Lecture

Recurrence Relations and how to solve them!

Last time....

- Sorting: InsertionSort and MergeSort
- What does it mean to work and be fast?
  - Worst-Case Analysis
  - Big-Oh Notation
- Analyzing running time of recursive algorithms
  - By writing out a tree and adding up all the work done.

Today

- Recurrence Relations!
  - How do we calculate the runtime a recursive algorithm?
- The Master Method
  - A useful theorem so we don't have to answer this question from scratch each time.
- The Substitution Method
  - A different way to solve recurrence relations, more general than the Master Method.

Running time of MergeSort

- Let T(n) be the running time of MergeSort on a length n array.
- We know that $T(n) = O(n\log(n))$.
- We also know that T(n) satisfies:

$$T(n) = 2 \cdot T\left(\frac{n}{2}\right) + O(n)$$

```
MERGESORT(A):
  n = length(A)
  if n ≤ 1:
    return A
  L = MERGESORT(A[:n/2])
  R = MERGESORT(A[n/2:])
  return MERGE(L,R)
```

Running time of MergeSort

- Let T(n) be the running time of MergeSort on a length n array.
- We know that T(n) = O(nlog(n)).
- We also know that T(n) satisfies:

$$T(n) \leq 2 \cdot T\left(\frac{n}{2}\right) + 11 \cdot n$$

Last time we showed that the time to run MERGE on a problem of size n is O(n). For concreteness, let's say that it's at most 11n operations.

```
MERGESORT(A):
  n = length(A)
  if n ≤ 1:
    return A
  L = MERGESORT(A[:n/2])
  R = MERGESORT(A[n/2:])
  return MERGE(L,R)
```

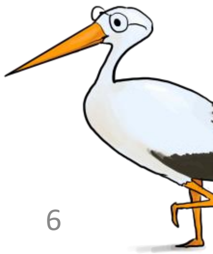
Recurrence Relations

- $T(n) = 2 \cdot T\left(\frac{n}{2}\right) + 11 \cdot n$ is a **recurrence relation.**
- It gives us a formula for T(n) in terms of T(less than n)
- The challenge:

Given a recurrence relation for T(n), find a closed form expression for T(n).

- For example, T(n) = O(nlog(n))

Note that $T(n) \leq 2 \cdot T\left(\frac{n}{2}\right) + 11 \cdot n$ (with a $\leq$) is also a recurrence relation! Does it matter for a conclusion like T(n) = O(n log(n))?



Technicalities I

Base Cases

- Formally, we should always have base cases with recurrence relations.
- $T(n) = 2 \cdot T\left(\frac{n}{2}\right) + 11 \cdot n$ with $T(1) = 1$

