

# Insertion Sort, Proving Correctness, Run time analysis, Merge Sort, Quick Sort

## 1 Introduction to Sorting - Insertion Sort (CLRS §2.1, 2.2)

Input: Array  $A[1 \dots n]$  of  $n$  number

Goal: Sort in non decreasing order.

in case of repeated elements (doesn't strictly increase)

How do we write the algorithm?

### 1.1 The algorithm in words

Scan the array from left to right.  
For each position  $j$ , insert  $A[j]$  into its correct position in  $A[1 \dots j]$  by moving all elements larger than  $A[j]$  one position to the right

### 1.2 Pseudocode

```

1. for  $j = 2$  to  $n$ :
2.    $Key \leftarrow A[j]$ 
3.    $i \leftarrow j - 1$ 
4.   while  $i > 0$  and  $A[i] > Key$ :
5.      $A[i+1] \leftarrow A[i]$ 
6.      $i \leftarrow i - 1$ 
7.    $A[i+1] \leftarrow Key$ 

```

move element one position to the right

## 2 Proof by Induction, Loop Invariants, and Proving Correctness

- **Proof by induction:** We want to prove some statement  $P(n)$  for integers  $n \geq 0$ . (or  $n \in \mathbb{N}$  i.e.  $n \geq 1$ )

1. **Base case:** Prove  $P(0)$  or  $P(1)$

2. **Inductive step:**

i.e. show  $P(k) \Rightarrow P(k+1)$

• Inductive hypothesis: Assume  $P(n)$  holds for  $n = k$   $n = k-1$   
 $[n = 0, 1, \dots, k]$  strong induction

• Goal: Show  $P(k+1)$  holds  $P(k)$

**WARNING!** DO NOT ASSUME  $P(k+1)$  and "work backwards" to a true statement for the formal proof.

3. **Conclusion:**  $P(n)$  holds for all integers  $n \geq 0$ . by principle of mathematical induction!

- **Loop invariant:** A property that holds throughout the execution of the algorithm

1. **Initialization:** loop invariant holds prior to the first iteration of the loop

2. **Maintenance:**

If the loop invariant is true before an iteration of the loop, then it remains true before the next iteration.

3. **Termination:** When the loop terminates, the invariant gives useful property that helps show that the algorithm is correct.

Our loop invariant for insertion sort:

At the beginning of the  $j$ th iteration (or equivalently, the end of  $j-1$ st iteration)

$A[1 \dots j-1]$  contains the elements that

were originally there, but in sorted order.

"Termination" if loop invariant holds at the end of the algorithm, it tells us the output is correct.

Claim: loop invariant holds for  $j=2, 3, 4, \dots, n+1$ .

At the end,  
 $j = n+1$

```

1. for  $j = 2$  to  $n$ :
2.    $Key \leftarrow A[j]$ 
3.    $i \leftarrow j-1$ 
4.   while  $i > 0$  and  $A[i] > Key$ :
5.      $A[i+1] \leftarrow A[i]$ 
6.      $i \leftarrow i-1$ 
7.    $A[i+1] \leftarrow Key$ 

```

## 2.1 Proof of correctness of Insertion Sort

By induction.

First base case of induction ("initialization")  
 when  $j=2$ ,  $A[1]$  is just one number and is  
 therefore sorted. ✓ inductive hypothesis.

Induction Step ("maintenance") Assume the  
loop invariant holds for  $j$ . We will prove it  
 holds for  $j+1$ .

By assumption  $A[1 \dots j-1]$  is sorted.  
 During the  $j$ th iteration, we insert  $A[j]$  into  
 its correct position. Therefore, after the  
 $j$ th iteration,  $A[1 \dots j]$  are sorted, so the  
 loop invariant holds at the end of the  $j$ th  
 iteration. □

$$\begin{aligned}
 S &= 1 + 2 + 3 + \dots + 98 + 99 + 100 \\
 S &= 100 + 99 + 98 + \dots + 3 + 2 + 1
 \end{aligned}$$

$\overset{n-2}{98} \quad \overset{n-1}{99} \quad \overset{n}{100}$ 
  
 $\underset{n}{100} \quad \underset{n-1}{99} \quad \underset{n-2}{98}$

$$\begin{aligned}
 2S &= 101 + 101 + 101 + \dots + 101 + 101 + 101 \\
 &= 100 \times 101
 \end{aligned}$$

$$S = \frac{100 \times 101}{2}$$

### 3 Introduction to analyzing running times

Example

$$\begin{cases} \text{for } i = 1 \text{ to } n \\ A[i] \leftarrow 5 \end{cases}$$

$n$  steps  
 $2n$  steps?  
 $3n$  steps?

$\Theta(n)$

#### 3.1 Running time of insertion sort

Actually mean "worst case"

$$\sum_{j=2}^n (c + c' t_j)$$

some constants (maybe  $c \approx 3$ ,  $c' \approx 2$ )

number of iterations of the while loop

```

1. for j = 2 to n:
2.   Key ← A[j]
3.   i ← j-1
4.   while i > 0 and A[i] > Key:
5.     A[i+1] ← A[i]
6.     i ← i-1
7.   A[i+1] ← Key

```

$\left. \begin{array}{l} \text{Key} \leftarrow A[j] \\ i \leftarrow j-1 \end{array} \right\} c$

$\left. \begin{array}{l} \text{while } i > 0 \text{ and } A[i] > \text{Key}: \\ \quad A[i+1] \leftarrow A[i] \\ \quad i \leftarrow i-1 \end{array} \right\} c'$

$A[i+1] \leftarrow \text{Key}$

$$\leq \sum_{j=2}^n (c + c' j) = c(n-1) + c' \sum_{j=2}^n j$$

worst case:  $t_j \leq j$

$j=2, \dots, n \rightarrow n-1$  iterations

$$= c(n-1) + c' \left( \frac{n(n+1)}{2} - 1 \right)$$

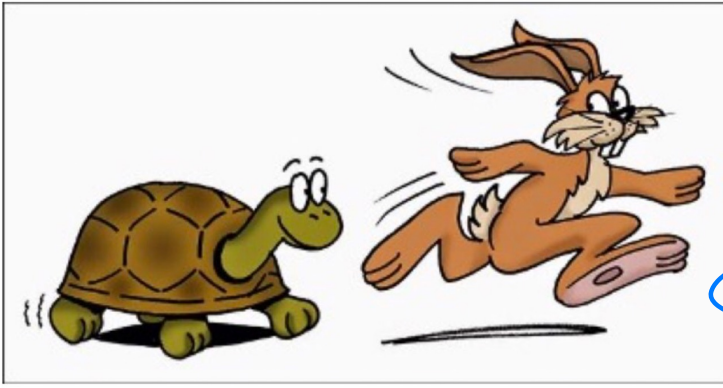
$$= \frac{c'}{2} n^2 + \left( c + \frac{c'}{2} \right) n - (c + c')$$

$$= \Theta(n^2) \leftarrow \text{running time of insertion sort.}$$





## Big-O (Asymptotic) Notation: Motivation



**Motivation** Why do we use it?

Which one below is easier to read?

- ☐ Algorithm  $A$  sorts a list containing  $n$  items in time at most  $2.95n^2 + 10n + 5$
- ☐ Algorithm  $A$  sorts a list containing  $n$  items in time at most  $Cn^2$  for some constant  $C > 0$
- ☐ Algorithm  $A$  sorts a list containing  $n$  items in time  $O(n^2)$

Prove that  $\sum_{i=1}^n i = \frac{n(n+1)}{2}$   $n \in \mathbb{N}$

let  $k \in \mathbb{N}$

inductive step

Inductive hypothesis: Assume

$$\sum_{i=1}^k i = \frac{k(k+1)}{2}$$

want to show

$$\sum_{i=1}^{k+1} i = \frac{(k+1)(k+2)}{2}$$

DON'T START with

$$\sum_{i=1}^{k+1} i =$$

$$\frac{(k+1)(k+2)}{2}$$

Assumes what we are trying to show.

$$\sum_{i=1}^{k+1} i$$

$$= \sum_{i=1}^k i + (k+1)$$

$$= \frac{k(k+1)}{2} + (k+1)$$

by the inductive hypothesis

$$= \frac{(k+1)(k+2)}{2}$$

→ large input sizes

## 4 Growth of Functions and Asymptotic Behavior (CLRS §3.1)

Goal: Establish notation that enables us to compare the relative performance of different algorithms.

a "nice" function (one term, no constant factors)

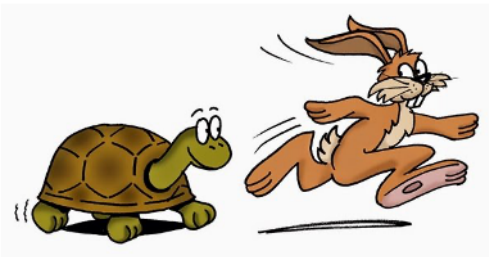
④ **Definition** large  $n$ . •  $T(n) = O(g(n))$  means there exists  $c > 0$  such that  $T(n) \leq cg(n)$  for sufficiently

(Big-Oh)

$T$  is bounded above by  $g$  asymptotically

⑤ •  $T(n) = \Omega(g(n))$  means there exists  $c > 0$  such that  $T(n) \geq cg(n)$  for sufficiently large  $n$ .

⑥ •  $T(n) = \Theta(g(n))$  means there exists  $c_1, c_2$  such that  $c_1g(n) \leq T(n) \leq c_2g(n)$  for sufficiently large  $n$ .



# O(.) notation exercises

## O(.)-Notation

### Asymptotic notation:

- Functions  $f(n)$  and  $g(n)$  represent running times.
- $f(n) = O(g(n))$  means that exist  $c, n_0 > 0$  such that  $f(n) \leq c \cdot g(n)$  for every  $n \geq n_0$ .
- Intuitively.**  $f(n)$  does not grow faster than  $g(n)$  when  $n$  is large.

negation:

For all  $c > 0$ ,  $n_0 > 0$   
there exists some  $n \geq n_0$   
such that  $100n^2 > cn$

$$100n^2 \neq O(n)$$

$$\text{Take } n \geq \max\{c, n_0\}$$

$$n \geq c$$

$$n \cdot n \geq cn$$

$$100n^2 \geq 100cn$$

$$> cn$$

$$4^n \neq O(2^n)$$

$$\lim_{n \rightarrow \infty} \frac{4^n}{2^n} = \lim_{n \rightarrow \infty} \left(\frac{4}{2}\right)^n$$

$$= \lim_{n \rightarrow \infty} 2^n = \infty$$

Which of the following are true and why? Discuss with your teammates!

For true statements, find valid values of  $c$  and  $n_0$

(Select all that apply.)

