Lecture 2

Asymptotic Notation, Worst-Case Analysis, and MergeSort

Announcements

- Sign up for Ed!
 - Connect with course staff!
 - Connect with your colleagues!
 - Receive course announcements!
- Please (continue to) send OAE letters and exam conflicts to cs161-win2021-staff@lists.stanford.edu

Homework!

- HW1 will be released **today** (Wednesday).
- It is due the next Wednesday, 11:59pm (in one week), on Gradescope.
 - Make sure you are signed up for the Winter 2021 Gradescope
- Homework comes in two parts:
 - Exercises:
 - More straightforward.
 - Try to do them on your own.
 - Problems:
 - Less straightforward.
 - Try them on your own first, but then collaborate!
- See the website for guidelines on homework:
 - Collaboration + Late Day policy (in the "Policies" tab)
 - Best practices (in the "Resources" tab)
 - Example Homework (in the "Resources" tab)
 - LaTeX help (in the "Resources" tab)

Office Hours and Sections

- Office hours calendar is on the Canvas calendar.
 - (under "Course Summary")
 - Office hours start tomorrow
- Sections have been scheduled.
 - See Canvas calendar
 - Thu 4:30-5:30pm
 - Thu 7-8pm
 - Fri 3-4pm
 - one will be recorded
 - Don't need to formally enroll in sections, just show up!

Huang basement





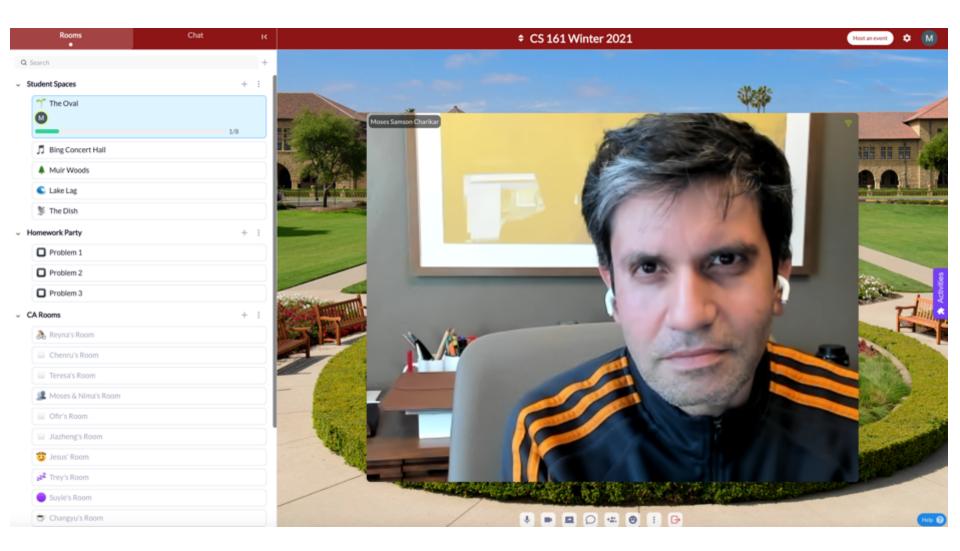


Nooks

Rooms	Chat IC	÷ CS 161 Winter 2021
Q Search	+	L 6 Active Users
 Student Spaces 	+ :	
↑ The Oval		Jesus Enriq., Caci Jiang Moses Sam.,
🞵 Bing Concert Hall		Seusening. Californing Processant
A Muir Woods		
🕒 Lake Lag		✓ Welcome to CS 161 Winter 2021 Getting Started
🔰 The Dish		★ Enter a room on the left!
 Homework Party 	+ :	★ Copy Community Link and share to invite people!
Problem 1		Notesting Started
Problem 2		Welcome! We will be using Nooks for office hours. We expect CS161 students to use this Nooks group to communicate only about the course related mater
Problem 3		promotional messages. Please do not share solutions of the course assignments here or violate honor code. Standard penalties and consequences will apply If you have questions for the teaching team, please reach out to us on Ed.
~ CA Rooms	+ :	Office Hour Schedule
🚴 Reyna's Room		The schedule for when each CA/instructor will be here to answer questions can be found on the Carwas calendar.
Chenru's Room		i ∕ Kooms
Teresa's Room		The teaching team will set up rooms for office hours in "CA Rooms." Students are encouraged to make their own rooms in "😁 tudent Spaces 🍂 so that they can collaborate with their friends and just hang out with their fellow
🕵 Moses & Nima's Room		Rules and Guidelines
Ofir's Room		1. Absolutely no harassment or abuse 2. Always be kind and welcoming to others
Jiazheng's Room		3. Have fun!
🤠 Jesus' Room		
z ^{zZ} Trey's Room		
Suyle's Room		