is not the same function as

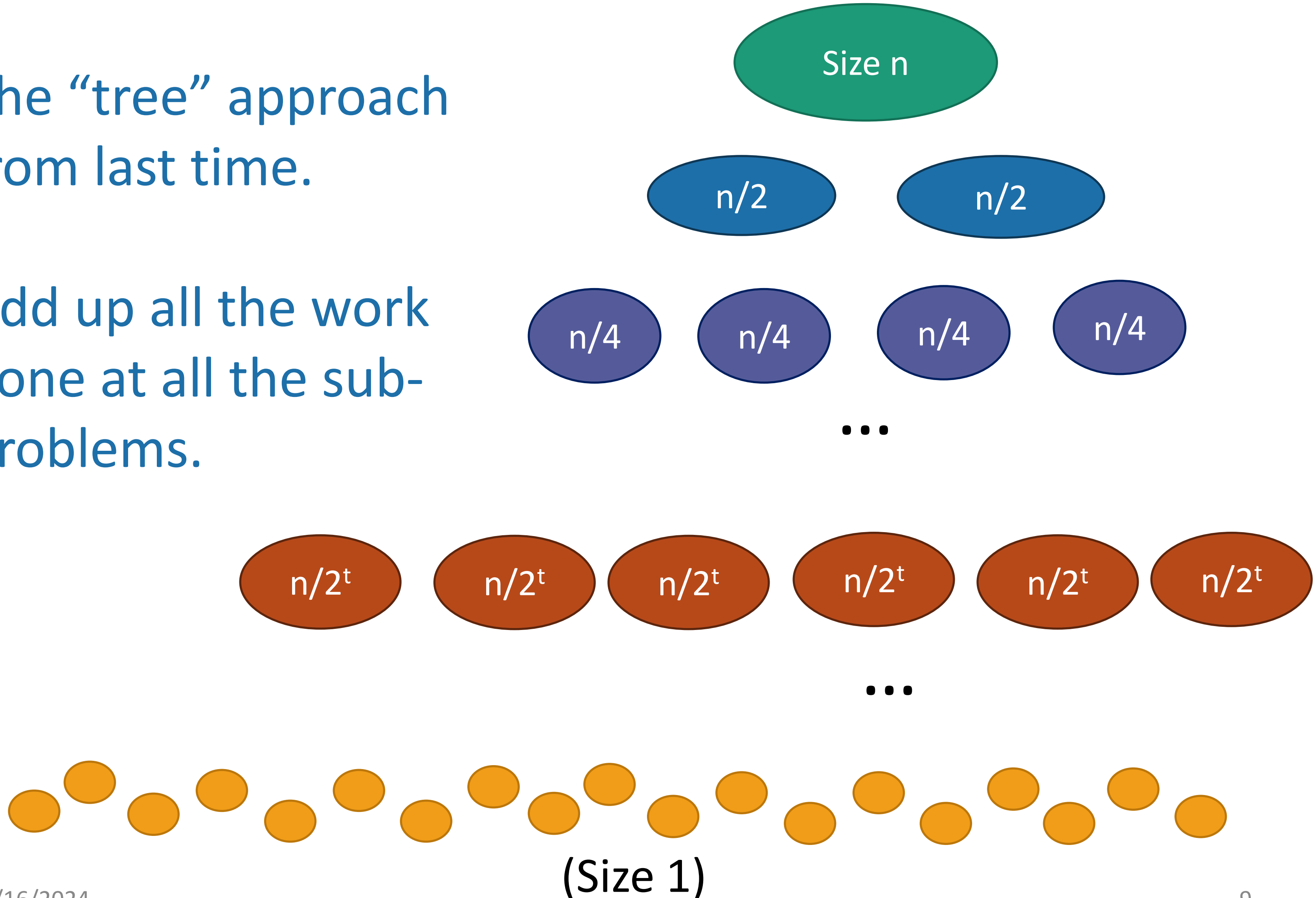
- $T(n) = 2 \cdot T\left(\frac{n}{2}\right) + 11 \cdot n$ with $T(1) = 1000000000$
- However, no matter what T is, T(1) is O(1), so sometimes we'll just omit it.

Why is T(1) = O(1)?

- You played around with these examples (when n is a power of 2):
  1. $T_1(n) = T_1\left(\frac{n}{2}\right) + n, \qquad T(1) = 1$
  2. $T(n) = 2 \cdot T\left(\frac{n}{2}\right) + n, \qquad T(1) = 1$
  3. $T_2(n) = 4 \cdot T_2\left(\frac{n}{2}\right) + n, \qquad T(1) = 1$

One approach for all of these

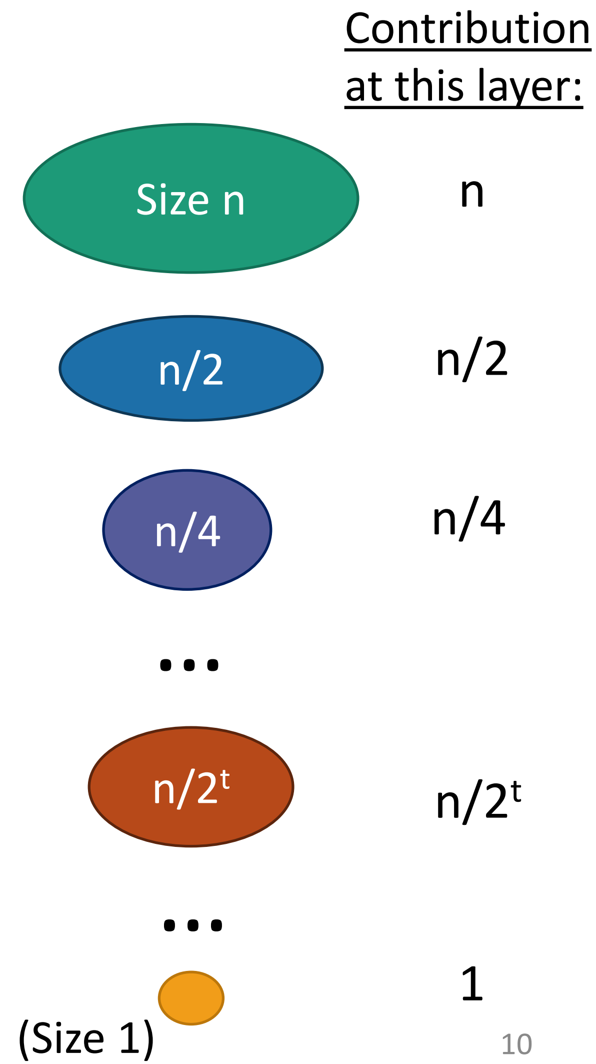
- The “tree” approach from last time.
- Add up all the work done at all the sub-problems.