☒  $1000 = O(n)$

OR take  $c=1, n_0=1000$

☐  $100n^2 = O(n)$  ← false

☐  $n + n^2 + n^3 + n^4 = O(n^4)$  True

☐  $n^2 \cdot n^3 = O(n^4)$  False

☐  $n^5 - 2n^4 = O(n^4)$  False

☐  $n^3 + n^4 = O(n^4)$  True

☒  $4^n = O(2^n)$  false

$$\lim_{n \rightarrow \infty} \frac{100n^2}{n} = \lim_{n \rightarrow \infty} 100n = \infty$$

True  $\lim_{n \rightarrow \infty} \frac{1000}{n} = 0$

$\lim_{n \rightarrow \infty} \frac{2^{2n}}{2^n} = \lim_{n \rightarrow \infty} 2^n = \infty$

$4^n = (2^2)^n = 2^{2n}$   
 $100n = \infty$

$$n^3 + n^4 = O(n^4)$$

$$n^3 + n^4 \leq cn^4 \text{ for } n \geq n_0$$

$$c=2$$

$$n^3 + n^4 \leq 2n^4$$

$$n_0 = 1$$

$$n^3 + n^4 \leq n^4 + n^4$$

$$4^n \neq O(2^n)$$

$$2^{2n} = \underbrace{2^n} \cdot 2^n$$

let  $c > 0, n_0 > 0$

want to show

$$\underbrace{4^n} > \underbrace{c \cdot 2^n} \quad \text{for all } n \geq n_0$$

want

$$2^n \cdot 2^n > \underbrace{c \cdot 2^n}$$

(Scratch)

$$2^n \cdot 2^n > \underbrace{2^c \cdot 2^n}_{\geq c} \quad \text{if } n > c$$

take  $n = \max \{c, n_0\}$

$$2^n \geq 2^c$$

$$2^n \cdot 2^n \geq 2^c \cdot 2^n$$

$$\underbrace{4^n} \geq 2^c \cdot 2^n$$

$$> \underbrace{c \cdot 2^n}$$

## 4 Growth of Functions and Asymptotic Behavior (CLRS §3.1)

Goal: Establish notation that enables us to compare the relative performance of different algorithms.

complicated  $T(n) = 5.5n^2 + 7.789n \leq cn^2$   
 a "nice" function

$$\lim_{n \rightarrow \infty} \frac{T(n)}{g(n)} = C < \infty$$

**Definition**  $T(n) = O(g(n))$  means there exists  $c > 0$  such that  $T(n) \leq cg(n)$  for sufficiently large  $n$ .  
 (Big-oh)

Really  $O(g(n))$  is a set

$$O(g(n)) = \left\{ f: \mathbb{N} \rightarrow \mathbb{R}^+ \mid \exists c > 0 \exists n_0 \geq 1 \text{ such that } \forall n \geq n_0, f(n) \leq cg(n) \right\}$$

$T(n) \leq g(n)$

$g(n)$  is an asymptotic upper bound for  $T(n)$ ,  $T(n)$  does not grow faster than  $g(n)$  when  $n$  is large.

$T(n) = \Omega(g(n))$  means there exists  $c > 0$  such that  $T(n) \geq cg(n)$  for sufficiently large  $n$ .  
 (Omega)

$T(n) \geq g(n)$

$g(n)$  is an asymptotic lower bound for  $T(n)$

$$\lim_{n \rightarrow \infty} \frac{T(n)}{g(n)} > 0$$

$T(n) = \Theta(g(n))$  means there exists  $c_1, c_2$  such that  $c_1g(n) \leq T(n) \leq c_2g(n)$  for sufficiently large  $n$ .  
 (Theta)

For large  $n$ ,  $T(n)$  and  $g(n)$  are "equal" up to a constant factor

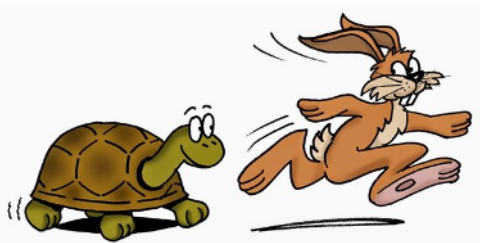
$$\lim_{n \rightarrow \infty} \frac{T(n)}{g(n)} = C > 0$$

$T(n) = o(g(n))$  (Little-oh)

means  $\lim_{n \rightarrow \infty} \frac{T(n)}{g(n)} = 0$

$T(n)$  is negligible in comparison to  $g(n)$ .

Remark if  $f \in \Omega(g(n))$  and  $f \in O(g(n))$  then  $f \in \Theta(g(n))$



**Example**

- polynomials - quadratic for any  $a > 0, b, c$   $an^2 + bn + c \in \Theta(n^2)$  *could be complicated*
- more generally  $a_k n^k + a_{k-1} n^{k-1} + \dots + a_0 \in \Theta(n^k)$   $a_k > 0$  *nice simple function*
- $n \log n = O(n^2)$
- in fact:  $\log n = O(n^\epsilon)$  for any  $\epsilon > 0$  *for little-oh*  $\lim_{n \rightarrow \infty} \frac{\ln \ln n}{n} = 0$
- $2^n \log n \not\in O(2^n)$  *not a constant factor*
- $3^n \not\in O(2^n)$  *but:  $3^n = \Omega(2^n)$*
- $3^n + 2^n \not\in \Theta(2^n)$   
 $= \Theta(3^n)$  *For all  $n \geq 1$*   
 $\frac{1}{c_1} \cdot 3^n \leq 3^n + 2^n \leq \frac{2}{c_2} \cdot 3^n$   *$3^n + 2^n \leq 3^n + 3^n$*

**4.1 Proofs involving order of growth**

Claim.  $f \in O(g(n))$  if and only if  $g \in \Omega(f(n))$

*added for posted notes (didn't go over in class)*

*Proof.*

We'll show if  $f \in O(g(n))$  then  $g = \Omega(f(n))$  ( $\Rightarrow$ )

Assume  $f \in O(g(n))$ . Then for all  $n$  sufficiently large,

$$f(n) \leq c g(n) \quad \text{for some } c > 0.$$

Then for all  $n$  sufficiently large,

$$g(n) \geq \frac{1}{c} f(n)$$

Thus there exists  $c_1$  (namely  $c_1 = \frac{1}{c}$ ) such that for  $n$

sufficiently large,  $g(n) \geq c_1 f(n)$ . Hence  $g \in \Omega(f(n))$ .

( $\Leftarrow$ ) Similar exercise for you to try! *i.e. lower bound up to constant factor for sufficiently large  $n$ .*  $\square$

**Example**  $3n^3 + 5n^2 + 10643n \in \Theta(n^3)$

$4n^3$

$$1 \cdot n^3 \leq 3n^3 + 5n^2 + 10643n \quad \text{for } n \geq 0$$

$$3n^3 + 5n^2 + 10643n \leq cn^3$$

$$n = 10^6 \quad \underbrace{3 \cdot 10^{18}} + \underbrace{5 \cdot 10^{12}} + \underbrace{10643 \cdot 10^6} \leq \underline{4 \cdot 10^{18}} \quad c = 4$$

**Example** Give an example of  $T(n)$  and  $g(n)$  such that  $T(n) \neq o(g(n))$ , but  $T(n) = O(g(n))$ .

$$T(n) = 2n^2$$

$$g(n) = n^2$$

$$\lim_{n \rightarrow \infty} \frac{2n^2}{n^2} = 2 \neq 0$$

Note: little-oh is stronger condition than big-oh.

**Example**  $\overbrace{5n^2 + 11} \in o(n^3) \quad \text{or} \quad o(n^4) \quad \text{or} \quad o(2^n)$   
 (little-oh)

## 4.2 Asymptotic notation in equations

A set in a formula represents an anonymous function in that set.

**Example**  $f(n) = n^3 + O(n^2)$

means there exists  $h(n) = O(n^2)$

such that  $f(n) = n^3 + h(n)$

e.g. if  $f(n) = n^3 + 2n$ , since  $2n \in O(n^2)$ ,  
 $f(n) = n^3 + O(n^2)$ .

Remark.  
 $f_1, f_2 \in O(g(n))$   
 $\Rightarrow f_1 + f_2 \in O(g(n))$   
 $2n = O(n^2)$   
 Since  $\lim_{n \rightarrow \infty} \frac{2n}{n^2} = 0$  or  
 for large enough  $n$ ,  
 $2n \leq 2n^2$   
 ( $c=2, n=1$ )

**Example**  $n^2 + O(n) = O(n^2)$

convention: implicit "for all" on left hand side of equals  
 and "there exists" on the right hand side.

$\rightarrow$  means for any  $f(n) = O(n)$ , there exists  $h(n) = O(n^2)$   
 such that  $n^2 + f(n) = h(n)$ .

intuitively: adding on any linear term to a quadratic  
 results in still bounded above by some quadratic.

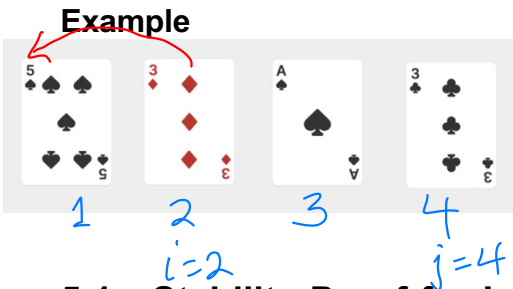


Also: (See posted Mergesort induction example)

## 5 Using a Loop Invariant to Prove Stability

HW 3  
Stability of Merge Sort

**Definition** A sorting algorithm is **stable** if objects with equal keys appear in the same order in the sorted output as in the unsorted input.



in the sorted output of a stable sorting algorithm  
3 of diamonds would appear before 3 of clubs.

Note  $i_1=2, j_1=4$   
 $i_2=1, j_2=4$   
 $i_3=2, j_3=4$

### 5.1 Stability Proof for Insertion Sort

Let  $i$  and  $j$  be <sup>initial</sup> indices with  $i < j$  such that  $A[i].key = A[j].key$ . We will show that their final positions  $i'$  and  $j'$  at the end of insertion sort satisfy  $i' < j'$ .

Loop invariant: Let  $i_k$  and  $j_k$  be the positions at the start of the  $k$ th iteration. Then  $i_k < j_k$ .

Initialization: Before the first iteration,

Maintenance:

Suppose for the inductive hypothesis that  $i_k < j_k$ . We will show that  $i_{k+1} < j_{k+1}$ .

