طراحی الگوریتم ها

مبحث چهارم:
رابطه بازگشتی و قضیه اصلی

سجاد شیرعلی شهرضا

بهار 1402

سه شنبه، 25 بهمن 1401

Based on Stanford CS161 Slides from Summer 2020 by Karey Shi

اطلاع رسانی

- بخش مرتبط کتاب برای این جلسه: 4.3، 4.4، 4.5

زمان اجرای مرتب سازی ادغامی

چقدر سریع است؟

MERGESORT: IS IT FAST?

```
MERGESORT(A):
    n = len(A)
    if n <= 1:
        return A
    L = MERGESORT(A[0:n/2])
    R = MERGESORT(A[n/2:n])
    return MERGE(L,R)
```

CLAIM: MergeSort runs in time **O(n log n)**

AN ASIDE: O(n log n) vs. O(n^2)?

log(n) grows very slowly! (Much more slowly than n)

AN ASIDE: O(n log n) vs. O(n^2)?

log(n) grows very slowly! (Much more slowly than n)

ALL LOGARITHMS IN THIS COURSE ARE BASE 2

log(2) = 1
log(4) = 2
...
log(64) = 6
log (128) = 7
...
log(4096) = 12
...
log(**# particles in the universe**) < 280

AN ASIDE: O(n log n) vs. $O(n^2)$?

log(n) grows very slowly! (Much more slowly than n)

ALL LOGARITHMS IN THIS COURSE ARE BASE 2

log(2) = 1
log(4) = 2
...
log(64) = 6
log (128) = 7
...
log(4096) = 12
...
log(**# particles in the universe**) < 280

n log n grows much more slowly than n^2

Punchline: A running time of O(n log n) is a LOT better than $O(n^2)$

MERGESORT: O(n log n) PROOF

Instead of counting every little operation and tracing all recursive calls, we can think about:
THE RECURSION TREE!
(and we'll add up all the work done across levels to compute the Big-O runtime)

```
MERGESORT(A):
    n = len(A)
    if n <= 1:
        return A
    L = MERGESORT(A[0:n/2])
    R = MERGESORT(A[n/2:n])
    return MERGE(L,R)
```

```
MERGE(L,R):
    result = length n array
    i = 0, j = 0
    for k in [0,...,n-1]:
        if L[i] < R[j]:
            result[k] = L[i]
            i += 1
        else:
            result[k] = R[j]
            j += 1
    return result
```

MERGESORT: O(n log n) PROOF

Instead of counting every little operation and tracing all recursive calls, we can think about:
THE RECURSION TREE!
(and we'll add up all the work done across levels to compute the Big-O runtime)

```
MERGESORT(A):
    n = len(A)
    if n <= 1:
        return A
    L = MERGESORT(A[0:n/2])
    R = MERGESORT(A[n/2:n])
    return MERGE(L,R)
```

```
MERGE(L,R):
    result = length n array
    i = 0, j = 0
    for k in [0,...,n-1]:
        if L[i] < R[j]:
            result[k] = L[i]
            i += 1
        else:
            result[k] = R[j]
            j += 1
    return result
```

n iterations, O(1) work per iteration

We can see that MERGE is **O(n)**

MERGESORT: O(n log n) PROOF

Instead of counting every little operation and tracing all recursive calls, we can think about:
THE RECURSION TREE!
(and we'll add up all the work done across levels to compute the Big-O runtime)

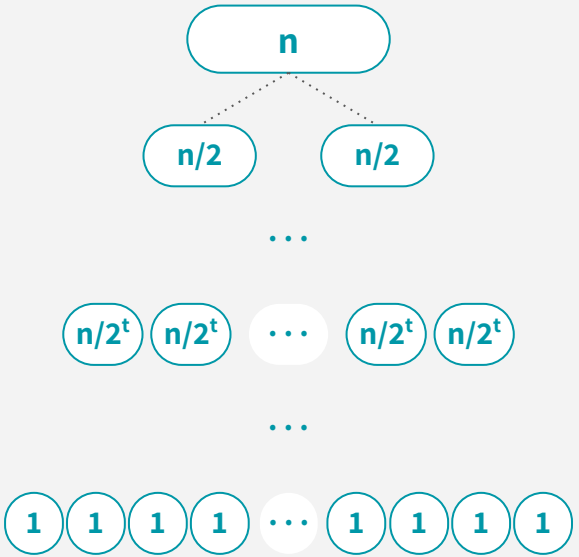
```
MERGESORT(A):
    n = len(A)
    if n <= 1:
        return A
    L = MERGESORT(A[0:n/2])
    R = MERGESORT(A[n/2:n])
    return MERGE(L,R)
```

```
MERGE(L,R):
    result = length n array
    ...
    return result
```

This means that within one recursive call that processes an array/subarray of length ***n***, the work done in that subproblem (creating subproblems & "merging" those results) is **O(n)**.

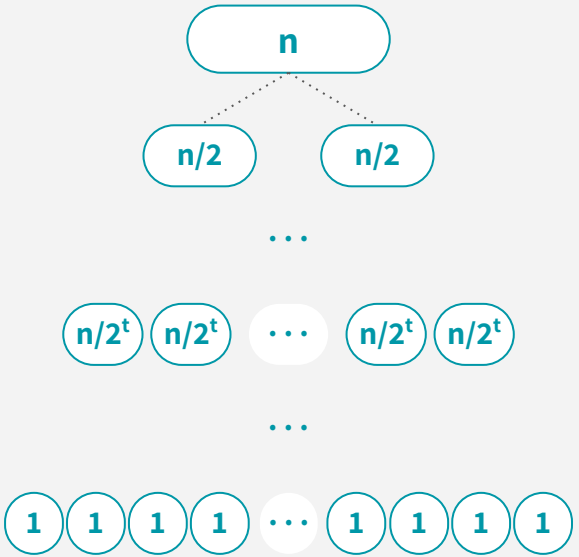
We can see that MERGE is **O(n)**

MERGESORT RECURSION TREE



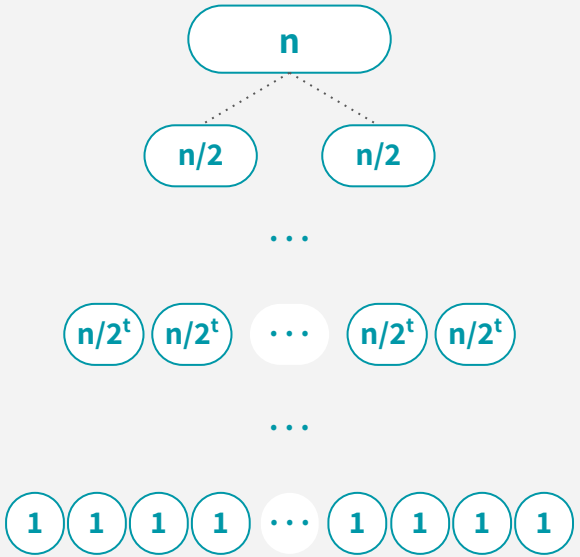
Level	Size of each Problem	# of Problems	Work done per Problem	Total work on this level
0				
1				
…				
t				
…				
?				

