Algorithms

Arthur Hoskey, Ph.D.
Farmingdale State College
Computer Systems Department

Today's Lecture

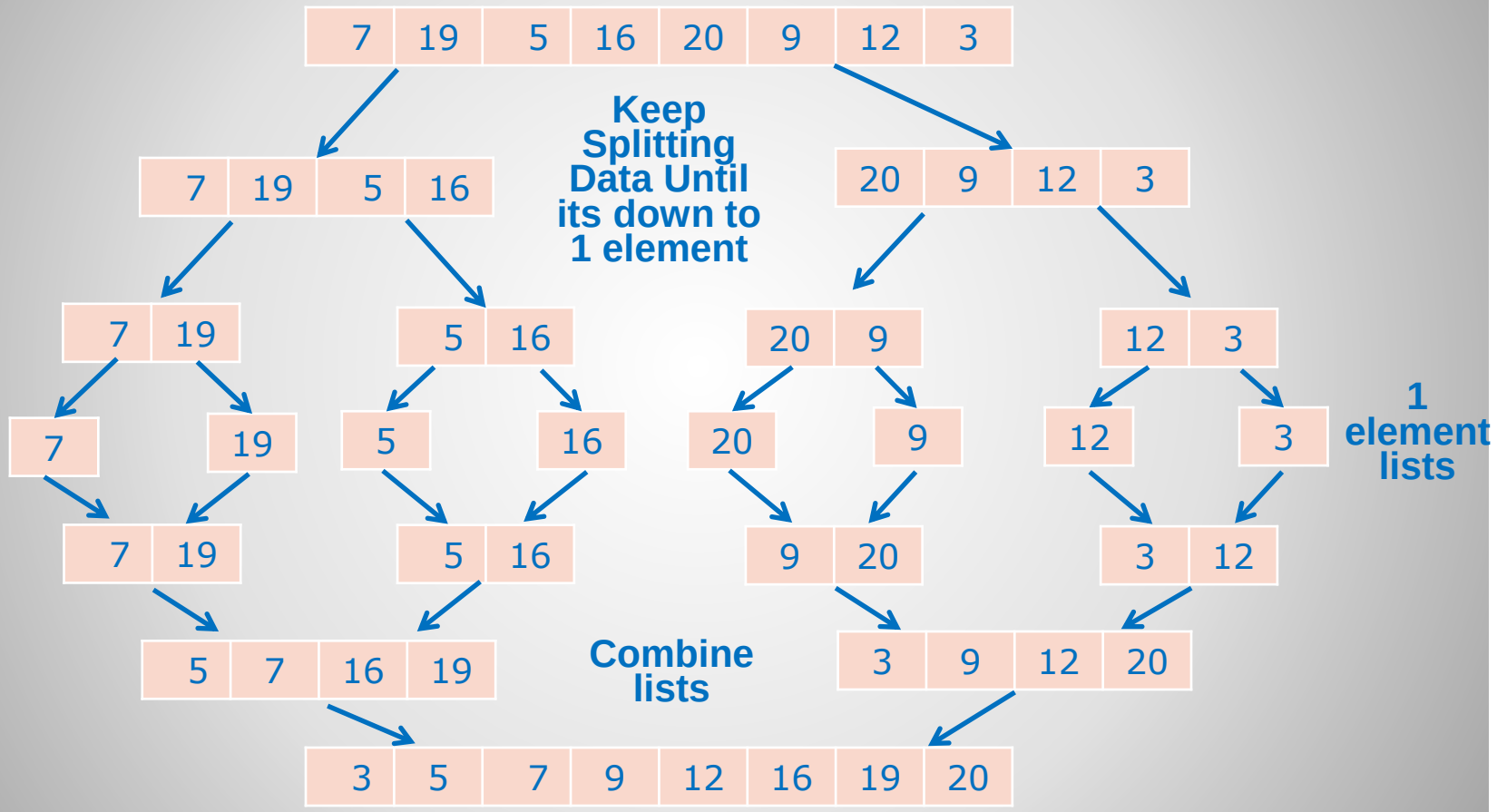
- Merge Sort Overview
- Master Theorem

Merge Sort Overview

- Merge sort is a faster sorting algorithm than insertion sort, selection sort, or bubble sort.
- The general idea is as follows:
  - Keep splitting the list into smaller lists and sort the smaller lists individually.
  - Combine the smaller sorted lists to get the whole sorted list.
- This is an example of a "Divide and Conquer" algorithm.