① If neither  $A[i_k]$  or  $A[j_k]$  are being inserted into the sorted portion in the  $k$ th iteration (either both already in sorted portion, or neither)

either  $i_{k+1} = i_k$  and  $j_{k+1} = j_k$

OR  $i_{k+1} = (i_k) + 1$  and  $j_{k+1} = (j_k) + 1$  (both shifted down 1 spot)

```

1. for j = 2 to n:
2.   Key ← A[j]
3.   i ← j - 1
4.   while i > 0 and A[i] > Key:
5.     A[i+1] ← A[i]
6.     i ← i - 1
7.   A[i+1] ← Key
  
```

here  $i$  &  $j$  are different from  $i$  &  $j$

(use inductive hypothesis to say  $i_k < j_{k+1}$ )

② If  $A[i_k]$  is in the sorted portion and  $A[j_k]$  is NOT being inserted in the  $k$ th iteration then:

$i_{k+1} = \begin{cases} (i_k) + 1 & \text{or} \\ i_k \end{cases}$   $j_{k+1} = j_k$

either shifted down by 1 or stays stays

(Termination):

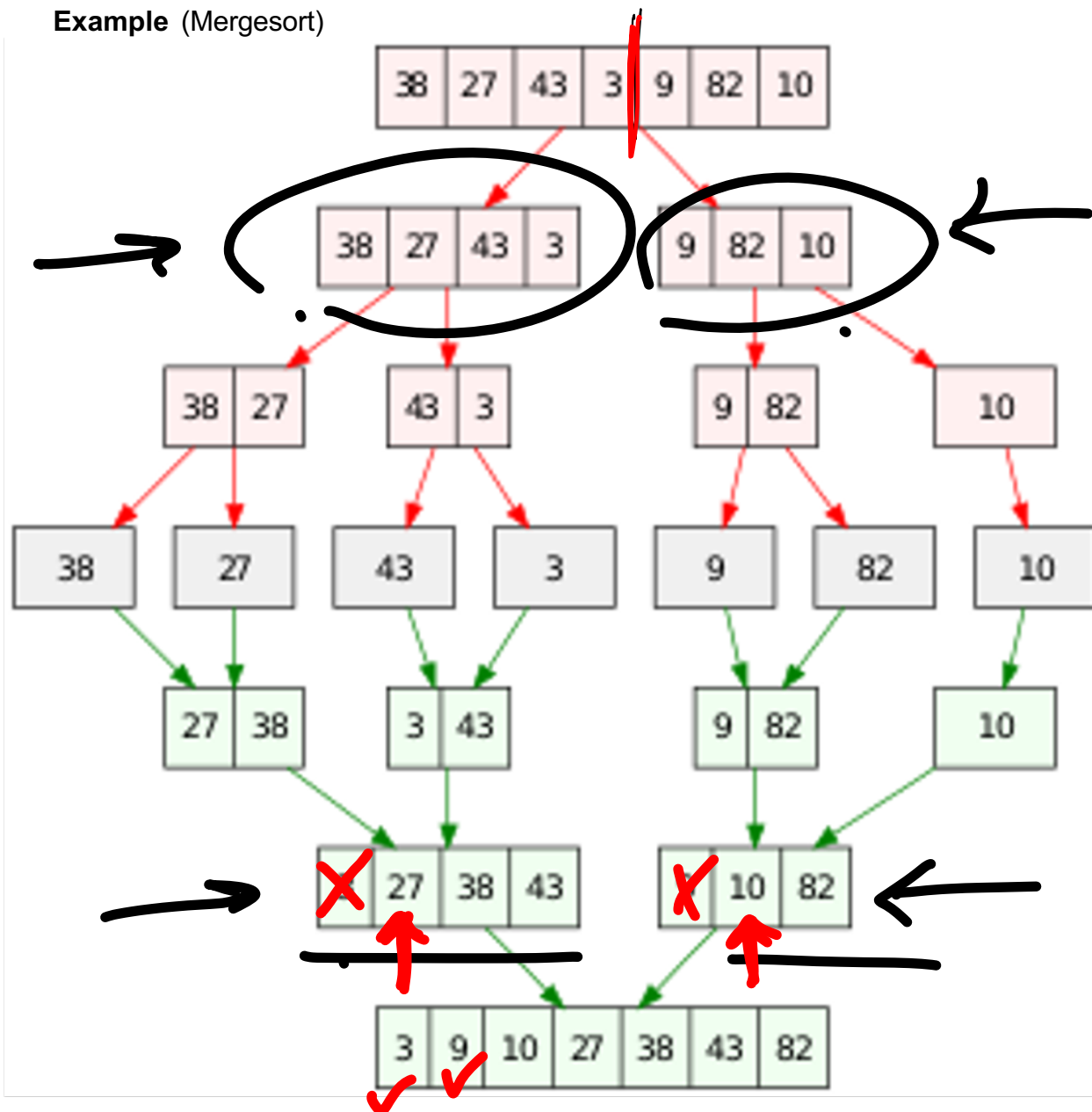
③ If  $A[j_k]$  is being inserted, then by the while loop condition on line 4,

Note: by the inductive hypothesis, couldn't have  $A[j_k]$  inserted and not  $A[i_k]$ .

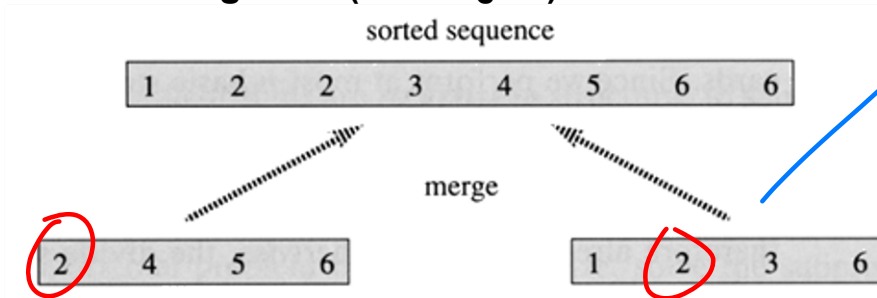
## 6 Divide and Conquer

- Divide the problem into subproblems similar to original but smaller in size
- Conquer each subproblem recursively
- Combine solutions to subproblems

**Example** (Mergesort)



## 6.1 Mergesort (CLRS §2.3)



repeated  
elements &  
stability-  
you'll prove in  
HW 2.

MERGESORT( $A[1 \dots n]$ )

MERGESORT( $A[1 \dots \frac{n}{2}]$ )

MERGESORT( $A[\frac{n}{2} + 1 \dots n]$ )

MERGE  $A[1 \dots \frac{n}{2}]$  and  $A[\frac{n}{2} + 1 \dots n]$

### 6.1.1 The Merge Subroutine

*Description of the algorithm, or more English version of pseudocode:*

To merge sorted arrays  $L[1 \dots m]$  and  $R[1 \dots p]$  into array  $C[1 \dots m+p]$

Maintain a current index for each list, each initialized to 1

While both lists have not been completely traversed:

Let  $L[i]$  and  $R[j]$  be the current elements

Copy the smaller of  $L[i]$  and  $R[j]$  to  $C$

Advance the current index for the array from which the smaller element was selected

EndWhile

Once one array has been completely traversed, copy the remainder of the other array to  $C$

$m = L.length$   
 $p = R.length$   
 MERGE( $L, m, R, p, C$ ) //  $L[1..m]$  and  $R[1..p]$  are sorted arrays

1  $i \leftarrow 1$

2  $j \leftarrow 1$

3 for  $k = 1$  to  $m + p$

4 if  $L[i] \leq R[j]$  and  $i \leq m$

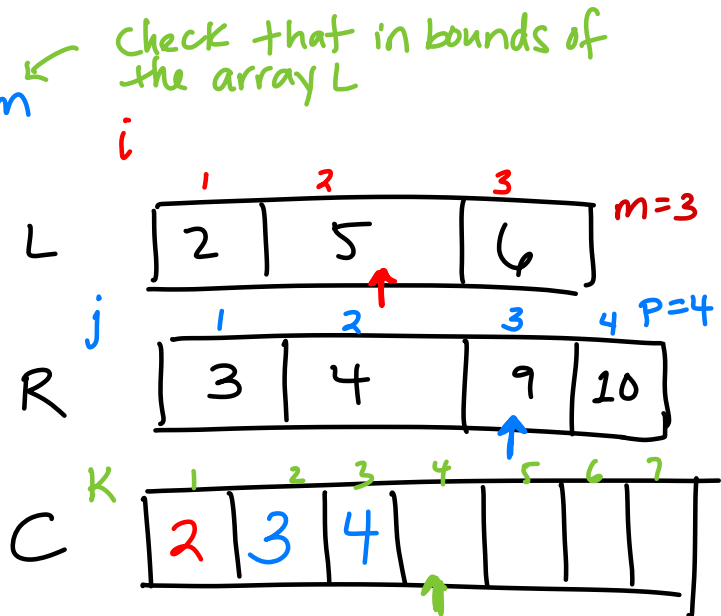
5  $C[k] \leftarrow L[i]$

6  $i \leftarrow i + 1$

7 else if  $j \leq p$

8  $C[k] \leftarrow R[j]$

9  $j \leftarrow j + 1$



### 6.1.2 Proof of Correctness of Merge

Loop invariant:

At the start of the  $k$ th iteration  $C[1..k-1]$  contains the  $k-1$  smallest elements of  $L$  and  $R$  in sorted order

These elements are from  $L[1..i-1]$  and  $R[1..j-1]$

• Initialization: (Base case)

$i = j = k = 1 \Rightarrow C$  is empty so  $C$  contains the 0 smallest elements of  $L$  and  $R$  in sorted order.

So the invariant holds.

# Desmos slide 3 : Mergesort on

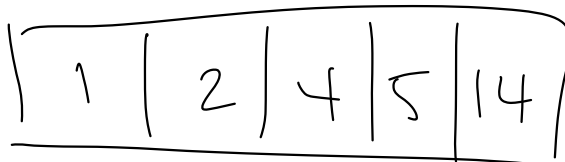
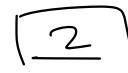
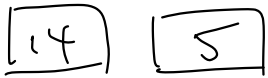
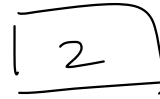
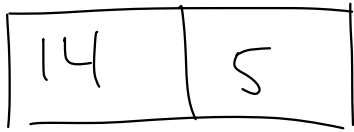
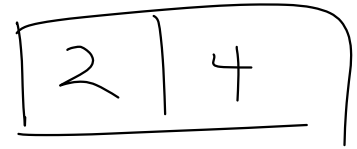
14

5

1

2

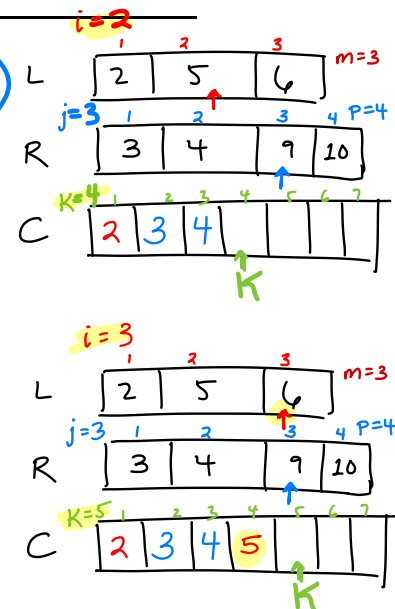
4



- Maintenance:

Assume the invariant holds for the  $k$ th iteration and (without loss of generality)  $L[i] \leq R[j]$ . Then  $L[i]$  is the smallest among elements from  $L$  and  $R$  not copied yet into  $C$ .