- $T_1(n) = T_1\left(\frac{n}{2}\right) + n, \quad T_1(1) = 1.$
- Adding up over all layers:

$$\sum_{i=0}^{\log(n)} \frac{n}{2^i} = 2n - 1$$

- So $T_1(n) = O(n)$.

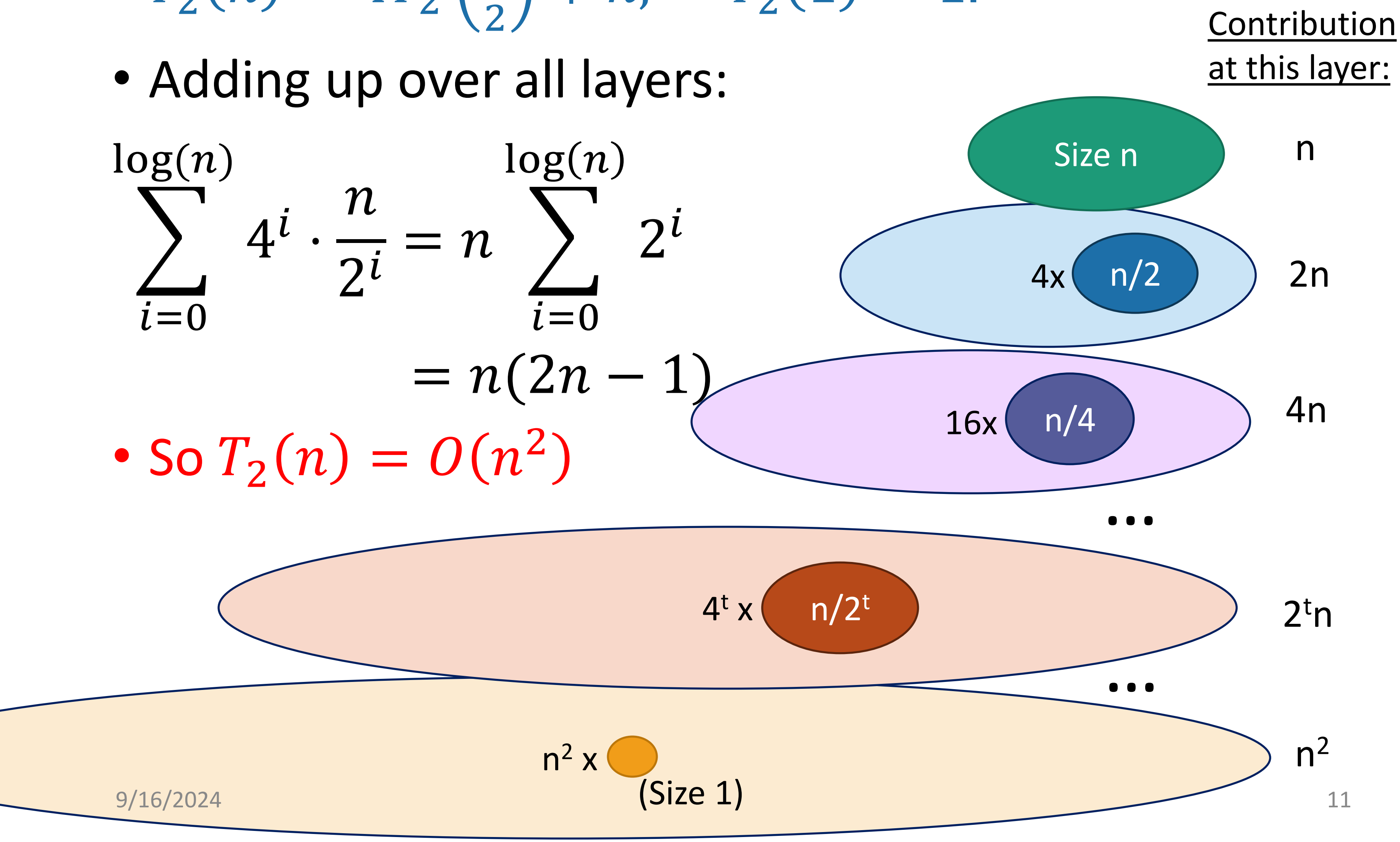


- $T_2(n) = 4T_2\left(\frac{n}{2}\right) + n, \quad T_2(1) = 1.$
- Adding up over all layers:

$$\sum_{i=0}^{\log(n)} 4^i \cdot \frac{n}{2^i} = n \sum_{i=0}^{\log(n)} 2^i$$

$$= n(2n - 1)$$

- So $T_2(n) = O(n^2)$



More examples

T(n) = time to solve a problem of size n.

- Needlessly recursive integer multiplication
- $T(n) = 4\,T(n/2) + O(n)$
- $T(n) = O(\,n^2\,)$

- Karatsuba integer multiplication
- $T(n) = 3\,T(n/2) + O(n)$
- $T(n) = O(\,n^{\log_2(3)} \approx n^{1.6}\,)$

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

What's the pattern?!?!?!?!

The master theorem

- A formula for many recurrence relations.
  - You'll come up with an example in next class when it won't work.
- Proof: "Generalized" tree method.

A useful
formula it is.
Know why it works
you should.

Jedi master Yoda

The master theorem

We can also take n/b to mean either $\left\lfloor \frac{n}{b} \right\rfloor$ or $\left\lceil \frac{n}{b} \right\rceil$ and the theorem is still true.

- Suppose that $a \geq 1, b > 1,$ and d are constants (independent of n).
- Suppose $T(n) = a \cdot T\left(\frac{n}{b}\right) + O\left(n^d\right)$. Then

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

Three parameters:
a : number of subproblems
b : factor by which input size shrinks
d : need to do n^d work to create all the subproblems and combine their solutions.

Many symbols those are….

Technicalities II

Integer division

- If n is odd, I can't break it up into two problems of size n/2.

$$T(n) = T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + T\left(\left\lceil \frac{n}{2} \right\rceil\right) + O(n)$$

- However one can show that the Master theorem works fine if you pretend that what you have is:

$$T(n) = 2 \cdot T\left(\frac{n}{2}\right) + \ O(n)$$

- From now on we'll mostly **ignore floors and ceilings** in recurrence relations.

Examples
(details on board)

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + O(n^d).$$

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

- Needlessly recursive integer mult.
  - T(n) = 4 T(n/2) + O(n)
  - T(n) = O(n^2)

a = 4
b = 2
d = 1

$a > b^d$ ✓

- Karatsuba integer multiplication
  - T(n) = 3 T(n/2) + O(n)
  - T(n) = O($n^{log\_2(3)} \approx n^{1.6}$)

a = 3
b = 2
d = 1

$a > b^d$ ✓

- MergeSort
  - T(n) = 2T(n/2) + O(n)
  - T(n) = O(nlog(n))

a = 2
b = 2
d = 1

$a = b^d$ ✓

- That other one
  - T(n) = T(n/2) + O(n)
  - T(n) = O(n)

a = 1
b = 2
d = 1

$a < b^d$ ✓

Understanding the Master Theorem

- Let $a \geq 1, b > 1,$ and d be constants.
- Suppose $T(n) = a \cdot T\left(\frac{n}{b}\right) + O\left(n^d\right)$. Then

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

- What do these three cases mean?

The eternal struggle

Branching causes the number of problems to explode!
The most work is at the bottom of the tree!

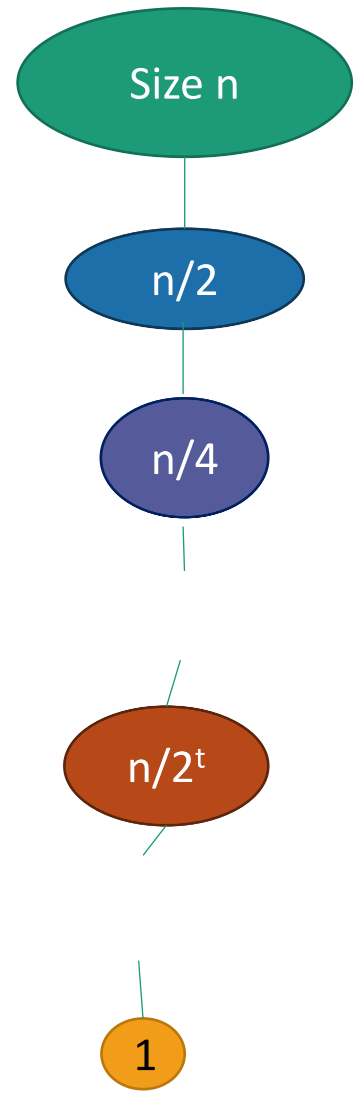
The problems lower in the tree are smaller!
The most work is at the top of the tree!