Merge Sort

- Keep splitting elements until it is a list of 1.
- Combine lists in sorted order.



Merge Sort

Merge Sort Pseudocode

Mergesort(List L)
If list contains 1 element return (already sorted)
Divide list L into two equal size lists (L1 and L2)
SortedL1 = Mergesort(L1)
SortedL2 = Mergesort(L2)
SortedL = Merge(SortedL1, SortedL2)
Return SortedL

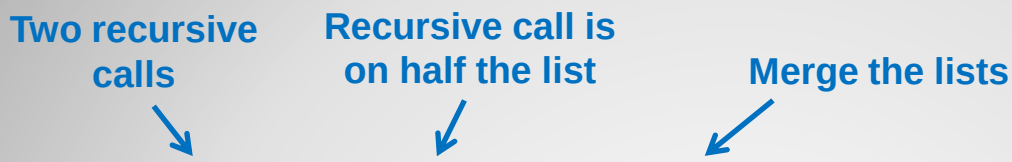
Mergesort returns a sorted list

Combine the two sorted lists into one sorted list

Merge(List L1, List L2)
Loop and combine two lists. Always take the lowest element from each list.
Return merged list

Merge Sort Pseudocode

Merge Sort Recurrence



- $T(n) = 2 * T(n/2) + \Theta(n)$

- There are two recursive calls to merge sort on the half size lists (2 * T(n/2).
- The $\Theta(n)$ is the cost to combine the lists.

Divide list L into two equal size lists (L1 and L2)
SortedL1 = Mergesort(L1)
SortedL2 = Mergesort(L2)
SortedL = Merge(SortedL1, SortedL2)

2 recursive calls to merge sort each on half the list so 2*T(n/2)

Combine lists. This visits each item once so $\Theta(n)$.