Therefore since  $i$  and  $K$  are incremented, the invariant holds in the next iteration.



- Termination:

After the last iteration  $K = m + p + 1$  and  $i = m + 1$ ,  $j = p + 1$ , so  $C$  contains all  $(K - 1 = m + p)$  elements of  $L$  and  $R$  in sorted order.

At the start of the  $k$ th iteration  $C[1 \dots k-1]$  contains the  $k-1$  smallest elements of  $L$  and  $R$  in sorted order.

These elements are from  $L[1 \dots i-1]$  and  $R[1 \dots j-1]$ .

```

MERGE( $L, m, R, p, C$ )
1  $i \leftarrow 1$ 
2  $j \leftarrow 1$ 
3 for  $k = 1$  to  $m + p$ 
4   if  $L[i] \leq R[j]$  and  $i \leq m$ 
5      $C[k] \leftarrow L[i]$ 
6      $i \leftarrow i + 1$ 
7   else if  $j \leq p$ 
8      $C[k] \leftarrow R[j]$ 
9      $j \leftarrow j + 1$ 

```

### 6.1.3 Running Time of Merge (worst case)

$\Theta(m + p)$   $m, p$  - sizes of subarrays of  $A[1 \dots n]$

$\Theta(n)$  complexity for  $A[1 \dots n]$



so far just did merge

#### 6.1.4 Back to Mergesort: Correctness, Running Time, Recursion Tree

correctness follows from correctness of Merge and using induction on the size of the array.  
(in general use induction to prove correctness of a recursive algorithm)

Running time: use recurrence - describe running time in terms of running time on smaller inputs.

let  $T(n)$  be the running time on a problem of size  $n$ .

$$T(n) = \underbrace{2T\left(\frac{n}{2}\right)}_{\text{recursion}} + \underbrace{cn}_{\text{merge}} + \underbrace{c_1}_{\text{divide}}$$

(really  $\lfloor \frac{n}{2} \rfloor$  and  $\lceil \frac{n}{2} \rceil$ )  
 floor                      ceiling

OR  $T(n) = \begin{cases} c & \text{if } n=1 \\ 2T\left(\frac{n}{2}\right) + cn & \text{if } n>1 \end{cases}$

for simplicity

number of subproblems

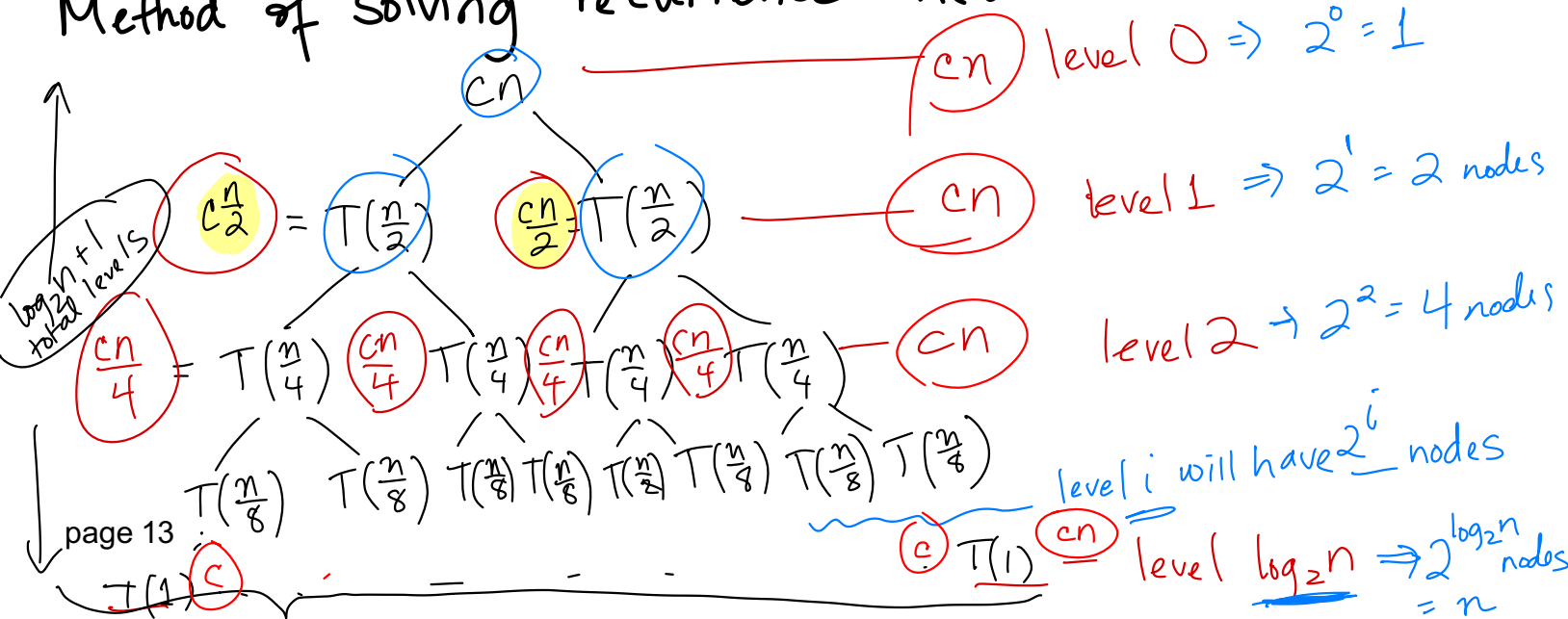
size of subproblem

$\theta(n)$

(informal but helpful)

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

Method of solving recurrence: Recursion tree expansion.



$$n \text{ leaves } (= 2^{\log_2(n)})$$

Note the depth is  $\boxed{\log_2 n + 1}$  since we start at level 0 and go up to level  $\log_2(n)$

[For  $\log_2(n)$  we also use  $\lg(n)$ ]



Merge Sort is better than insertion Sort.

$$\log(n) \leq n \text{ for large enough } n$$

$$n \log(n) \leq n^2$$

So overall runtime is:  $T(n) = (\log_2(n) + 1) * cn = cn \log_2 n + cn$

$$= \Theta(n \log_2 n)$$

Proof of time complexity

Claim. For large enough  $c_1 > 0$ , and for all  $n \geq 2$   $T(n) \leq c_1 n \log n$  (\*) (worst case and average case)

Proof sketch.

to be determined

Base case:  $n=2$ 

$$T(2) = 2T\left(\frac{2}{2}\right) + c \cdot 2 = 2T(1) + c \cdot 2$$

$$= 2c + c \cdot 2$$

$$= 4c$$

 $n=2$  in the right hand side of (\*)

$$c_1 n \log n = c_1 \cdot 2 \log_2 2$$

$$= c_1 \cdot 2 = 2c_1$$

$$4c \leq 2c_1 \quad (\text{Take } c_1 \geq 2c)$$

Assume by induction that (\*) holds for  $n < k$ .Then for  $n=k$ :

$$T(k) = 2T\left(\frac{k}{2}\right) + c \cdot k$$

$$\leq 2c_1 \frac{k}{2} \log_2\left(\frac{k}{2}\right) + ck$$

$$= c_1 k \log_2\left(\frac{k}{2}\right) + ck$$

$$= c_1 k (\log_2 k - \log_2 2) + ck$$

$$= c_1 k (\log_2 k - 1) + ck$$

$$= c_1 k \log_2 k - c_1 k + ck$$

$$= c_1 k \log_2 k - k(c_1 - c)$$

$$\leq c_1 k \log_2 k$$

using the inductive hypothesis on

$$T\left(\frac{k}{2}\right) \leq c_1 \frac{k}{2} \log_2\left(\frac{k}{2}\right)$$

$$c_1 \geq 2c$$

$$\Rightarrow c_1 - 2c \geq 0$$

$$c_1 - c - c \geq 0$$

$\therefore T(k) = O(k \log k)$  (similarly prove lower bound to show  $\Theta$ )

Show

$$T(n) = \Omega(n \log n)$$

$$\text{i.e. } T(n) \geq d n \log n \quad \text{for } n \text{ suff large}$$

---

$d > 0$

$$T(K) = 2T\left(\frac{K}{2}\right) + cn$$

$$\geq \quad \quad \quad \rightarrow$$

## 7 Solving Recurrences

We are exploring the algorithm design technique known as **Divide and Conquer**. We'll see various algorithms that use this technique.

Running time analysis of such algorithms naturally involves *recurrences* since we may state the running time in terms of the running time on smaller inputs. Let's think more about solving recurrences to help us determine the running time of such algorithms!

[HW2 Q4]

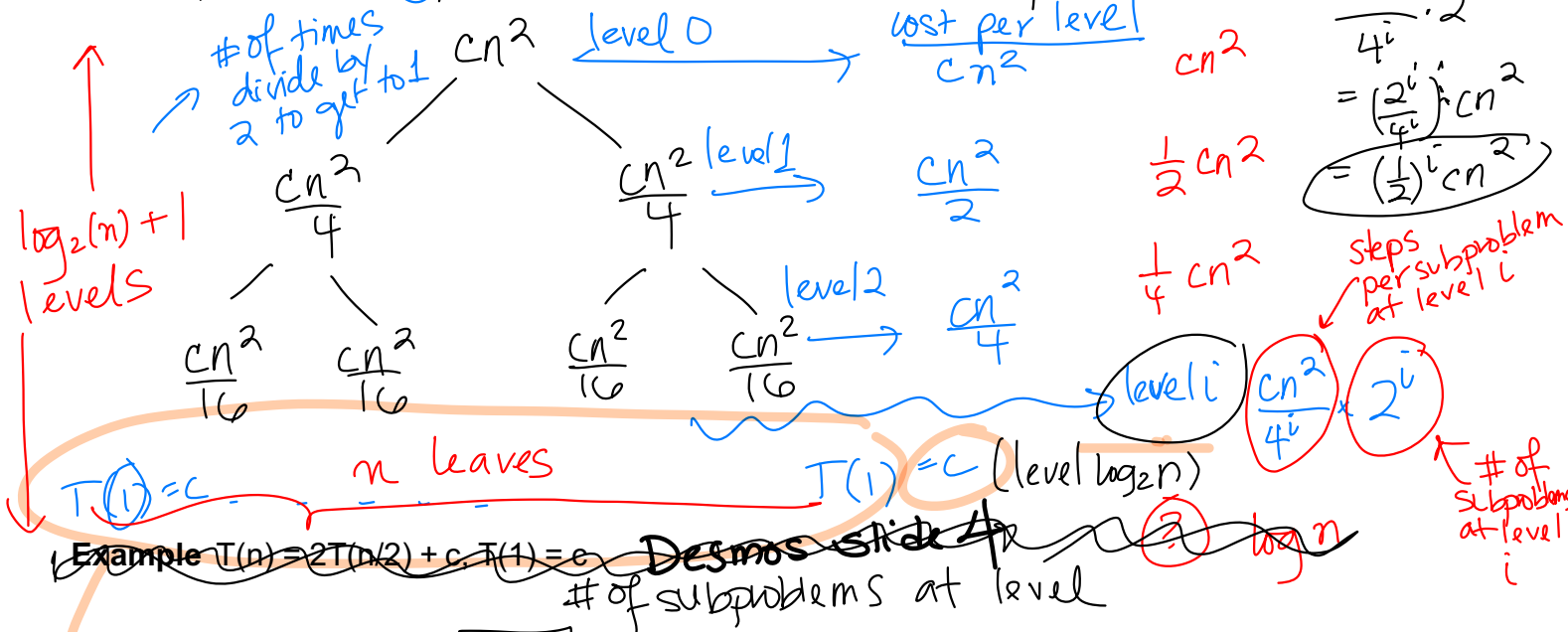
### 7.1 Using the Recursion Tree Method (CLRS §4.4)

