity of $T(n)$
heavy leaves	$r > 1$	$y < x$	$T(n) \in \Theta(n^x)$
balanced	$r = 1$	$y = x$	$T(n) \in \Theta(n^x \log n)$
heavy top	$r < 1$	$y > x$	$T(n) \in \Theta(n^y)$

heavy leaves means that the value of the recursion tree is dominated by the values of the leaf nodes.

balanced means that the values of the levels of the recursion tree are constant (except for the last level).

heavy top means that the value of the recursion tree is dominated by the value of the root node.

36

WORKED EXAMPLES

Recall: simplified master theorem

Suppose that $a \geq 1$ and $b > 1$. Consider the recurrence

$T(n) = aT\left(\frac{n}{b}\right) + \Theta(n^y)$, *where n is a power of b.*

Denote $x = \log_b a$. Then

$$T(n) \in \begin{cases} \Theta(n^x) & \text{if } y < x \\ \Theta(n^x \log n) & \text{if } y = x \\ \Theta(n^y) & \text{if } y > x. \end{cases}$$

Questions: a=? b=? y=? x=?
which Θ function?

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

a=2; b=2; y=1; x=1
$\Theta(n^x \log n) = \Theta(n \log n)$

$T(n) = 3T(n/2) + cn.$

a=3; b=2; y=1; x=$\log_2 3$
$\Theta(n^x) = \Theta(n^{\log_2 3})$

$T(n) = 4T(n/2) + cn.$

a=4; b=2; y=1; x=$\log_2 4$
$\Theta(n^x) = \Theta(n^2)$

$T(n) = 2T(n/2) + cn^{3/2}.$

a=2; b=2; y=3/2; x=1
$\Theta(n^y) = \Theta(n^{3/2})$

37

MASTER THEOREM WHEN $b^{j-1} < n < b^j$

Bonus slide, for you at home

- n/b is **not always an integer!**
  - floors/ceilings are hard
  - not a geometric sequence
- Suppose we get a **big-O** bound for $b^{j-1} < n < b^j$ by instead considering the **larger problem size b^j**
  - So $T(n) \leq T(b^j) \in \begin{cases} \Theta\left(\left(b^j\right)^x\right) & \text{if } y < x \\ \Theta\left(\left(b^j\right)^x \log b^j\right) & \text{if } y = x \\ \Theta\left(\left(b^j\right)^y\right) & \text{if } y > x \end{cases}$

38

MASTER THEOREM WHEN $b^{j-1} < n < b^j$

Bonus slide, for you at home

- $T(n) \leq T(b^j) \in \begin{cases} \Theta\left(\left(b^j\right)^x\right) & \text{if } y < x \\ \Theta\left(\left(b^j\right)^x \log b^j\right) & \text{if } y = x \\ \Theta\left(\left(b^j\right)^y\right) & \text{if } y > x \end{cases}$
- **Observation: $b^j < bn$** since n is between b^{j-1} and b^j
- So $T(n) \leq T(b^j) \in \begin{cases} \Theta((bn)^x) & \text{if } y < x \\ \Theta((bn)^x \log bn) & \text{if } y = x \\ \Theta((bn)^y) & \text{if } y > x \end{cases}$

39

MASTER THEOREM WHEN $b^{j-1} < n < b^j$

Bonus slide, for you at home

- $T(n) \in \begin{cases} \Theta((bn)^x) & \text{if } y < x \\ \Theta((bn)^x \log bn) & \text{if } y = x \\ \Theta((bn)^y) & \text{if } y > x \end{cases}$
- **Case 1** $(y < x)$: $(bn)^x = \mathbf{b^x} n^x$ and $\mathbf{b^x}$ is a **constant**
  - So $T(n) \in O(n^x)$
- **Case 2** $(y = x)$: $(bn)^x \log bn = b^x n^x (\log b + \log n)$
  - $T(bn) \in \Theta(\mathbf{b^x} n^x \mathbf{\log b} + \mathbf{b^x} n^x \log n) = \Theta(n^x + n^x \log n)$
  - So $T(n) \in O(n^x \log n)$
- **Case 3** $(y > x)$: $(bn)^y = \mathbf{b^y} n^y$
  - So $T(n) \in O(n^y)$

Can tackle Ω similarly to get θ

40

GENERAL MASTER THEOREM

Example recurrence: $T(n) = 3T(n/4) + n \log n$

Suppose that $a \geq 1$ and $b > 1$. Consider the recurrence

$$T(n) = aT\left(\frac{n}{b}\right) + f(n),$$

Arbitrary function of n (not just cn^y)

where n is a power of b. Denote $x = \log_b a$. Then

$$T(n) \in \begin{cases} \Theta(n^x) & \text{if } f(n) \in O(n^{x-\epsilon}) \text{ for some } \epsilon > 0 \\ \Theta(n^x \log n) & \text{if } f(n) \in \Theta(n^x) \\ \Theta(f(n)) & \text{if } f(n)/n^{x+\epsilon} \text{ is an increasing function of } n \text{ for some } \epsilon > 0. \end{cases}$$

Must reason about relationship between $f(n)$ and n^x

41

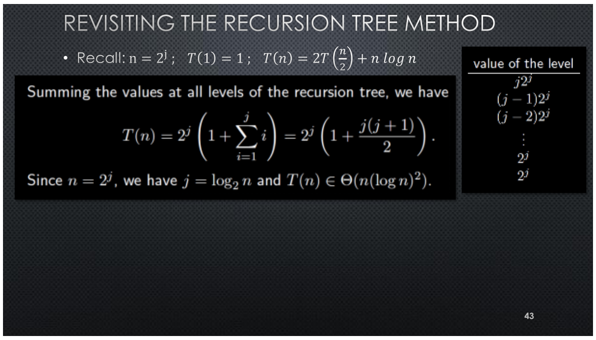
REVISITING THE RECURSION TREE METHOD

- Some recurrences with complex f(n) functions (such as f(n) = log n) can still be solved "by hand"
- Example: Let $n = 2^j$; $T(1) = 1$; $T(n) = 2T\left(\frac{n}{2}\right) + n \log n$

level	# nodes	value at each node	value of the level
j	1	$j2^j$	$j2^j$
$j-1$	2	$(j-1)2^{j-1}$	$(j-1)2^j$
$j-2$	2^2	$(j-2)2^{j-2}$	$(j-2)2^j$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
1	2^{j-1}	2^1	2^j
0	2^j	1	2^j

Note
$\log_2 n = j$
So
$j2^j = n \log_2 n$
And
$(j-1)2^{j-1} = \frac{n}{2} \log \frac{n}{2}$

42


REVISITING THE RECURSION TREE METHOD
• Recall: $n = 2^j$; $T(1) = 1$; $T(n) = 2T\left(\frac{n}{2}\right) + n \log n$
Summing the values at all levels of the recursion tree, we have
$T(n) = 2^j \left(1 + \sum_{i=1}^{j} i\right) = 2^j \left(1 + \frac{j(j+1)}{2}\right)$.
Since $n = 2^j$, we have $j = \log_2 n$ and $T(n) \in \Theta(n(\log n)^2)$.
value of the level
$j2^j$
$(j-1)2^j$
$(j-2)2^j$
$\vdots$
2^j
2^j
43