Merge Sort Recurrence

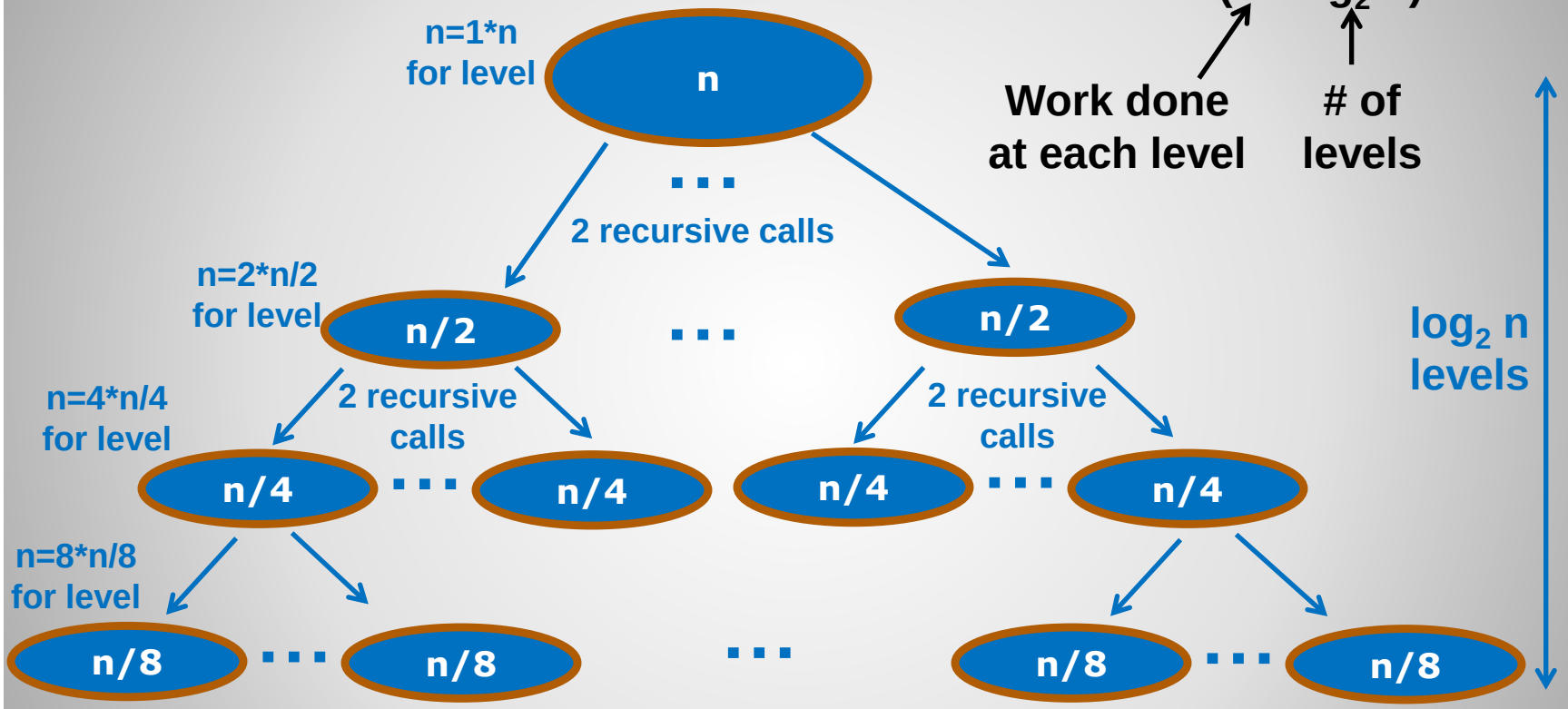
Merge Sort

- $T(n) = 2 * T(n/2) + \Theta(n)$

n work done at each level of tree.
There are $\log_2 n$ levels.
Cost is $O(n * \log_2 n)$

Work done at each level

# of levels



Note: There are actually $(\log_2 n + 1)$ levels, but we can ignore the +1

Merge Sort

Master Theorem

- Easier way to find asymptotic bounds of a recurrence relation.
- Only works on some types of recurrences (divide and conquer).
- Allows you to plug in values and figure out the runtime (assuming the recurrence has the proper form).
- Using the master theorem follows the form of the previous example with a few extra calculations.

Master Theorem – Divide and Conquer

Master Theorem

- The master theorem takes a recurrence (of the correct form) as input and returns a runtime.

Master Theorem – Divide and Conquer

Master Theorem Requirements

- Using the master theorem requires that we have a recurrence in the following form:

Must divide into subproblems

Work done outside must be a polynomial

- $T(1) = O(1)$
- $T(n) = a * T(n/b) + O(n^d)$

Work done in recursive call

Work done outside recursion

Note: b must be greater than 1 so that we are making the problem smaller in the recursive calls