Consider our three warm-ups

1. $T(n) = T\left(\frac{n}{2}\right) + n$
2. $T(n) = 2 \cdot T\left(\frac{n}{2}\right) + n$
3. $T(n) = 4 \cdot T\left(\frac{n}{2}\right) + n$

First example: tall and skinny tree

1. $T(n) = T\left(\frac{n}{2}\right) + n, \qquad (a < b^d)$



- The amount of work done at the top (the biggest problem) swamps the amount of work done anywhere else.

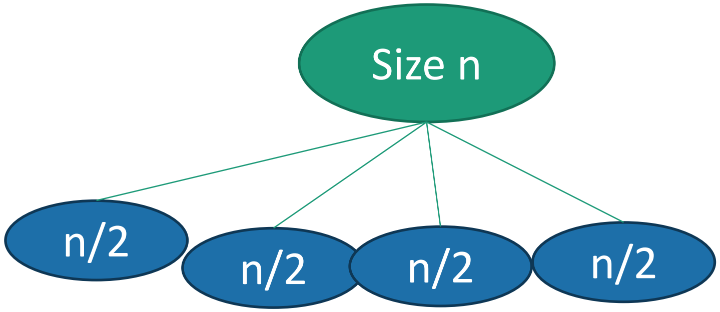
- T(n) = O(work at top) = O(n)

WINNER

Most work at the top of the tree!

Third example: bushy tree

$$3.\quad T(n) = 4 \cdot T\left(\frac{n}{2}\right) + n, \qquad (a > b^d)$$

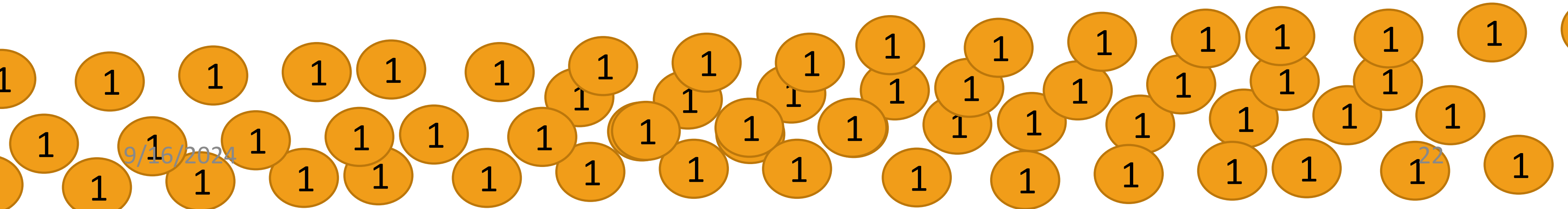


WINNER

Most work at the bottom of the tree!