st

Nooks



Homework Parties

- Calendar is on Canvas.
 - Sat 3-5pm
 - Mon 8-10am and 4-6pm
 - Tue 8-10am and 3-5pm

CS 161 Student Survey

- <u>https://forms.gle/3PsgYHHRzvdHP8j36</u>
- Please respond! It will help us adapt the course to better serve you.
- Worth 1 point on Homework 1

End of announcements!

Last time

Philosophy

- Algorithms are awesome!
- Our motivating questions:
 - Does it work?
 - Is it fast?
 - Can I do better?

Technical content

- Karatsuba integer multiplication
- Example of "Divide and Conquer"
- Not-so-rigorous analysis

Cast

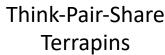




Plucky the pedantic penguin

Lucky the lackadaisical lemur







Ollie the over-achieving ostrich

Siggi the studious stork

Today

- We are going to ask:
 - Does it work?
 - Is it fast?
- We'll start to see how to answer these by looking at some examples of sorting algorithms.
 - InsertionSort
 - MergeSort



SortingHatSort not discussed

The Plan

- Sorting!
- Worst-case analyisis
 - InsertionSort: Does it work?
- Asymptotic Analysis
 - InsertionSort: Is it fast?
- MergeSort
 - Does it work?
 - Is it fast?

Sorting

- Important primitive
- For today, we'll pretend all elements are distinct.

I hope everyone did the pre-lecture exercise!

What was the mystery sort algorithm?

- 1. MergeSort
- 2. QuickSort
- 3. InsertionSort

4. BogoSort

```
def mysteryAlgorithmOne(A):
  for x in A:
    B = [None for i in range(len(A))]
    for i in range(len(B)):
        if B[i] == None or B[i] > x:
            j = len(B)-1
            while j > i:
                B[j] = B[j-1]
                j -= 1
            B[i] = x
            break
    return B
```

```
def mysteryAlgorithmTwo(A):
  for i in range(1,len(A)):
    current = A[i]
    j = i-1
    while j >= 0 and A[j] > current:
        A[j+1] = A[j]
        j -= 1
        A[j+1] = current
```

I hope everyone did the pre-lecture exercise!

What was the mystery sort algorithm?

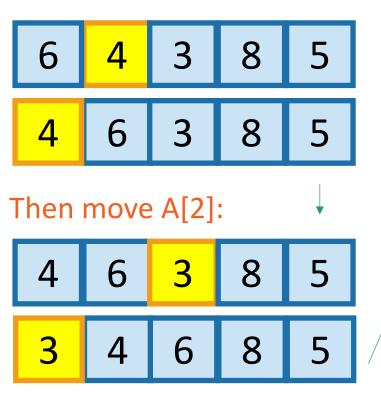
- 1. MergeSort
- 2. QuickSort
- 3. InsertionSort
- 4. BogoSort

def mysteryAlgorithmOne(A):
 for x in A:
 B = [None for i in range(len(A))]
 for i in range(len(B)):
 if B[i] == None or B[i] > x:
 j = len(B)-1
 while j > i:
 B[j] = B[j-1]
 j -= 1
 B[i] = x
 break
 return B

```
def MysteryAlgorithmTwo(A):
  for i in range(1,len(A)):
    current = A[i]
    j = i-1
    while j >= 0 and A[j] > current:
        A[j+1] = A[j]
        j -= 1
        A[j+1] = current
```