$a>=1, b>1$

Description of Pieces

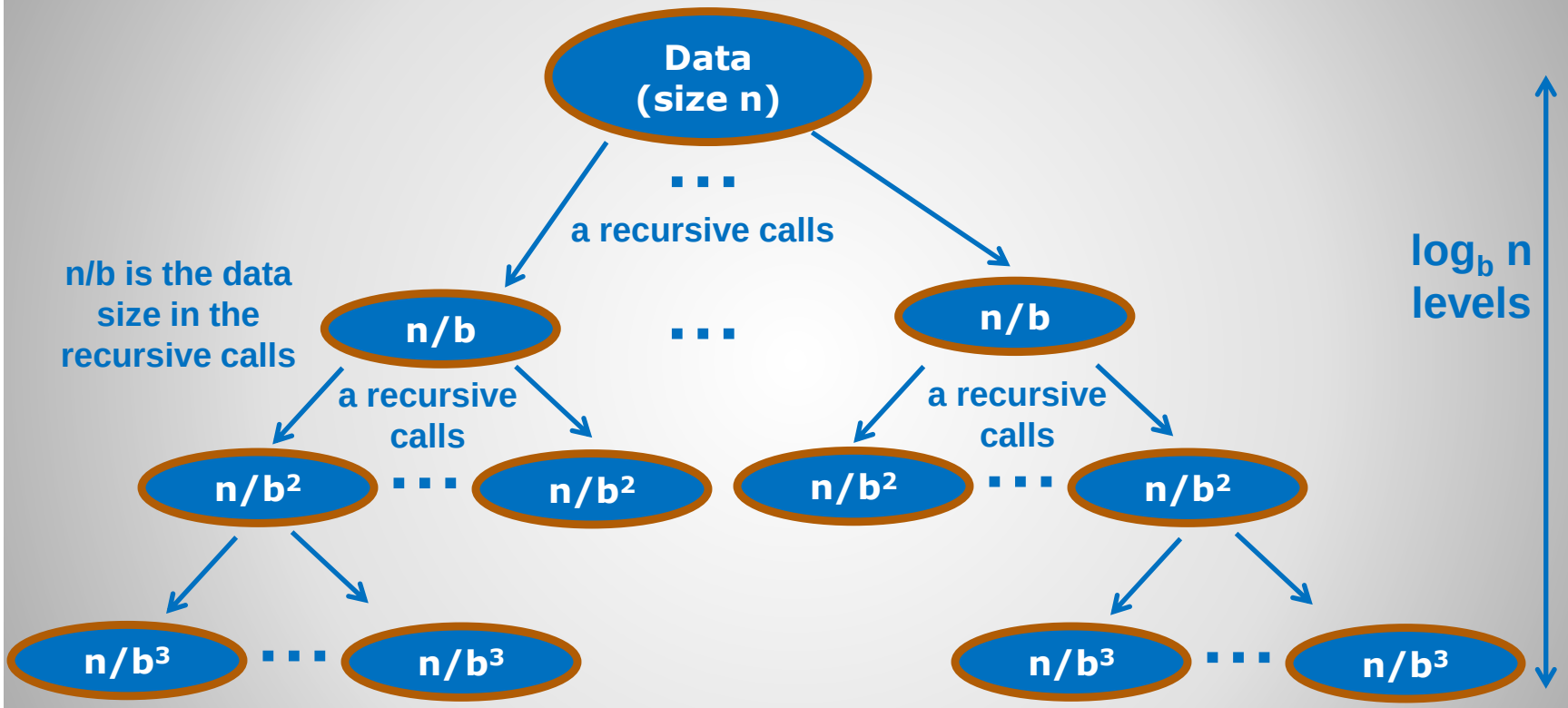
Note: All subproblems must have the same size

- a - Number of subproblems (a >= 1)
- b – Factor for dividing problem (b > 1)
- $O(n^d)$ – Non-recursive piece of function.

Master Theorem – Divide and Conquer

Master Theorem

- $T(n) = a * T(n/b) + O(n^d)$



Master Theorem – Divide and Conquer

Master Theorem Procedure

1. Check if recurrence is in correct form.
2. Calculate a, b, and d.
3. Use master theorem rules to determine the runtime.

Must divide into subproblems in recursive call

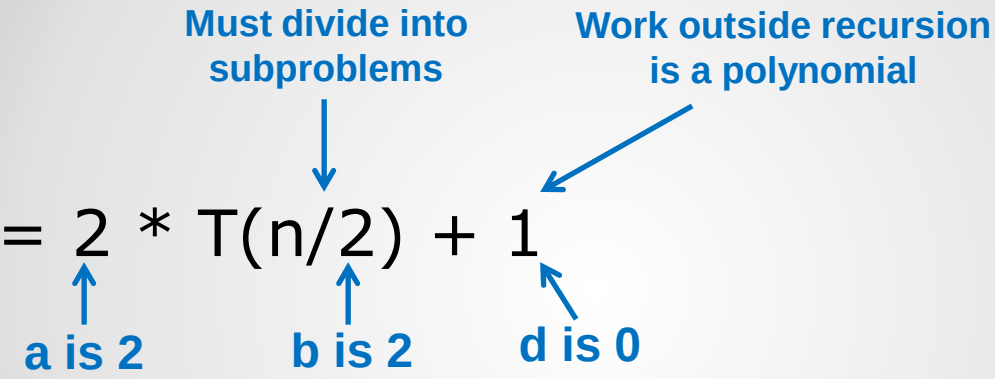
- $T(n) = a * T(n/b) + O(n^d)$

$a>=1, b>1$

Master Theorem – Divide and Conquer

Calculate a, b, d

- $T(n) = a * T(n/b) + O(n^d)$



Master Theorem – Divide and Conquer

Master Theorem Rules (Overview)

- When you have the values of a, b, and d you can check the cases.