**Example**  $T(n) = 2T(n/2) + cn^2$ ,  $T(1) = c$

Desmos slide 3

(?)  $n^2 \log n$

$$T\left(\frac{n}{2}\right) = 2T\left(\frac{n}{4}\right) + c\left(\frac{n}{2}\right)^2 = 2T\left(\frac{n}{4}\right) + c\frac{n^2}{4}$$



$$\frac{cn^2}{4^i} \times 2^i = \left(\frac{1}{2}\right)^i cn^2 \Rightarrow \text{Total steps across all levels}$$

Steps per subproblem at level  $i$

$$= \sum_{i=0}^{\log_2(n)-1} \left(\frac{1}{2}\right)^i cn^2 + cn$$

$$= cn^2 \sum_{i=0}^{\log_2(n)-1} \left(\frac{1}{2}\right)^i + cn$$

Sum of Geometric sequence  $\frac{1}{1-r}$   $r = 1/2$

$$\leq cn^2 \left(1 + \frac{1}{2} + \frac{1}{4} + \dots\right) = cn^2 \frac{1}{1-1/2} = 2cn^2 + cn = \Theta(n^2)$$

work at the leaves:

$$\left(\frac{1}{2}\right)^i cn^2 \text{ when } i = \log_2(n)$$

$$= \left(\frac{1}{2}\right)^{\log_2 n} cn^2 = n^{-1} \cdot cn^2 = cn$$

## 7 Solving Recurrences

We are exploring the algorithm design technique known as **Divide and Conquer**. We'll see various algorithms that use this technique.

Running time analysis of such algorithms naturally involves *recurrences* since we may state the running time in terms of the running time on smaller inputs. Let's think more about solving recurrences to help us determine the running time of such algorithms!

### 7.1 Using the Recursion Tree Method (CLRS §4.4)

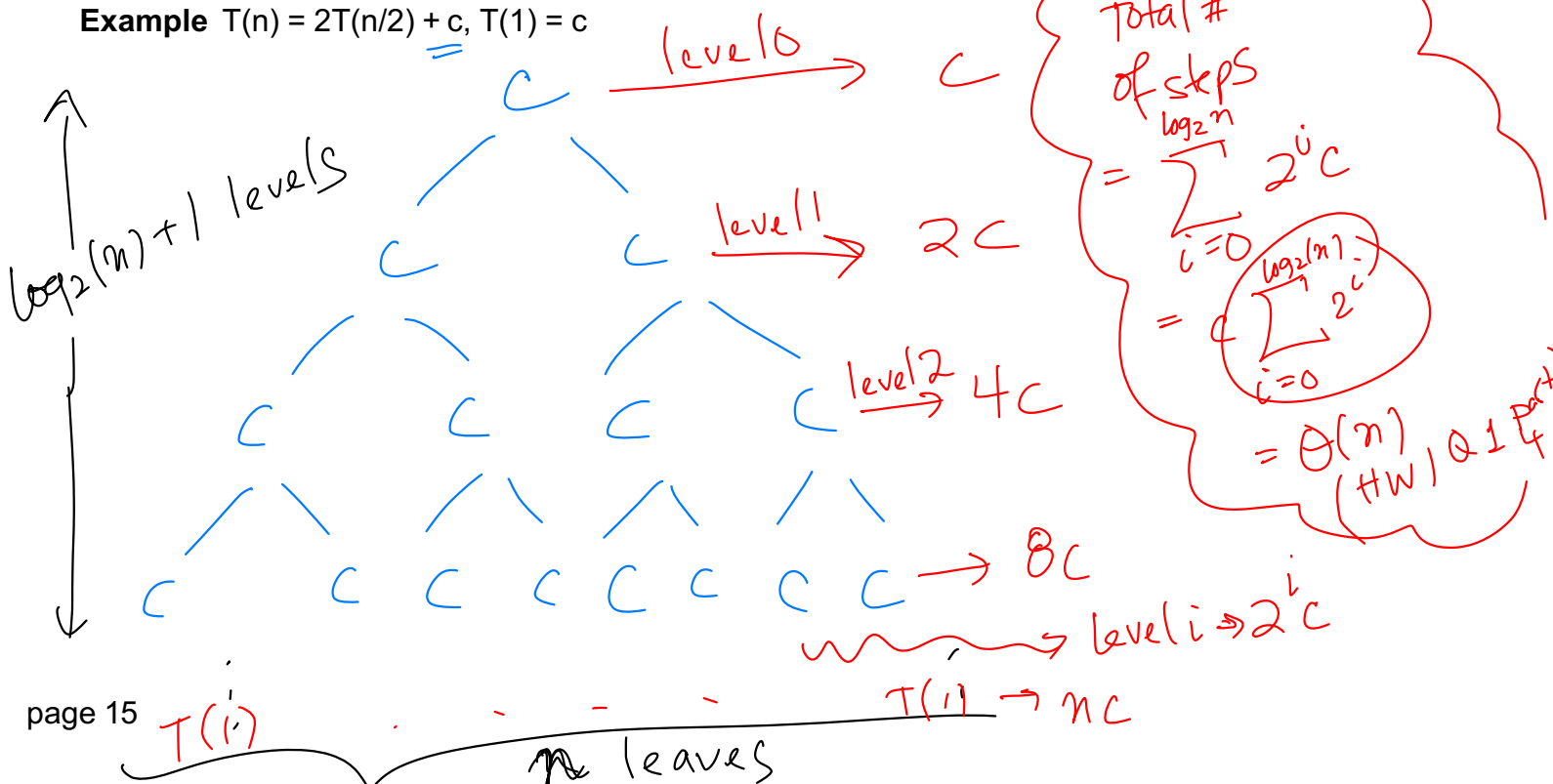
**Example**  $T(n) = 2T(n/2) + cn^2, T(1) = c \rightarrow T(n) = \Theta(n^2)$

$$\underline{c} \rightarrow T(n) = \Theta(n)$$

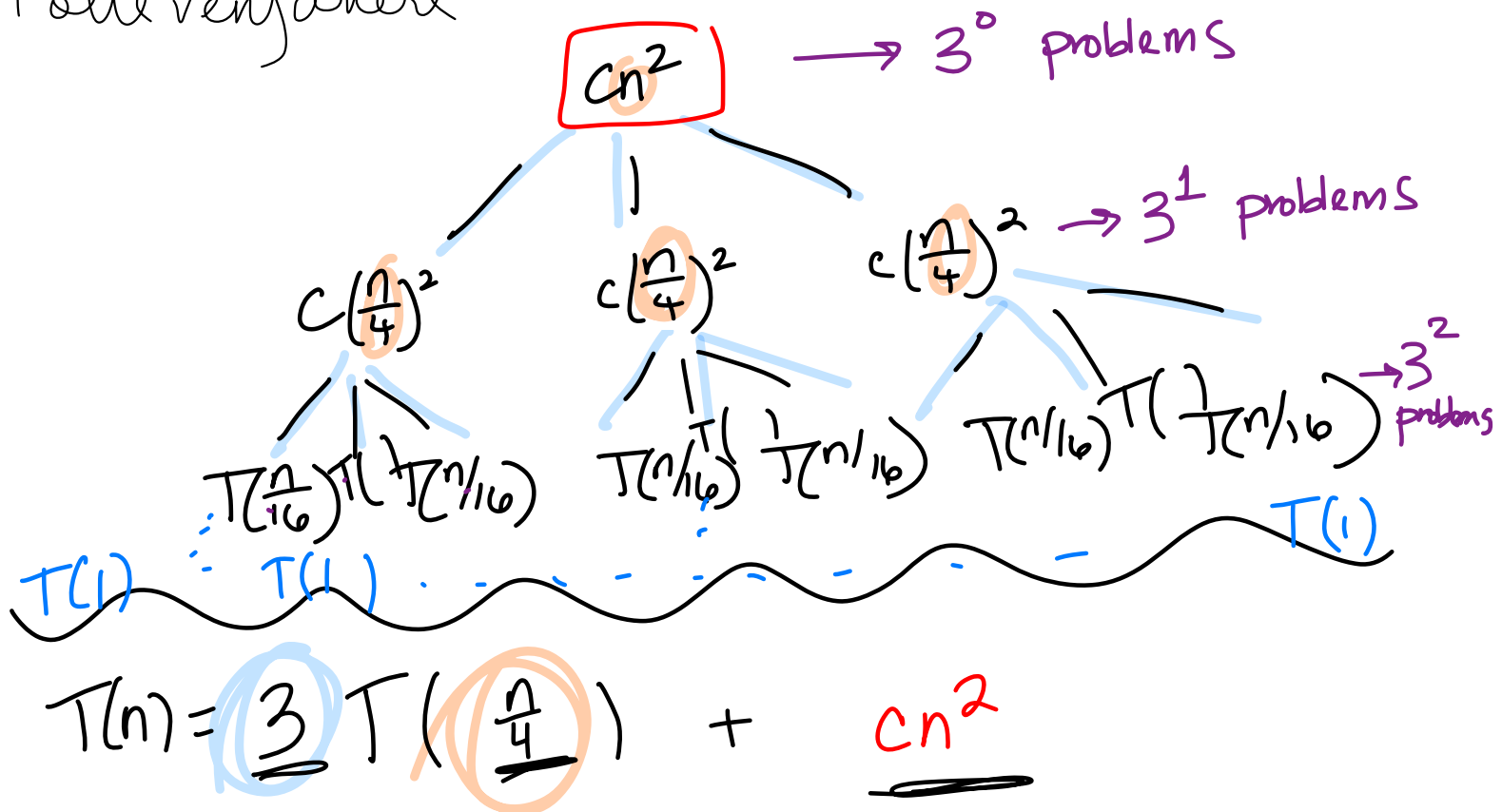
$$\underline{cn} \rightarrow T(n) = \Theta(n \log(n))$$

$$r \neq 1 \quad \sum_{k=0}^n r^k = \left( \frac{1-r^{n+1}}{1-r} \right)$$

**Example**  $T(n) = 2T(n/2) + c, T(1) = c$



Polle everywhere



How many leaves do we end up with?