MERGESORT RECURSION TREE



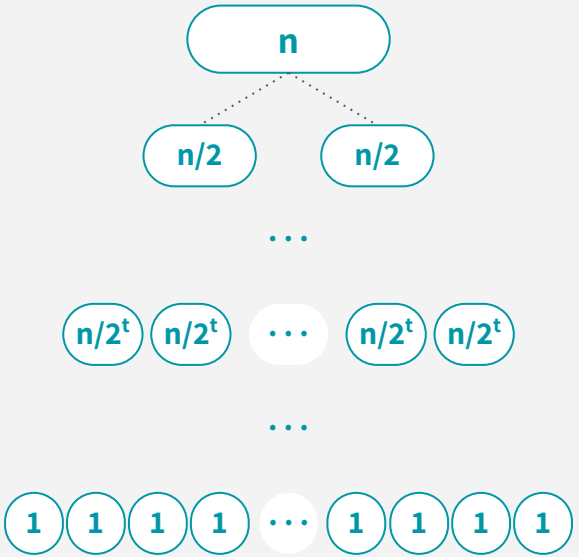
Level	Size of each Problem	# of Problems	Work done per Problem	Total work on this level
0				
1				
…				
t				
…				
$\log_2 n$				

MERGESORT RECURSION TREE



Level	Size of each Problem	# of Problems	Work done per Problem	Total work on this level
0	n			
1	$n/2^1$			
…				
t	$n/2^t$			
…				
$\log_2 n$	1			

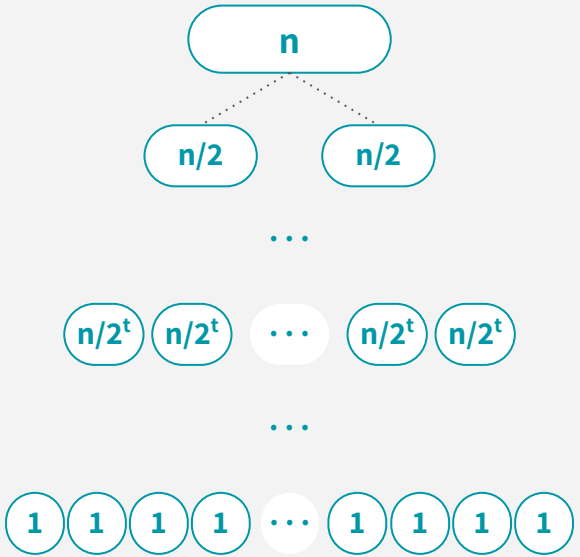
MERGESORT RECURSION TREE



Level	Size of each Problem	# of Problems	Work done per Problem	Total work on this level
0	n	1		
1	$n/2^1$	2^1		
…				
t	$n/2^t$	2^t		
…				
$\log_2 n$	1	$2^{\log_2 n} = n$		

MERGESORT RECURSION TREE

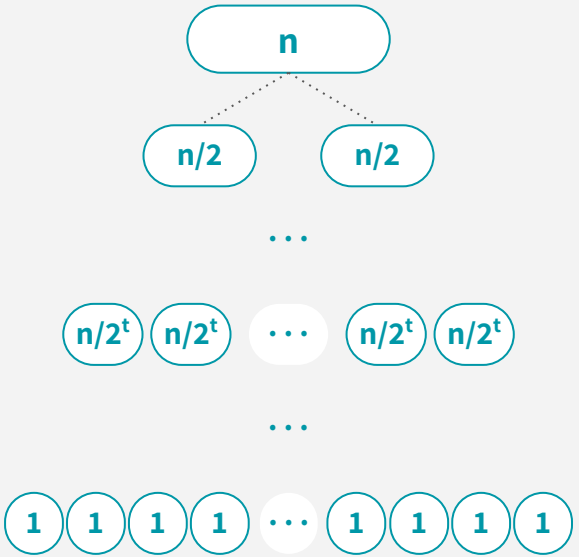
If a subproblem is of size **n**, then the work done in that subproblem is **O(n)**.
⇒ **Work ≤ c · n** (c is a constant)



Level	Size of each Problem	# of Problems	Work done per Problem	Total work on this level
0	n	1	$c \cdot n$	
1	$n/2^1$	2^1	$c \cdot (n/2)$	
…				
t	$n/2^t$	2^t	$c \cdot (n/2^t)$	
…				
$\log_2 n$	1	$2^{\log_2 n} = n$	$c \cdot (1)$	

MERGESORT RECURSION TREE

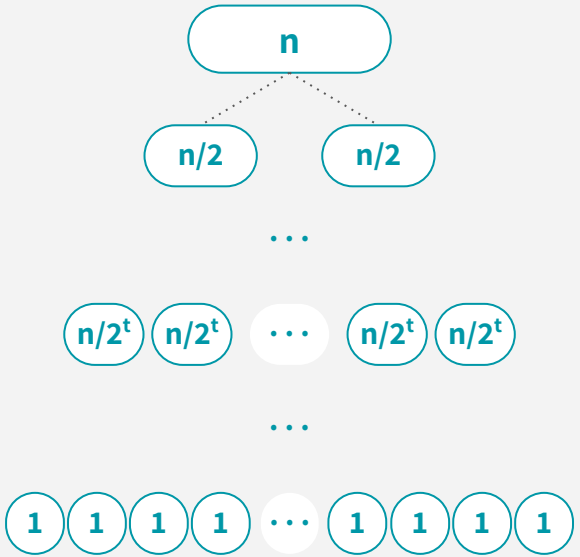
If a subproblem is of size **n**, then the work done in that subproblem is **O(n)**.
⇒ **Work ≤ c · n** (c is a constant)



Level	Size of each Problem	# of Problems	Work done per Problem	Total work on this level
0	n	1	$c \cdot n$	**O(n)**
1	$n/2^1$	2^1	$c \cdot (n/2)$	$2^1 \cdot c \cdot (n/2) =$ **O(n)**
…				
t	$n/2^t$	2^t	$c \cdot (n/2^t)$	$2^t \cdot c \cdot (n/2^t) =$ **O(n)**
…				
$\log_2 n$	1	$2^{\log_2 n} = n$	$c \cdot (1)$	$n \cdot c \cdot (1) =$ **O(n)**

MERGESORT RECURSION TREE

If a subproblem is of size **n**, then the work done in that subproblem is **O(n)**.
⇒ **Work ≤ c · n** (c is a constant)

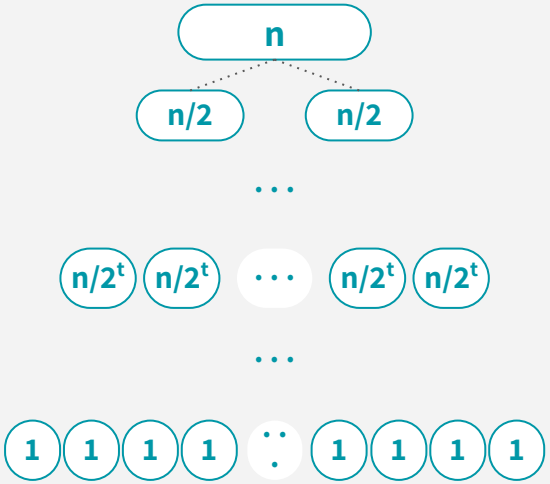


Level	Size of each Problem	# of Problems	Work done per Problem	Total work on this level
0	n	1	$c \cdot n$	**O(n)**
1	$n/2^1$	2^1	$c \cdot (n/2)$	$2^1 \cdot c \cdot (n/2) =$ **O(n)**
…				
t	$n/2^t$	2^t	$c \cdot (n/2^t)$	$2^t \cdot c \cdot (n/2^t) =$ **O(n)**
…				
$\log_2 n$	1	$2^{\log_2 n} = n$	$c \cdot (1)$	$n \cdot c \cdot (1) =$ **O(n)**

We have $(\log_2 n + 1)$ levels, each level has O(n) work total ⇒ **O(n log n)** work overall!