Here is an overview of the main cases:

- Case 1: If ($a > b^d$) then $O(n^{\log_b a})$ **Most work is done at root**
- Case 2: If ($a = b^d$) then $O(n^d \log n)$ **Even amount of work done at root and leaves**
- Case 3: If ($a < b^d$) then $O(n^d)$ **Most work is done at leaves**

Master Theorem – Divide and Conquer

Exercise

- $T(n) = 2 * T(n/2) + n$

Has required form for master theorem

Solution

Here is the general form: $T(n) = a * T(n/b) + f(n)$

$T(n) = 2 * T(n/2) + O(n^1)$

a b d

1. Calculate a, b, d. a=2, b=2, d=1.
2. Check Cases. The following case works:

$a = b^d$

$2 = 2^1$

$2 = 2$ (true!)

$a = b^d$ case applies so the answer is in the form $O(n^d \log n)$

Answer

$O(n^d \log n) \rightarrow O(n^1 \log n) \rightarrow$ **$O(n \log n)$**

Master Theorem - Exercise

Exercise

- T(n) = 4 * T(n/2) + n

Master Theorem - Exercise

Exercise

- $T(n) = 4 * T(n/2) + n$

Has required form for master theorem

Solution

Here is the general form: $T(n) = a * T(n/b) + f(n)$

$T(n) = 4 * T(n/2) + O(n^1)$

a b d

1. Calculate a, b, d. a=4, b=2, d=1.
2. Check Cases. The following case works:

$a > b^d$

$4 > 2^1$

$4 > 2$ (true!)

$a > b^d$ case applies so the answer is in the form $O(n^{\log_b a})$

Answer

$O(n^{\log_b a}) \rightarrow O(n^{\log_2 4}) \rightarrow \mathbf{O(n^2)}$

Master Theorem - Exercise

Exercise

- $T(n) = 4 * T(n/2) + n^2$

Master Theorem - Exercise

Exercise

- $T(n) = 4 * T(n/2) + n^2$

Has required form for master theorem

Solution

Here is the general form: $T(n) = a * T(n/b) + f(n)$

$T(n) = 4 * T(n/2) + O(n^2)$

a b d

1. Calculate a, b, d. a=4, b=2, d=2.
2. Check Cases. The following case works:

$a = b^d$

$4 = 2^2$

$4 = 4$ (true!)

$a = b^d$ case applies so the answer is in the form $O(n^d \log n)$

Answer

$O(n^d \log n) \rightarrow$ **$O(n^2 \log n)$**

Master Theorem - Exercise

Exercise

- $T(n) = T(n/2) + n^2$

Master Theorem - Exercise

Exercise

- $T(n) = T(n/2) + n^2$

Has required form for master theorem

Solution

Here is the general form: $T(n) = a * T(n/b) + f(n)$

$T(n) = 1 * T(n/2) + O(n^2)$

a b d

1. Calculate a, b, d. a=1, b=2, d=2.
2. Check Cases. The following case works:

$a < b^d$

$1 < 2^2$

$1 < 4$ (true!)

$a < b^d$ case applies so the answer is in the form $O(n^d)$

Answer

$O(n^d) \rightarrow$ **$O(n^2)$**

Master Theorem - Exercise

Exercise

- T(n) = T(n-1) + 1

Master Theorem - Exercise

Master Theorem - Exercise

Exercise

- T(n) = T(n-1) + 1

Does NOT have required form for master theorem

Solution

Here is the general form: T(n) = a *T(n/b) + f(n)

T(n) = T(n-1) + 1

Does not divide in the recursive step so master theorem does not apply. Need to use other methods to find the answer.

Master theorem only works on a divide and conquer recurrence (b must be > 1)

- **End of Slides**

End of Slides