① What is the last level?

$$\log_4 n$$

$\frac{n}{4^i}$  is the size of the subproblem at level  $i$

$$\frac{n}{4^d} = 1$$

$$n = 4^d \Rightarrow d = \log_4 n$$

$d = \text{last level}$

② At level  $i$ , we have  $3^i$  problems

so at level  $\log_4 n$ , we have  $3^{\log_4 n}$  problems (how many leaves we have)

equal to  $n^{\log_4 3}$

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

↑ helpful for  
analyzing  
 $\Theta$  bounds  
(big-oh)

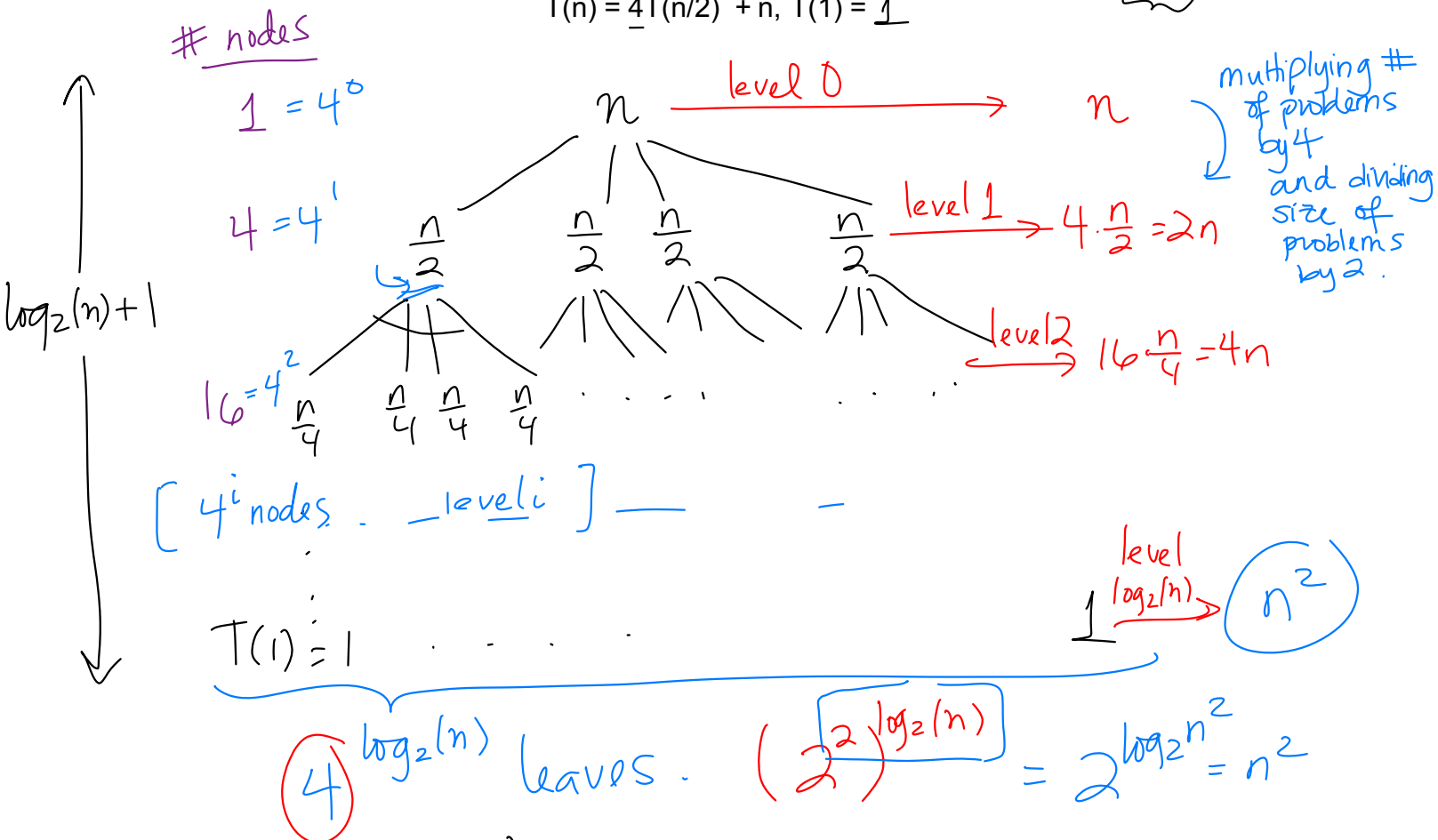
## 7.2 The substitution method (CLRS §4.3)

Example

$$T(n) = 4T(n/2) + n, T(1) = 1$$

$$T\left(\frac{n}{2}\right) = 4T\left(\frac{n}{4}\right) + \frac{n}{2}$$

$$\downarrow T\left(\frac{n}{4}\right) = 4T\left(\frac{n}{16}\right) + \frac{n}{4}$$

Guess  $T(n) = \Theta(n^2)$ Attempt 1:  $T(n) \leq cn^2$  for  $c > 0, n \geq 2$ Base case:  $n=2$ . using the recurrence,  $T(2) = 4T(1) + 2 = 4 + 2 = 6$  $T(n) \leq cn^2 \rightarrow$  for  $n=2$   $6 \leq 4c$ , so pick  $c \geq 3/2$ Inductive step: Suppose  $T(n) \leq cn^2$  for  $2 \leq n < k$ 

Consider

$$T(k) = 4T\left(\frac{k}{2}\right) + k$$

$$\leq 4c\left(\frac{k}{2}\right)^2 + k$$

$$= ck^2 + k$$

by the inductive hypothesis

$$= \Theta(k^2)$$

WRONG!

We needed to show

$$T(k) \leq ck^2$$

$$\checkmark CK^2 + K = \Theta(K^2) \leftarrow \text{not in inductive step } P(\text{sad face})$$

Note: We need the exact form of the inequality to complete the induction, because  $C$  cannot depend on  $K$ .

Attempt 2 Instead try to show  $T(n) \leq cn^2 - dn$ , for some  $c, d > 0$ . (if we pick  $d=1$ ,  $6 \leq 4c-2 \Rightarrow c \geq 3$ )

This shows  $T(n) = \Theta(n^2)$

Base case:  $n=2$   $T(2) = 6 \leq c \cdot 2^2 - d \cdot 2$  we need  $6 \leq 4c - 2d$

Suppose  $T(n) \leq cn^2 - dn$  for  $2 \leq n < K$

Consider  $T(K) = 4T(\frac{K}{2}) + K$

$$\begin{aligned} &\leq 4 \left( c \left( \frac{K}{2} \right)^2 - d \left( \frac{K}{2} \right) \right) + K \quad \leftarrow \text{by the inductive hypothesis} \\ &= cK^2 - (2d-1)K \\ &\leq cK^2 - dK \quad \downarrow \text{true as long as } (2d-1) \geq d \text{ or } (d \geq 1) \end{aligned}$$

we'll choose  $c$  &  $d$  appropriately later

So e.g. if  $\underline{c=2}$  and  $d=1$  (for base case), then  $T(K) \leq cK^2 - dK$ . So by induction,  $T(n) \leq cn^2 - dn$ .

### 7.2.1 Careful with asymptotic notation and bogus proofs! [HW 3]

①  $T(n) = 2T(\frac{n}{2}) + n$

Mergesort Runtime  
 $T(n) = \Theta(n \log n)$

"Claim"  $T(n) = O(n)$

"proof" (inductive step)  $T(K) = 2T(\frac{K}{2}) + K$  (Assume true for  $n \leq K$ )  
 $= 2O(K) + K = O(K)$  **WRONG!!**

② "Claim"  $n = O(1)$  means  $n \leq c \cdot 1$  for some  $c > 0$

Base case  $n=1$   $1 \leq c \cdot 1$  for  $c=1$   $\rightarrow (K \leq c \cdot 1)$

Assume true for  $n=K > 1$  ( $K = O(1)$ ) for some  $c > 0$

Then  $K+1 = O(1) + 1 = O(1)$  ✓

$K+1 \leq c$ ?  
( $K \leq c-1$ )

Moral: The final constants in proving the inductive step need to be the same as the constants in the inductive hypothesis.



Try to use  $\begin{cases} \cdot x \leq \lceil x \rceil \leq x+1 \\ \cdot x-1 \leq \lfloor x \rfloor \leq x \end{cases}$   
 (ceiling of  $\frac{n}{2}$  to get an integer)

## 7.2.2 Ceilings and floors

[HW3 Q2]

Example

$$T(n) = T(\lceil n/2 \rceil) + 1, T(1) = 1$$

non-recursive work

one subproblem of size roughly  $1/2$  of original

$$T(n) = \begin{array}{c} 1 \\ / \\ T(n/2) \end{array} = \begin{array}{c} 1 \\ / \\ 1 \end{array} = \dots = \begin{array}{c} 1 \\ / \\ 1 \\ / \\ 1 \end{array} \left. \vphantom{\begin{array}{c} 1 \\ / \\ T(n/2) \end{array}} \right\} \log_2(n) + 1$$

We'll show  $T(n) = O(\log(n))$  by showing  $T(n) \leq C \log(n)$  for sufficiently large  $n$ .

Attempt (inductive step) Assume  $(*)$  true for  $n < k$ .

$$T(k) = T(\lceil k/2 \rceil) + 1$$

$$\text{Claim: } T(k) \leq C \log_2(k) + 1$$

using the inductive hypothesis.

$$\text{Q: what is an algorithm that } T(k) \text{ could represent the runtime of?}$$

$$\begin{aligned} &\leq C \log_2\left(\frac{k}{2} + 1\right) + 1 \\ &= C \log_2(k+2) + 1 \end{aligned}$$

$$= C(\log_2(k+2) - \log_2(2)) + 1$$

$$= C \log_2(k+2) - C + 1$$

can't finish inductive step because this is not the same form as inductive hypothesis.

$$\text{need } \leq C \log_2(k)$$

If we had  $C \log_2(k) - C + 1$  instead we could say it's  $\leq C \log_2(k)$  for  $C \geq 1$

## 7.2.2 Ceilings and floors

## Example

$$T(n) = T(\lceil n/2 \rceil) + 1, T(1) = 1$$

$$\left( \begin{array}{l} \text{we'll} \\ \text{use} \end{array} \cdot x \leq \lceil x \rceil \leq \underline{\underline{x+1}} \right)$$

We'll show  $T(n) = O(\log(n))$  by showing  $T(n) \leq C \log(n-2)$  for  $n$  suff. large.  $C > 0$

Attempt 2 Base case:

$$n=4 \quad T(4) = 3 \quad (\text{using the recurrence + base case})$$

$$C \log_2(4-2) = C \quad \text{We need } 3 \leq C \text{ in order for the base case to hold.}$$

(inductive step) Assume  $(*)$  true for  $4 \leq n < K$ .

$$T(K) = T(\lceil \frac{K}{2} \rceil) + 1$$

$$\leq C \log_2 \lceil \frac{K}{2} - 2 \rceil + 1 \quad \text{using the inductive hypothesis.}$$

$$\leq C \log_2 \left( \frac{K}{2} - 1 \right) + 1 \quad \text{using } \lceil \frac{K}{2} - 2 \rceil \leq \frac{K}{2} - 2 + 1$$

$$= C \log_2 \left( \frac{K-2}{2} \right) + 1 = C (\log_2(K-2) - \log_2(2)) + 1 = C \log_2(K-2) - C + 1$$

$$\leq C \log_2(K-2) \text{ as long as } C \geq 1$$

Take  $C=3$ . This completes the induction  $\square$

[We've used the fact that  $\log_2(n-2) = O(\log_2(n))$ ]

# Substitution method for solving recurrences

on Desmos  
last time.