MERGESORT: O(n log n) RUNTIME

Using the "Recursion Tree Method" (i.e. drawing the tree & filling out the table), we showed that the runtime of MergeSort is **O(n log n)**



Level	Size of each Problem	# of Problems	Work done per Problem	Total work on this level
0	n	1	$c \cdot n$	**O(n)**
1	$n/2^1$	2^1	$c \cdot (n/2)$	$2^1 \cdot c \cdot (n/2) =$ **O(n)**
…				
t	$n/2^t$	2^t	$c \cdot (n/2^t)$	$2^t \cdot c \cdot (n/2^t) =$ **O(n)**
…				
$\log_2 n$	1	$2^{\log_2 n} = n$	$c \cdot (1)$	$n \cdot c \cdot (1) =$ **O(n)**

سوال؟

رابطه بازگشتی

RUNTIMES FOR RECURSIVE ALGOS

Previously, we used the "Recursion Tree Method" (i.e. drawing the tree & filling out the table) to manually add up all the work in the tree and find that the runtime of MergeSort is **O(n log n)**.

Drawing the tree & doing all that adding kind of takes a lot of work...
Here's another way to reason about the runtime of a recursive algorithm like Mergesort:

INTRODUCING...

RECURRENCE RELATIONS

RECURRENCE RELATIONS

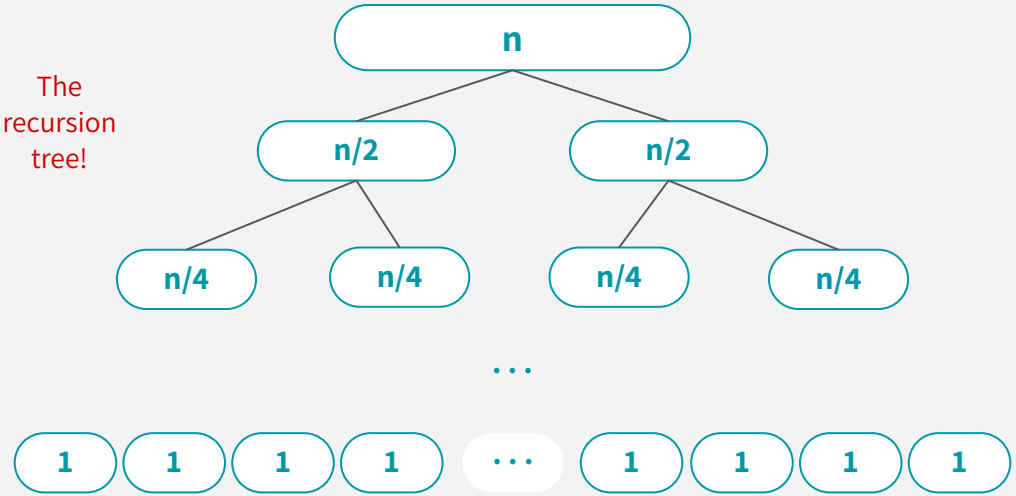
Basically, Recurrence Relations give us a *recursive* way to express runtimes for *recursive* algorithms!

We can then employ some math-ier approaches to analyze these recurrence relations.

RECURRENCE RELATIONS

To build the recurrence relation for MergeSort, we can think of its runtime as follows:

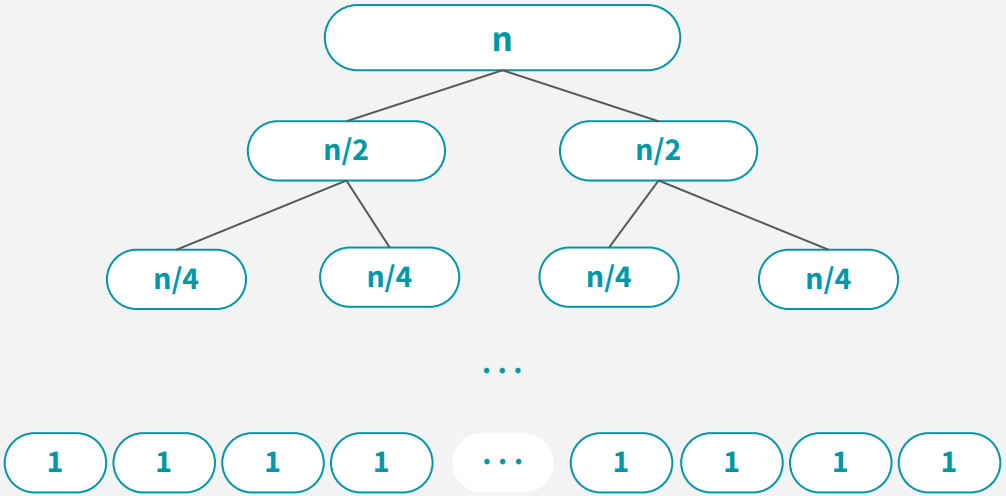
The recursion tree!



RECURRENCE RELATIONS

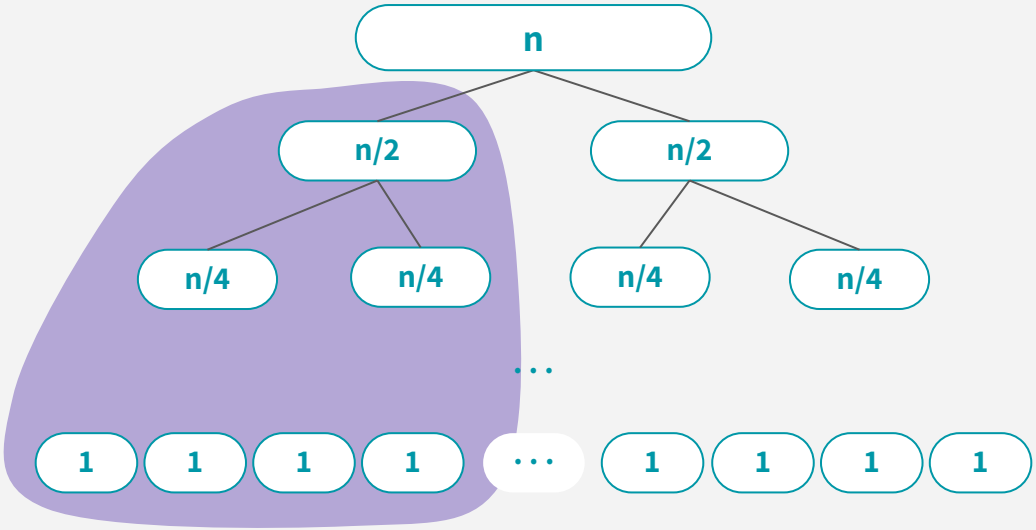
To build the recurrence relation for MergeSort, we can think of its runtime as follows:

Work in the whole tree =



RECURRENCE RELATIONS

To build the recurrence relation for MergeSort, we can think of its runtime as follows:

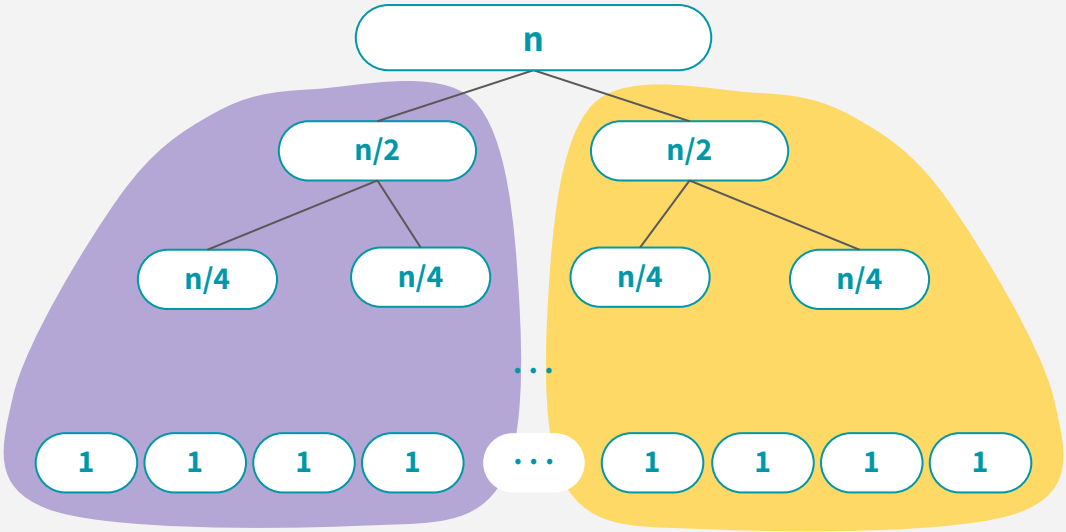


Work in the whole tree =

total work in LEFT recursive call (left subtree)

RECURRENCE RELATIONS

To build the recurrence relation for MergeSort, we can think of its runtime as follows:



Work in the whole tree =

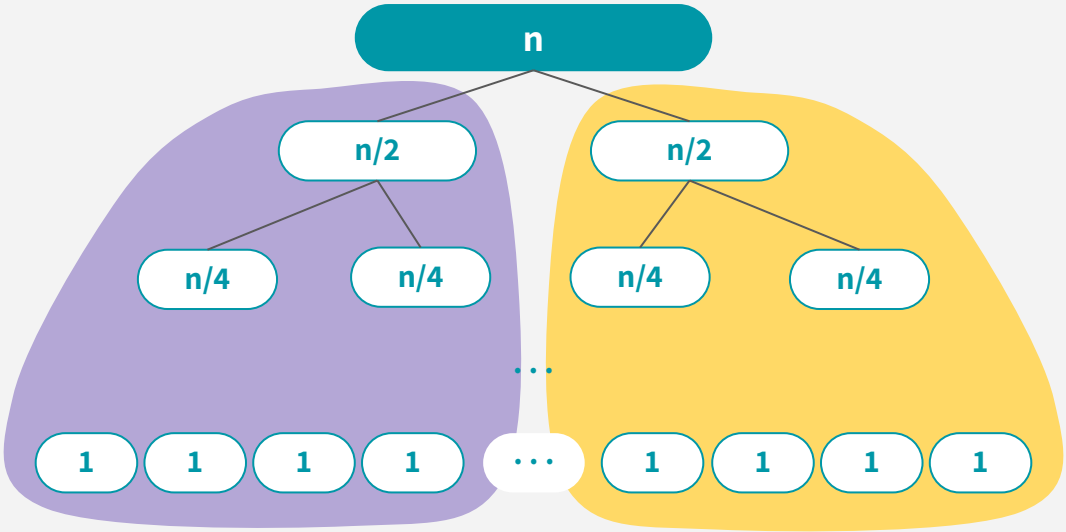
total work in LEFT recursive call (left subtree)

+

total work in RIGHT recursive call (right subtree)

RECURRENCE RELATIONS

To build the recurrence relation for MergeSort, we can think of its runtime as follows:



Work in the whole tree =

total work in LEFT recursive call (left subtree)

+

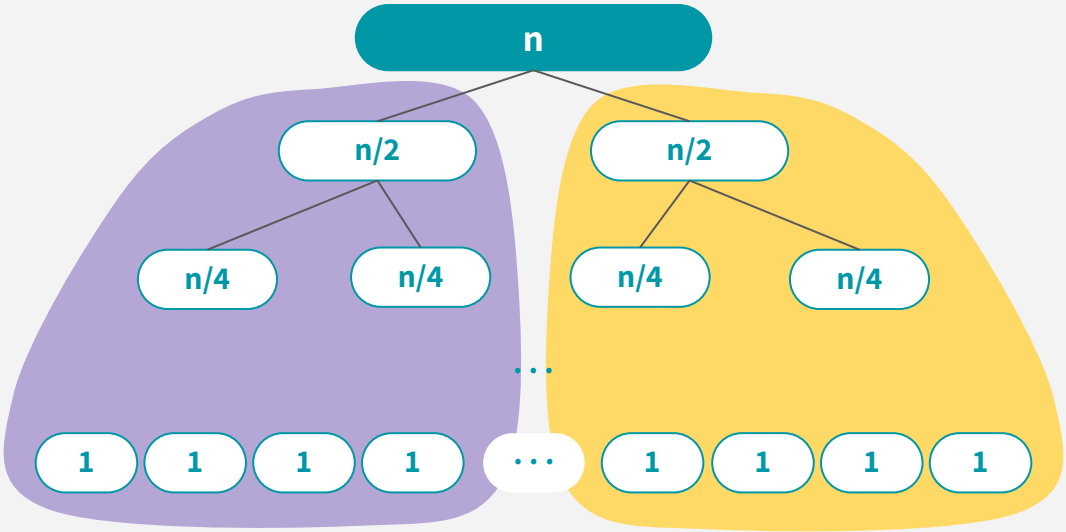
total work in RIGHT recursive call (right subtree)

+

work done *within* top problem

RECURRENCE RELATIONS

To build the recurrence relation for MergeSort, we can think of its runtime as follows:



Work in the whole tree =

total work in LEFT recursive call (left subtree)

+

total work in RIGHT recursive call (right subtree)