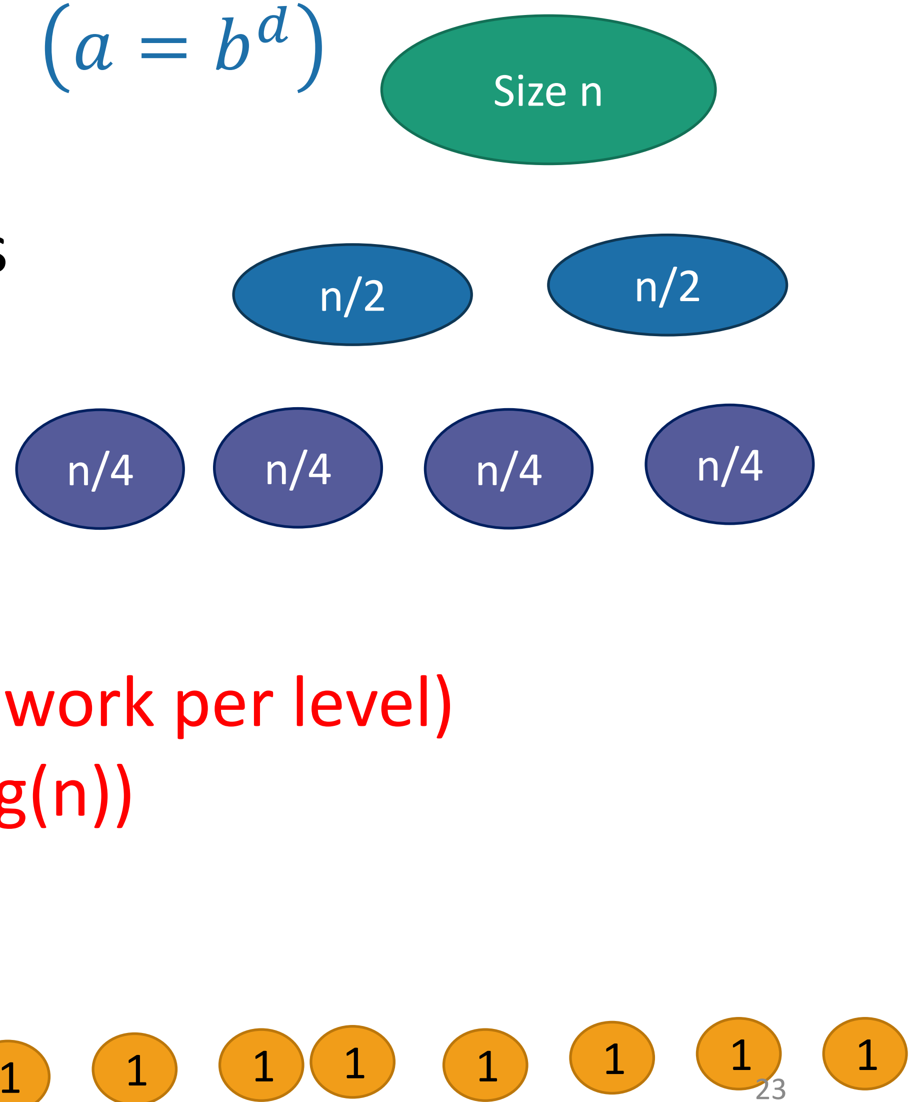
- There are a HUGE number of leaves, and the total work is dominated by the time to do work at these leaves.

- T(n) = O(work at bottom) = O($4^{\text{depth of tree}}$) = O(n^2)



Second example: just right

2. $T(n) = 2 \cdot T\left(\frac{n}{2}\right) + n, \qquad \left(a = b^d\right)$



- The branching **just** balances out the amount of work.
- The same amount of work is done at every level.
- T(n) = (number of levels) * (work per level)
- = log(n) * O(n) = O(nlog(n))

What have we learned?

- The “Master Method” makes our lives easier.
- But it’s basically just codifying a calculation we could do from scratch if we wanted to.

The Substitution Method

- Another way to solve recurrence relations.
- More general than the master method.

The Substitution Method

first example

- Let's return to:

$$T(n) = 2 \cdot T\left(\frac{n}{2}\right) + n, \text{ with } T(1) = 1.$$

  - (assuming n is a power of 2…)
- The Master Method says $T(n) = O(n \log(n))$.
- We will prove this via the Substitution Method.

$T(n) = 2 \cdot T\left(\frac{n}{2}\right) + n$, with $T(1) = 1$.

Step 1: Guess the answer

- $T(n) = 2 \cdot T\left(\frac{n}{2}\right) + n$
- $T(n) = 2 \cdot \left(2 \cdot T\left(\frac{n}{4}\right) + \frac{n}{2}\right) + n$
- $T(n) = 4 \cdot T\left(\frac{n}{4}\right) + 2n$
- $T(n) = 4 \cdot \left(2 \cdot T\left(\frac{n}{8}\right) + \frac{n}{4}\right) + 2n$
- $T(n) = 8 \cdot T\left(\frac{n}{8}\right) + 3n$
- ...

Expand $T\left(\frac{n}{2}\right)$

Simplify

Expand $T\left(\frac{n}{4}\right)$

Simplify

You can guess the answer however you want: meta-reasoning, a little bird told you, wishful thinking, etc. One useful way is to try to "unroll" the recursion, like we're doing here.

Guessing the pattern: $T(n) = 2^j \cdot T\left(\frac{n}{2^j}\right) + j \cdot n$

Plug in $j = \log(n)$, and get

$$T(n) = n \cdot T(1) + \log(n) \cdot n = n(\log(n) + 1)$$

Why two methods?

- Sometimes the Substitution Method works where the Master Method does not.

Next Time

- What happens if the sub-problems are different sizes?
- And when might that happen?