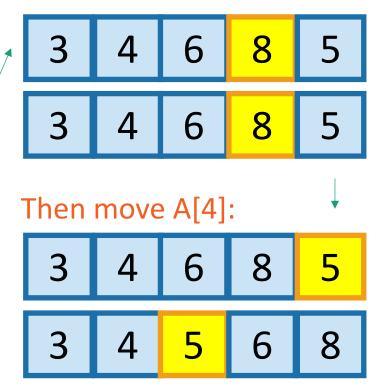
InsertionSort example

Start by moving A[1] toward the beginning of the list until you find something smaller (or can't go any further):









Then we are done!

Insertion Sort

- 1. Does it work?
- 2. Is it fast?

What does that mean???



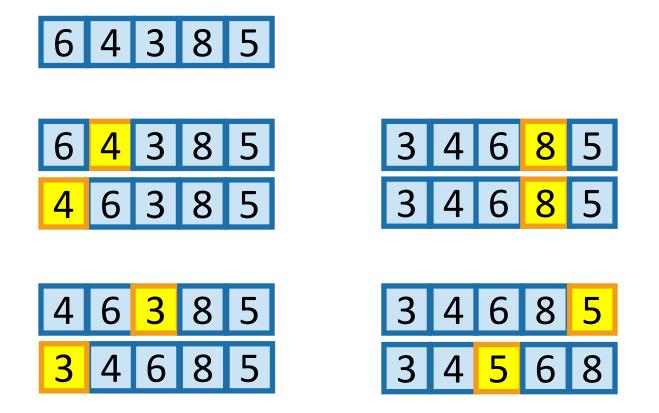
Plucky the Pedantic Penguin

The Plan

- InsertionSort recap
- Worst-case Analysis
 - Back to InsertionSort: Does it work?
- Asymptotic Analysis
 - Back to InsertionSort: Is it fast?
- MergeSort
 - Does it work?
 - Is it fast?

Claim: InsertionSort "works"

• "Proof:" It just worked in this example:



Sorted!

Claim: InsertionSort "works"

• "Proof:" I did it on a bunch of random lists and it always worked:

```
A = [1,2,3,4,5,6,7,8,9,10]
for trial in range(100):
    shuffle(A)
    InsertionSort(A)
    if is_sorted(A):
        print('YES IT IS SORTED!')
```