Try using substitution to show  $T(n) \leq cn^2$  for the  
recurrence  $T(n) = 4T\left(\frac{n}{2}\right) + n^2$ .

(it doesn't work out!)

Inductive step: Inductive hypothesis  
for  $2 \leq n \leq k$  assume  $T(n) \leq cn^2$

$$T(k) = 4T\left(\frac{k}{2}\right) + k^2$$

$$\leq 4c\left(\frac{k}{2}\right)^2 + k^2$$

$$= (c+1)k^2 = O(k^2) \checkmark$$

does not complete  
the induction

In fact,  
not true  
in this  
case

## Bogus proofs

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

claim:  $T(n) = O(n)$

$$\text{proof: } T(k) = 2T\left(\frac{k}{2}\right) + k = 2O(k) + k = O(k)$$

Hm... that doesn't seem right, especially since we said  
 $T(n) = \Theta(n \log n)$ . What's wrong with the proof above?

FAKE

HOAX

FRAUD

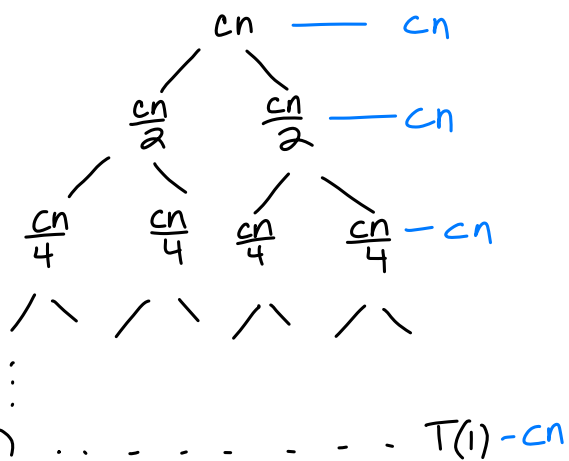
SCAM

PHONY

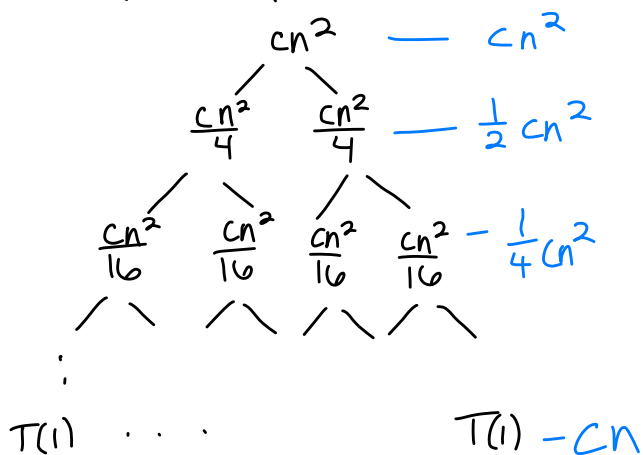
RIP-OFF

BOGUS

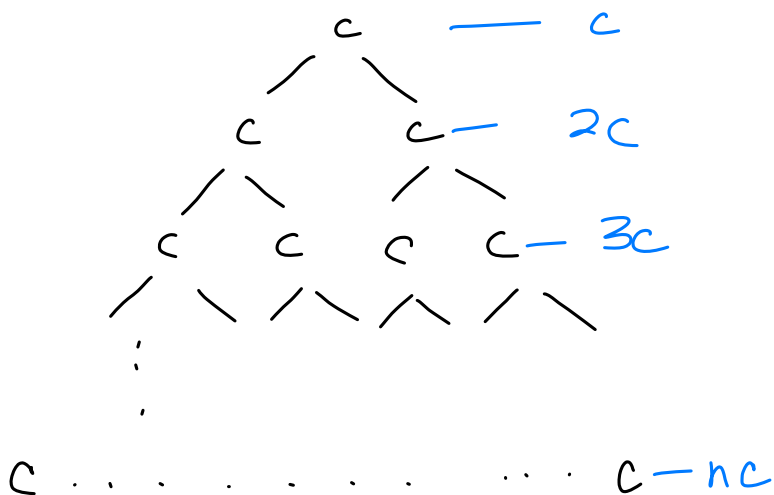
$$T(n) = 2T(n/2) + cn$$



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



$$T(n) = 2T(n/2) + c$$



### 7.3 Master Theorem (CLRS §4.5)

Let  $a \geq 1$  and  $b > 1$  be constants,  $f(n)$ ,  $T(n)$  a function defined on natural numbers by the recurrence:

$$T(n) = aT(n/b) + f(n)$$

Subproblem size at level  $k$  is  $n/b^k$  and  $a^k$  total subproblems

$n^{\log_b a}$  vs  $f(n)$

$$f(n) = \Theta(g(n)) \Rightarrow f(n) = O(g(n)) \text{ AND } f(n) = \Omega(g(n))$$

We interpret  $n/b$  to mean either  $\lceil n/b \rceil$  or  $\lfloor n/b \rfloor$ . Then,  $T(n)$  has the following asymptotic bounds:

1. If  $f(n) = O(n^{\log_b a - \epsilon})$  for some constant  $\epsilon > 0$ , then  $T(n) = \Theta(n^{\log_b a})$ . *Leaves dominate.*
2. If  $f(n) = \Theta(n^{\log_b a})$ , then  $T(n) = \Theta(n^{\log_b a} \log n)$ . *balanced across all levels*
3. If  $f(n) = \Omega(n^{\log_b a + \epsilon})$  for some constant  $\epsilon > 0$ , and  $af(n/b) \leq cf(n)$  for some constant  $c < 1$  and all sufficiently large  $n$ , then  $T(n) = \Theta(f(n))$ . *Root dominates*

problem size is  $n$

level 0

$f(n)$

problem size is  $n/b$

$f(n/b)$

$f(n/b)$

$f(n/b)$

level 1  $a f(n/b)$

problem size  $n/b^2$

$f(n/b^2)$

$f(n/b^2)$

$f(n/b^2)$

level 2  $a^2 f(n/b^2)$

(at level  $i$ ,  $a^i$  total nodes)

problem size is  $n/b^i$

at last level:  
 $\frac{n}{b^l} = 1 \Rightarrow l = \log_b n$

level  $\log_b n$

$\Theta(1)$  ...  $\Theta(1)$

$a^{\log_b n} = \# \text{ of leaves}$

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

internal nodes.

Total:

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

Root

$$= f(n)$$

$\log_b n - 1$

$$\sum_{i=1} a^i f\left(\frac{n}{b^i}\right)$$

leaves

$$+ c n^{\log_b a}$$

Desmos slides 3,4,5  
Day 8

**Example** Use the Master theorem to solve the following:

1.  $T(n) = 9T(\lceil n/3 \rceil) + n$
2.  $T(n) = T(\lceil 2n/3 \rceil) + 1$
3.  $T(n) = 2T(\lceil n/2 \rceil) + n^2$

$$\frac{2n}{3} = \frac{n}{3/2}$$

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

What is an example of a recurrence that does not fit the form of the Master theorem?

①  $a = 9, b = 3, f(n) = n$

$$n^{\log_3 9} = n^2$$

Case 1

$$f(n) = O(n^{2-\epsilon}) \text{ for some } \epsilon > 0$$

$$\left( \begin{array}{l} \text{for example } \epsilon = 1 \\ n = O(n) \end{array} \right)$$

Conclusion:  $T(n) = \Theta(n^2)$

②  $a = 1, b = \frac{3}{2}, f(n) = 1$

$$n^{\log_{3/2} 1} = n^0 = 1$$

Case 2

$$f(n) = \Theta(1)$$

Conclusion:  $T(n) = \Theta(\log n)$

③  $a = 2, b = 2, f(n) = n^2$

$$n^{\log_2 2} = n$$

Case 3

$$f(n) = \Omega(n^{1+\epsilon}) \text{ for some } \epsilon > 0$$

$$\left( \begin{array}{l} \text{for example } \epsilon = 1 \\ n^2 = \Omega(n^2) \end{array} \right)$$

Also check:

$$a \cdot f\left(\frac{n}{b}\right) = 2 \left(\frac{n}{2}\right)^2 = \frac{n^2}{2} \leq c f(n)$$

where  $c < 1$

$$\left( = \frac{1}{2} n^2 \leq \frac{1}{2} f(n) \text{ pick } c = \frac{1}{2} \right)$$

Conclusion:  $T(n) = \Theta(n^2)$

Master Theorem doesn't apply

1)  $T(n) = T(n-1) + 1$   
not of the form  $\frac{n}{b}$

2)  $T(n) = T(\frac{1}{3}n) + T(\frac{2}{3}n) + 1$   
not the same size subproblems

3)  $T(n) = T(\sqrt{n}) + 1$

4)  $T(n) = 2T(\frac{n}{2}) + n \log(n)$  why?  
 $T(1) = c$

$a=2$   
 $n^{\log_2 2} = n^1$

Why?  $\rightarrow$

$\begin{cases} n \log(n) = O(n^{1-\epsilon}) \checkmark \\ n \log(n) = \Theta(n) \checkmark \\ n \log(n) = \Omega(n^{1+\epsilon}) \end{cases}$

$\log(n) \neq \Omega(n^\epsilon)$  for any  $\epsilon > 0$ !

$n$  dominates  $\log(n)$  — why?