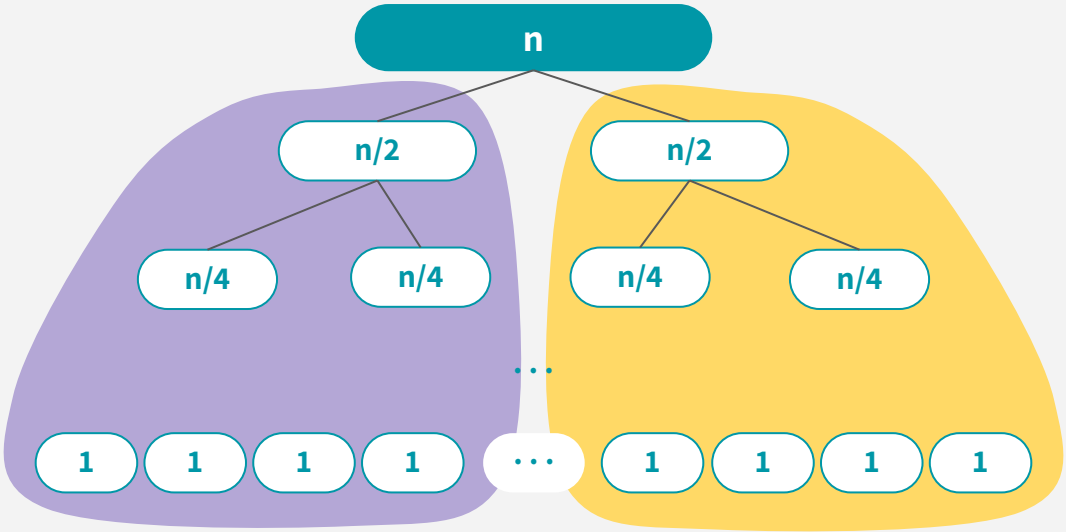
+

work done *within* top problem

work to create suproblems & "merge" their solutions

RECURRENCE RELATIONS

To build the recurrence relation for MergeSort, we can think of its runtime as follows:



T(n) =

total work in LEFT recursive call (left subtree)

+

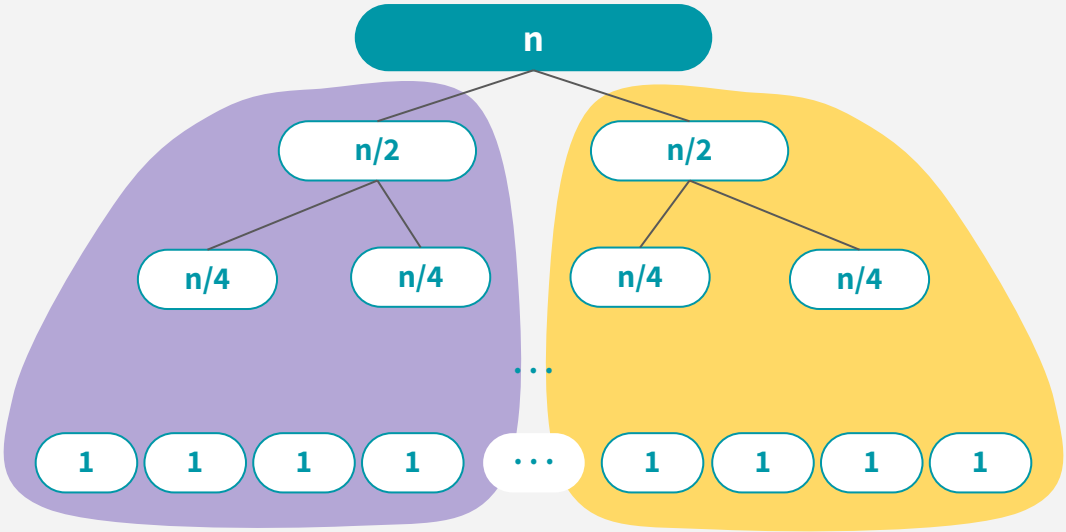
total work in RIGHT recursive call (right subtree)

+

work done *within* top problem

RECURRENCE RELATIONS

To build the recurrence relation for MergeSort, we can think of its runtime as follows:



T(n) =

T(n/2)

+

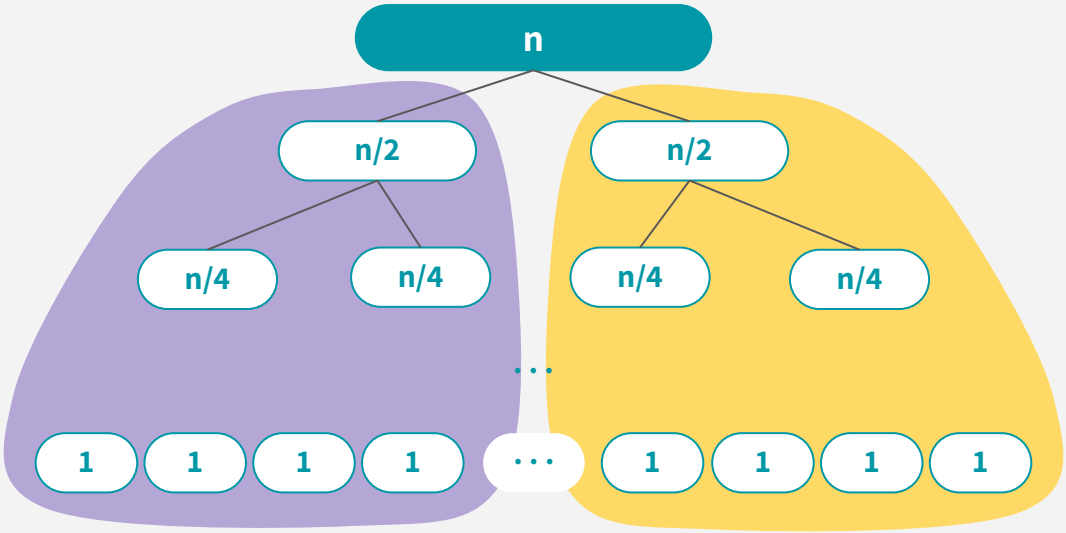
total work in RIGHT recursive call (right subtree)

+

work done *within* top problem

RECURRENCE RELATIONS

To build the recurrence relation for MergeSort, we can think of its runtime as follows:



T(n) =

T(n/2)

+

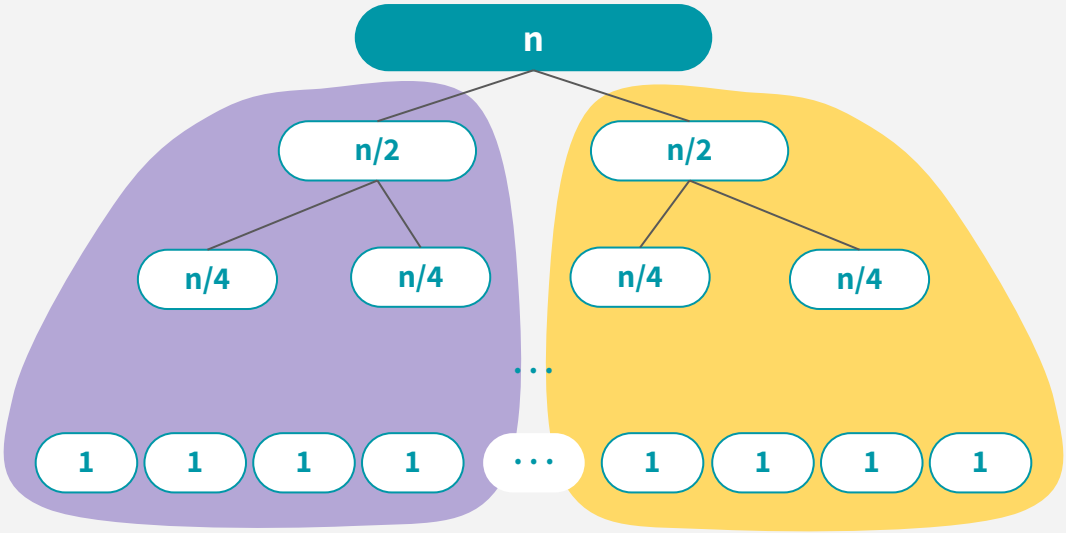
T(n/2)

+

work done *within* top problem

RECURRENCE RELATIONS

To build the recurrence relation for MergeSort, we can think of its runtime as follows:



T(n) =

T(n/2)

+

T(n/2)

+

O(n)

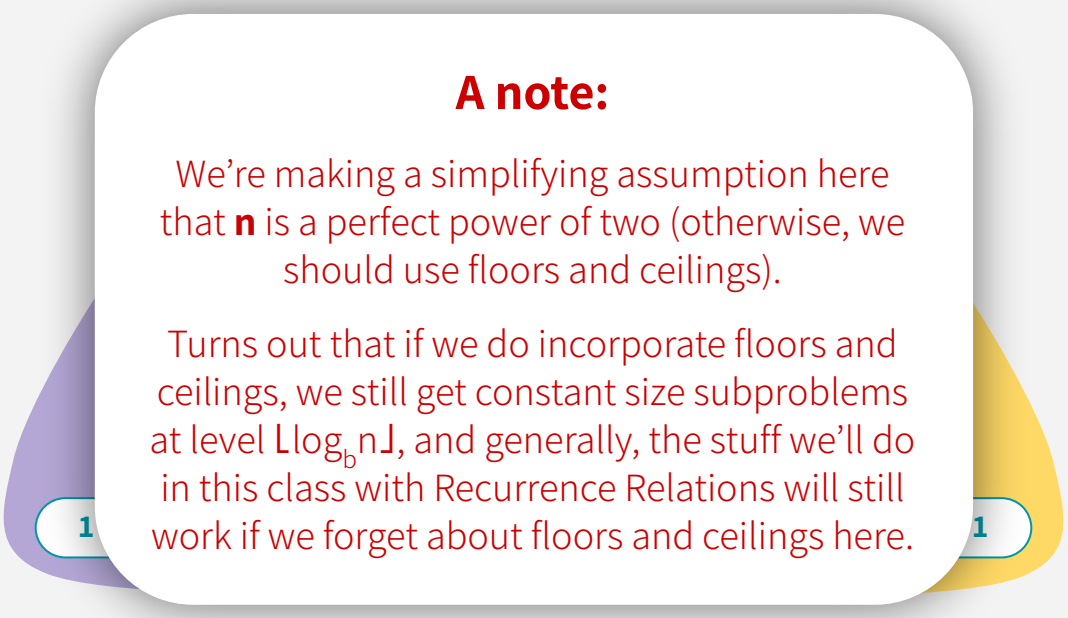
RECURRENCE RELATIONS

To build the recurrence relation for MergeSort, we can think of its runtime as follows:

A note:

We're making a simplifying assumption here that **n** is a perfect power of two (otherwise, we should use floors and ceilings).

Turns out that if we do incorporate floors and ceilings, we still get constant size subproblems at level $\lfloor \log_b n \rfloor$, and generally, the stuff we'll do in this class with Recurrence Relations will still work if we forget about floors and ceilings here.



T(n) =

T(n/2)

+

T(n/2)

+

O(n)

RECURRENCE RELATIONS

To build the recurrence relation for MergeSort, we can think of its runtime as follows:

$$\mathbf{T(n)} = T(n/2) + T(n/2) + \mathbf{O(n)}$$

since the subproblems are equal sizes, we can also write this as ***2 · T(n/2)***

This is a *recursive* definition for T(n), so we also need a BASE CASE:

$$\mathbf{T(1) = O(1)}$$

No matter what T is, T(1) = O(1). If it's greater than O(1), then the problem size wouldn't actually be 1.

Since we already used the Recursion Tree to compute the runtime of MergeSort, we know that **T(n) = O(n log n)**.

EXAMPLE RECURRENCE RELATIONS

Useless Divide-and-Conquer Multiplication

$$T(n) = 4 \cdot T(n/2) + O(n)$$
$$T(n) = O(n^{\log_2 4}) = \mathbf{O(n^2)}$$

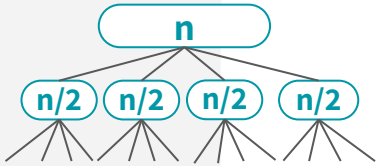
Karatsuba Integer Multiplication

$$T(n) = 3 \cdot T(n/2) + O(n)$$
$$T(n) = O(n^{\log_2 3}) \approx \mathbf{O(n^{1.6})}$$

MergeSort

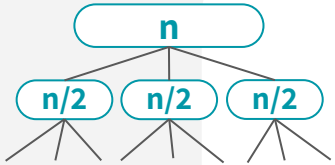
$$T(n) = 2 \cdot T(n/2) + O(n)$$
$$T(n) = \mathbf{O(n \log n)}$$

EXAMPLE RECURRENCE RELATIONS



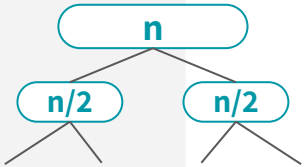
Useless Divide-and-Conquer Multiplication

$$T(n) = 4 \cdot T(n/2) + O(n)$$
$$T(n) = O(n^{\log_2 4}) = \mathbf{O(n^2)}$$



Karatsuba Integer Multiplication

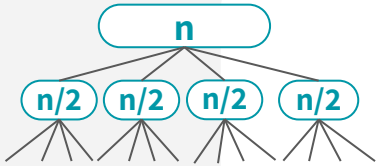
$$T(n) = 3 \cdot T(n/2) + O(n)$$
$$T(n) = O(n^{\log_2 3}) \approx \mathbf{O(n^{1.6})}$$



MergeSort

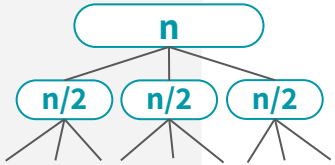
$$T(n) = 2 \cdot T(n/2) + O(n)$$
$$T(n) = \mathbf{O(n \log n)}$$

EXAMPLE RECURRENCE RELATIONS



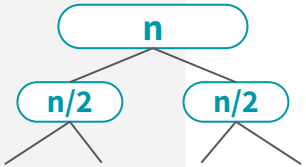
Useless Divide-and-Conquer Multiplication

$$T(n) = 4 \cdot T(n/2) + O(n)$$
$$T(n) = O(n^{\log_2 4}) = \mathbf{O(n^2)}$$



Karatsuba Integer Multiplication

$$T(n) = 3 \cdot T(n/2) + O(n)$$
$$T(n) = O(n^{\log_2 3}) \approx \mathbf{O(n^{1.6})}$$



MergeSort

$$T(n) = 2 \cdot T(n/2) + O(n)$$
$$T(n) = \mathbf{O(n \log n)}$$

IS THERE A
PATTERN
???