$\lim_{n \rightarrow \infty} \frac{\log(n)}{n} = 0$

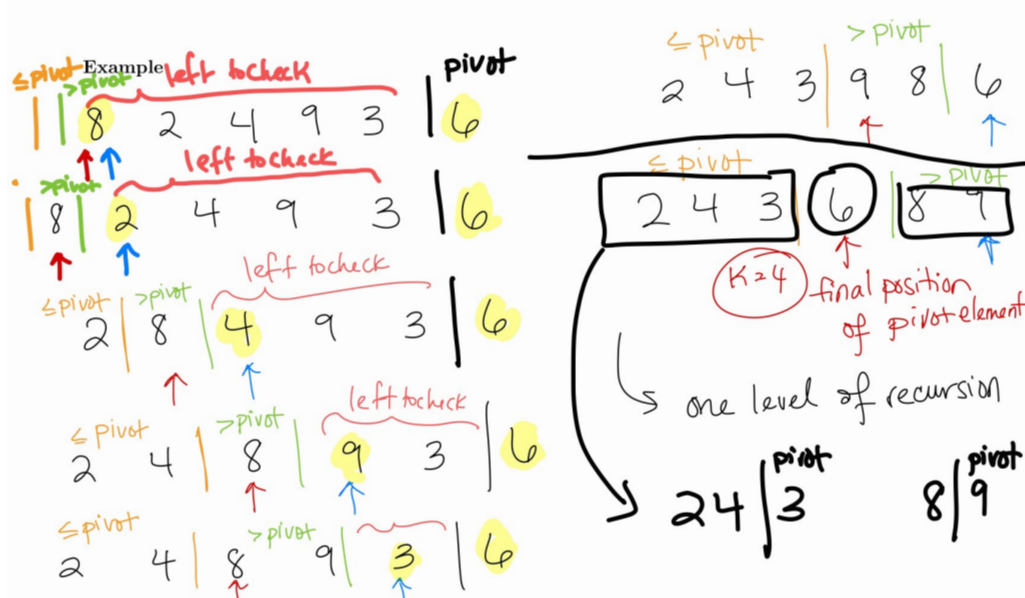
L'Hopitals

also  $\lim_{n \rightarrow \infty} \frac{\log(n)}{n^\epsilon} = 0$

## 8 Quicksort (CLRS §7.1, 7.2) (1962)

Idea: { 1) select pivot  
 Divide { 2) partition: everything smaller than pivot goes to the left, larger to the right  
 Conquer { 3) Recursively solve both halves.

### Example



Blue arrow is j  
 Red arrow is i  
 Highlighted number is being compared w/ pivot

QUICKSORT(A[1...n])

- 1 k = PARTITION(A)  $\rightarrow$  k is the final position of the pivot after partition
- 2 QUICKSORT(A[1... k-1])
- 3 QUICKSORT(A[k+1... n])

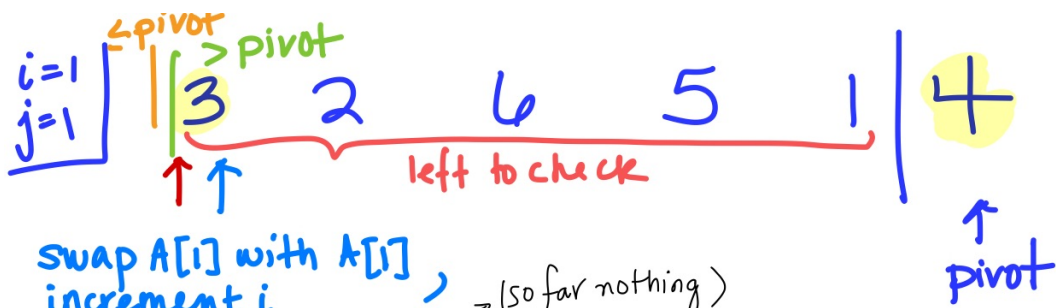
### 8.1 Partition

PARTITION(A[1...n])

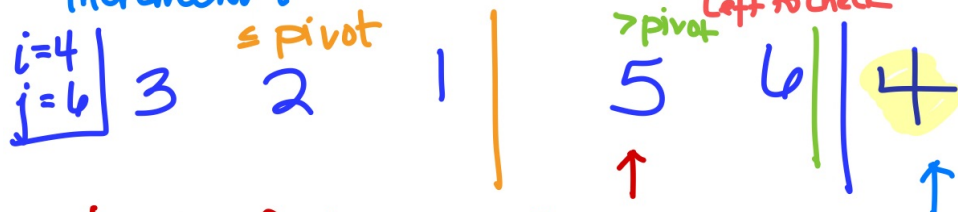
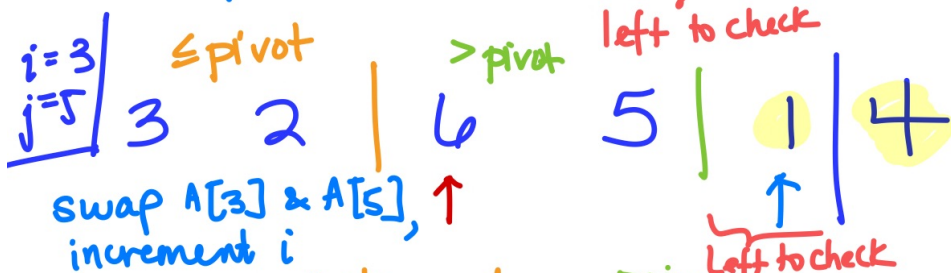
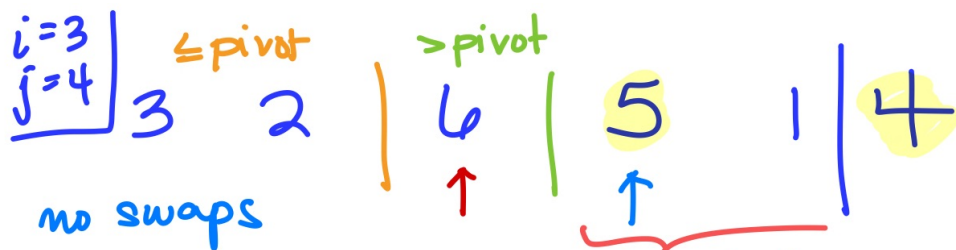
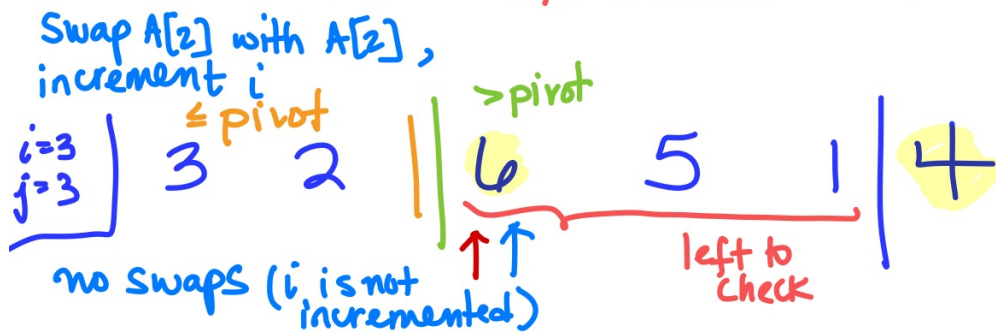
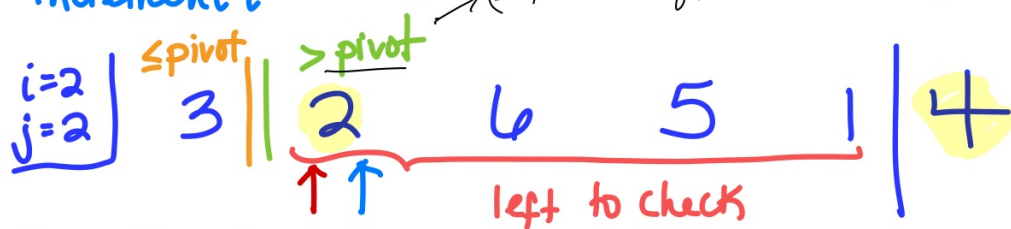
- 1 pivot  $\leftarrow$  A[n]
- 2 i  $\leftarrow$  1  $\rightarrow$  index of the start of elements greater than pivot
- 3 for j = 1 to n-1
- 4     if A[j] ≤ pivot
- 5         Swap A[i] with A[j]
- 6         i  $\leftarrow$  i + 1
- 7 Swap A[i] with A[n]
- 8 RETURN i

$\rightarrow$  moves a certain element "pivot" to its final location & all other elements to the left or right depending on if they are larger or smaller





Blue arrow is  $j$   
 Red arrow is  $i$   
 highlighted number is being compare w/ pivot



4 & 5 swapped after loop ends.

$i=4$  is the final position of 4 : 3 2 1 4 6 5



## 8.2 Correctness of Partition (and Quicksort)

Loop invariant: For any index  $k$

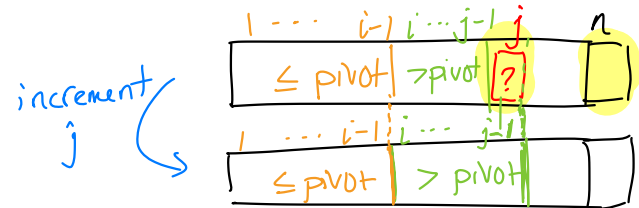
- 1) If  $k = 1, \dots, i-1$ , then  $A[k] \leq \text{pivot}$
- 2) If  $k = i, \dots, j-1$ , then  $A[k] > \text{pivot}$
- 3) If  $k = n$ , then  $A[k] = \text{pivot}$

- Initialization:  $i = 1, j = 1$   $A[k]$  is nothing so (1) & (2) hold vacuously.  
(3) is true by the assignment in line 1.

Maintenance:

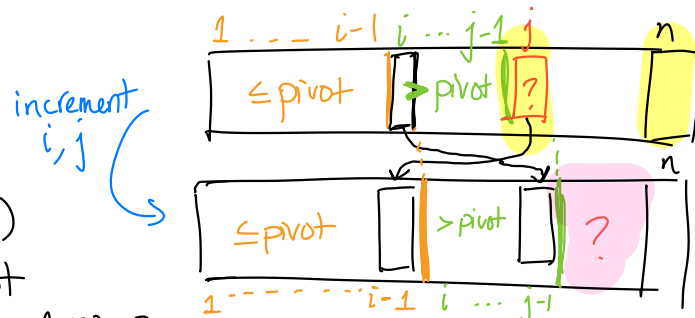
Case 1)  $A[j] > \text{pivot}$

$j \leftarrow j+1$  at the start of the next iteration  
so conditions (1) & (3) still hold and also condition (2) holds since  $A[j] > \text{pivot}$  in the previous iteration.



Case 2)  $A[j] \leq \text{pivot}$

Because of the swap  
 $A[i-1] \leq \text{pivot}$  so condition (1) still holds. Also  $A[j-1] > \text{pivot}$  since the item swapped in to  $A[j-1]$  is (by the loop invariant) greater than the pivot.  
So condition (2) still holds. Also condition (3) still holds.



Termination:

At the end of the for loop and line 7,

$A[1 \dots i-1] \leq \text{pivot}$

$A[i+1 \dots n] > \text{pivot}$

and  $A[i] = \text{pivot}$ ; i.e. every entry in the array is partitioned.

### 8.3 Running Time of Partition and Quicksort



partition : each line inside the  
loop takes constant time  
→  $\Theta(n)$ .