سوال؟

قضیه اصلی

فرمولی برای حل بسیاری از روابط بازگشتی
(اما نه همه آنها!)

THE MASTER THEOREM

Suppose that **a ≥ 1**, **b > 1**, and **d** are constants (i.e. independent of **n**).

Suppose $\mathbf{T(n) = a \cdot T(n/b) + O(n^d)}$. The Master Theorem states:

THE MASTER THEOREM

Suppose that **a ≥ 1**, **b > 1**, and **d** are constants (i.e. independent of **n**).

Suppose $\mathbf{T(n) = a \cdot T(n/b) + O(n^d)}$. The Master Theorem states:

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

a: number of subproblems (branching factor)

b: factor by which input size shrinks (shrinking factor)

d: need to do $O(n^d)$ work to create subproblems + "merge" their solutions

MASTER THEOREM EXAMPLES

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

a: # of subproblems (branching factor)
b: factor by how input size shrinks (shrinking factor)
d: need to do $O(n^d)$ work to create subproblems + "merge" solutions

USELESS DIVIDE & CONQUER MULTIPLICATION

$$\mathbf{T(n) = 4 \cdot T(n/2) + O(n)}$$

MASTER THEOREM EXAMPLES

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

a: # of subproblems (branching factor)
b: factor by which input size shrinks (shrinking factor)
d: need to do $O(n^d)$ work to create subproblems + "merge" solutions

USELESS DIVIDE & CONQUER MULTIPLICATION

$$\mathbf{T(n) = 4 \cdot T(n/2) + O(n)}$$

a = 4
b = 2
d = 1

MASTER THEOREM EXAMPLES

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

a: # of subproblems (branching factor)
b: factor by which input size shrinks (shrinking factor)
d: need to do $O(n^d)$ work to create subproblems + "merge" solutions

USELESS DIVIDE & CONQUER MULTIPLICATION

$$T(n) = 4 \cdot T(n/2) + O(n)$$

a = 4
b = 2
d = 1

$$a > b^d$$

MASTER THEOREM EXAMPLES

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

a: # of subproblems (branching factor)
b: factor by which input size shrinks (shrinking factor)
d: need to do $O(n^d)$ work to create subproblems + "merge" solutions

USELESS DIVIDE & CONQUER MULTIPLICATION

$T(n) = 4 \cdot T(n/2) + O(n)$
$T(n) = O(n^{\log_2 4}) =$ **$O(n^2)$**

a = 4
b = 2
d = 1

$a > b^d$

MASTER THEOREM EXAMPLES

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

a: # of subproblems (branching factor)
b: factor by which input size shrinks (shrinking factor)
d: need to do $O(n^d)$ work to create subproblems + "merge" solutions

USELESS DIVIDE & CONQUER MULTIPLICATION

$T(n) = 4 \cdot T(n/2) + O(n)$
$T(n) = O(n^{\log_2 4}) =$ **$O(n^2)$**

a = 4
b = 2
d = 1

$a > b^d$

KARATSUBA INTEGER MULTIPLICATION

$T(n) = 3 \cdot T(n/2) + O(n)$

MASTER THEOREM EXAMPLES

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

a: # of subproblems (branching factor)
b: factor by which input size shrinks (shrinking factor)
d: need to do $O(n^d)$ work to create subproblems + "merge" solutions

USELESS DIVIDE & CONQUER MULTIPLICATION

$\mathbf{T(n) = 4 \cdot T(n/2) + O(n)}$
$T(n) = O(n^{\log_2 4}) = \mathbf{O(n^2)}$

a = 4
b = 2
d = 1

$\mathbf{a > b^d}$

KARATSUBA INTEGER MULTIPLICATION

$\mathbf{T(n) = 3 \cdot T(n/2) + O(n)}$

a = 3
b = 2
d = 1

MASTER THEOREM EXAMPLES

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

a: # of subproblems (branching factor)
b: factor by which input size shrinks (shrinking factor)
d: need to do $O(n^d)$ work to create subproblems + "merge" solutions

USELESS DIVIDE & CONQUER MULTIPLICATION	**$T(n) = 4 \cdot T(n/2) + O(n)$** $T(n) = O(n^{\log_2 4}) = \mathbf{O(n^2)}$	**a = 4** **b = 2** **d = 1**	**$a > b^d$**
KARATSUBA INTEGER MULTIPLICATION	**$T(n) = 3 \cdot T(n/2) + O(n)$**	**a = 3** **b = 2** **d = 1**	**$a > b^d$**

MASTER THEOREM EXAMPLES

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

a: # of subproblems (branching factor)
b: factor by which input size shrinks (shrinking factor)
d: need to do $O(n^d)$ work to create subproblems + "merge" solutions

USELESS DIVIDE & CONQUER MULTIPLICATION	**$T(n) = 4 \cdot T(n/2) + O(n)$** $T(n) = O(n^{\log_2 4}) = \mathbf{O(n^2)}$	**a = 4** **b = 2** **d = 1**	**$a > b^d$**
KARATSUBA INTEGER MULTIPLICATION	**$T(n) = 3 \cdot T(n/2) + O(n)$** $T(n) = O(n^{\log_2 3}) \approx \mathbf{O(n^{1.6})}$	**a = 3** **b = 2** **d = 1**	**$a > b^d$**

MASTER THEOREM EXAMPLES

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

a: # of subproblems (branching factor)
b: factor by which input size shrinks (shrinking factor)
d: need to do $O(n^d)$ work to create subproblems + "merge" solutions

USELESS DIVIDE & CONQUER MULTIPLICATION	**$T(n) = 4 \cdot T(n/2) + O(n)$** $T(n) = O(n^{\log_2 4}) = \mathbf{O(n^2)}$	**a = 4** **b = 2** **d = 1**	**$a > b^d$**
KARATSUBA INTEGER MULTIPLICATION	**$T(n) = 3 \cdot T(n/2) + O(n)$** $T(n) = O(n^{\log_2 3}) \approx \mathbf{O(n^{1.6})}$	**a = 3** **b = 2** **d = 1**	**$a > b^d$**
MERGESORT	**$T(n) = 2 \cdot T(n/2) + O(n)$**		

MASTER THEOREM EXAMPLES

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

a: # of subproblems (branching factor)
b: factor by which input size shrinks (shrinking factor)
d: need to do $O(n^d)$ work to create subproblems + "merge" solutions

USELESS DIVIDE & CONQUER MULTIPLICATION	**$T(n) = 4 \cdot T(n/2) + O(n)$** $T(n) = O(n^{\log_2 4}) = \mathbf{O(n^2)}$	**a = 4** **b = 2** **d = 1**	**$a > b^d$**
KARATSUBA INTEGER MULTIPLICATION	**$T(n) = 3 \cdot T(n/2) + O(n)$** $T(n) = O(n^{\log_2 3}) \approx \mathbf{O(n^{1.6})}$	**a = 3** **b = 2** **d = 1**	**$a > b^d$**
MERGESORT	**$T(n) = 2 \cdot T(n/2) + O(n)$**	**a = 2** **b = 2** **d = 1**	

MASTER THEOREM EXAMPLES

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

a: # of subproblems (branching factor)
b: factor by which input size shrinks (shrinking factor)
d: need to do $O(n^d)$ work to create subproblems + "merge" solutions

USELESS DIVIDE & CONQUER MULTIPLICATION	**$T(n) = 4 \cdot T(n/2) + O(n)$** $T(n) = O(n^{\log_2 4}) = \mathbf{O(n^2)}$	**a = 4** **b = 2** **d = 1**	**$a > b^d$**
KARATSUBA INTEGER MULTIPLICATION	**$T(n) = 3 \cdot T(n/2) + O(n)$** $T(n) = O(n^{\log_2 3}) \approx \mathbf{O(n^{1.6})}$	**a = 3** **b = 2** **d = 1**	**$a > b^d$**
MERGESORT	**$T(n) = 2 \cdot T(n/2) + O(n)$**	**a = 2** **b = 2** **d = 1**	**$a = b^d$**

MASTER THEOREM EXAMPLES

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

a: # of subproblems (branching factor)
b: factor by which input size shrinks (shrinking factor)
d: need to do $O(n^d)$ work to create subproblems + "merge" solutions

USELESS DIVIDE & CONQUER MULTIPLICATION	**$T(n) = 4 \cdot T(n/2) + O(n)$** $T(n) = O(n^{\log_2 4}) = \mathbf{O(n^2)}$	**a = 4** **b = 2** **d = 1**	**$a > b^d$**
KARATSUBA INTEGER MULTIPLICATION	**$T(n) = 3 \cdot T(n/2) + O(n)$** $T(n) = O(n^{\log_2 3}) \approx \mathbf{O(n^{1.6})}$	**a = 3** **b = 2** **d = 1**	**$a > b^d$**
MERGESORT	**$T(n) = 2 \cdot T(n/2) + O(n)$** $T(n) = \mathbf{O(n \log n)}$	**a = 2** **b = 2** **d = 1**	**$a = b^d$**

سوال؟

اثبات قضیه اصلی

MASTER THEOREM "INTUITION"

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

amount of work at each level ~ same

highest level "dominates": work per level decreases (subproblem work shrinks more!)

leaves "dominate": work per level increases (branch more!)

a: number of subproblems (branching factor)

b: factor by which input size shrinks (shrinking factor)

d: need to do $O(n^d)$ work to create subproblems + "merge" their solutions

MASTER THEOREM "INTUITION"

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

To see why this is true:

We need to look at a "generalized" recursion tree

increases (branch more!)

a: number of subproblems (branching factor)

b: factor by which input size shrinks (shrinking factor)

d: need to do $O(n^d)$ work to create subproblems + "merge" their solutions

MASTER THEOREM "INTUITION"

Original set up: $T(n) = a \cdot T(n/b) + O(n^d)$

We're going to suppose that $T(n) \leq a \cdot T(n/b) + c \cdot n^d$

Wait a minute: Is it okay to just replace $O(n^d)$ with $c \cdot n^d$? What about n_0? Do we just drop that? For simplicity, we're going to assume that we're using $\mathbf{n_0 = 1}$ in the definition of big-O. (The proof of the Master Theorem" will work for larger n_0 too - verify this yourself!)

GENERALIZED RECURRENCE TREE

1. Draw the tree for $\mathbf{T(n) = a \cdot T(n/b) + c \cdot n^d}$
2. Fill out the table & sum up last column (from t = 0 to t = $\log_b n$)

LEVEL	# OF PROBLEMS	SIZE OF EACH PROBLEM	WORK PER PROBLEM	TOTAL WORK AT THIS LEVEL
0	1	n	$c \cdot n^d$	$\mathbf{1 \cdot c \cdot n^d}$
1	a	n/b	$c \cdot (n/b)^d$	$\mathbf{a \cdot c \cdot (n/b)^d}$
...				
t	a^t	n/b^t	$c \cdot (n/b^t)^d$	$\mathbf{a^t \cdot c \cdot (n/b^t)^d}$
...				
$\log_b n$	$a^{\log_b n}$	$n/b^{\log_b n} = 1$	$c \cdot (n/b^{\log_b n})^d$	$\mathbf{a^{\log_b n} \cdot c \cdot (n/b^{\log_b n})^d}$ (This = 1)

Total amount of work:

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

(add work across all levels up, then factor out the c & n^d terms & write in summation form)

GENERALIZED RECURRENCE TREE

1. Draw the tree for $\mathbf{T(n) = a \cdot T(n/b) + c \cdot n^d}$
2. Fill out the table & sum up last column (from t = 0 to t = $\log_b n$)

So T(n) ≤ $c \cdot n^d \cdot \sum_{t=0}^{log_b(n)} \left(\frac{a}{b^d}\right)^t$

We can verify that for each of the three cases (**a =, <, or > b^d**), this equation above gives us the desired results:

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

CASE 1: $a = b^d$

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

$$\begin{aligned} T(n) &= c \cdot n^d \cdot \sum_{t=0}^{\log_b(n)} \left(\frac{a}{b^d}\right)^t \\ &= c \cdot n^d \cdot \sum_{t=0}^{\log_b(n)} 1 \\ &= c \cdot n^d \cdot (\log_b(n) + 1) \\ &= c \cdot n^d \cdot \left(\frac{\log(n)}{\log(b)} + 1\right) \\ &= \Theta(n^d \log(n)) \end{aligned}$$

This is equal to 1!

CASE 2: $a < b^d$

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

This is less than 1!

$$\begin{aligned} T(n) &= c \cdot n^d \cdot \sum_{t=0}^{\log_b(n)} \left(\frac{a}{b^d}\right)^t \\ &= c \cdot n^d \cdot [\text{ some constant }] \\ &= \Theta(n^d) \end{aligned}$$

Geometric series with the "multiplier" < 1 & constant!

CASE 2: $a > b^d$

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

$$T(n) = c \cdot n^d \cdot \sum_{t=0}^{\log_b(n)} \left(\frac{a}{b^d}\right)^t$$

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

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

This is greater than 1!

The n^d term cancels with $(b^d)^{\log_b n}$! And $\mathbf{a^{\log_b n} = n^{\log_b a}}$!

Use the geometric series formula to convince yourself that this is legitimate!

WE CHECKED ALL THREE CASES!

$$\mathbf{T(n) = a \cdot T(n/b) + O(n^d)}$$

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

سوال؟