YESITISSORTED!YESYESITISSORTED!YESITISSORTED!YESITISSORTED!YESYESITISSORTED!YESITISSORTED!YESITISSORTED!YESYESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSO												
YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESYESITISSORTED!YESITISSORTED!YESITISSORTED!YESYESITISSORTED!YESITISSORTED!YESITISSORTED!YESYESITISSORTED!YESITISSORTED!YESITISSORTED!YESYESITISSORTED!YESYESITISSORTED!YESITISSORTED!YESYESITISSORTED!YESITISSORTED!YESITISSORTED!YESYESITISSORTED!YESITISSORTED!YESITISSORT	YES	IT	IS	SORTED!	YES	IT	IS	SORTED!	YES	IT	IS	SORTED!
YESITISSORTED1YESYESITISSORTED1YESITISSORTED1YESITISSORTED1YESYESITISSORTED1YESITISSORTED1YESITISSORTED1YESYESITISSORTED1YESITISSORTED1YESITISSORTED1YESITISSORTED1YESITISSORTED1YESITISSORTED1YESITISSO	YES	IT	IS	SORTED!	YES	IT	IS	SORTED!	YES	IT	IS	SORTED!
YESITISSORTED1YESITISSORTED1YESITISSORTED1YESITISSORTED1YESITISSORTED1YESITISSORTED1YESITISSORTED1YESITISSORTED1YESITISSORTED1YESITISSORTED1YESITISSORTED1YESITISSORTED1YESITISSORTED1YESITISSORTED1YESITISSORTED1YESITISSORTED1YESITISSORTED1YESITISSORTED1YESITISSORTED1YESYESITISSORTED1YESITISSORTED1YESITISSORTED1YESYESITISSORTED1YESITISSORTED1YESITISSORTED1YESYESITISSORTED1YESYESITISSORTED1YESITISSORTED1YESYESITISSORTED1YESYESITISSORTED1YESITISSORTED1YESYESITISSORTED1YESYESITISSORTED1YESITISSORTED1YESYESITISSORTED1YESYESITISSORTED1YE	YES	IT	IS	SORTED!	YES	IT	IS	SORTED!	YES	IT	IS	SORTED!
YESITISSORTED1YESITISSORTED1YESITISSORTED1YESITISSORTED1YESITISSORTED1YESITISSORTED1YESITISSORTED1YESITISSORTED1YESITISSORTED1YESITISSORTED1YESITISSORTED1YESITISSORTED1YESITISSORTED1YESITISSORTED1YESITISSORTED1YESITISSORTED1YESITISSORTED1YESITISSORTED1YESITISSORTED1YESYESITISSORTED1YESITISSORTED1YESITISSORTED1YESITISSORTED1YESITISSORTED1YESITISSORTED1YESITISSORTED1YESITISSORTED1YESITISSORTED1YESYESITISSORTED1YESITISSORTED1YESITISSORTED1YESYESITISSORTED1YESITISSORTED1YESITISSORTED1YESITISSORTED1YESITISSORTED1YESITISSORTED1YESITISSO	YES	IT	IS	SORTED!	YES	IT	IS	SORTED!	YES	IT	IS	SORTED!
YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESYESITISSORTED!YESITISSORTED!YESITISSORTED!YESYESITISSORTED!YESITISSORTED!YESITISSORTED!YESYESITISSORTED!YESITISSORTED!YESITISSORTED!YESYESITISSORTED!YESITISSORTED!YESITISSORTED!YESYESITISSORTED!YESITISSORTED!YESITISSORTED!YESYESITISSORTED!YESITISSORTED!YESITISSORTED!YESYESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORT	YES	IT	IS	SORTED!	YES	IT	IS	SORTED!	YES	IT	IS	SORTED!
YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESYESITISSORTED!YESITISSORTED!YESITISSORTED!YESYESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESYESITISSORTED!YESITISSORTED!YESITISSORTED!YESYESITISSORTED!YESITISSORTED!YESITISSORTED!YESYESITISSORTED!YESITISSORTED!YESITISSORTED!YESYESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!	YES	IT	IS	SORTED!	YES	IT	IS	SORTED!	YES	IT	IS	SORTED!
YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESYESITISSORTED!YESITISSORTED!YESITISSORTED!YESYESITISSORTED!YESYESITISSORTED!YESITISSORTED!YESITISSORTED!YESYESITISSORTED!YESITISSORTED!YESITISSORTED!YESYESITISSORTED!YESITISSORTED!YESYESITISSORTED!YESYESITISSORTED!YESITISSORTED!YESYESITISSORTED!YESYESITISSORTED!YESITISSORTED!YESYESITISSORTED!YESYESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED! <td>YES</td> <td>IT</td> <td>IS</td> <td>SORTED!</td> <td>YES</td> <td>IT</td> <td>IS</td> <td>SORTED!</td> <td>YES</td> <td>IT</td> <td>IS</td> <td>SORTED!</td>	YES	IT	IS	SORTED!	YES	IT	IS	SORTED!	YES	IT	IS	SORTED!
YESITISSORTED!YESYESITISSORTED!YESITISSORTED!YESITISSORTED!YESYESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED! <t< td=""><td>YES</td><td>IT</td><td>IS</td><td>SORTED!</td><td>YES</td><td>IT</td><td>IS</td><td>SORTED!</td><td>YES</td><td>IT</td><td>IS</td><td>SORTED!</td></t<>	YES	IT	IS	SORTED!	YES	IT	IS	SORTED!	YES	IT	IS	SORTED!
YESITISSORTED!YESYESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YES <t< td=""><td>YES</td><td>IT</td><td>IS</td><td>SORTED!</td><td>YES</td><td>IT</td><td>IS</td><td>SORTED!</td><td>YES</td><td>IT</td><td>IS</td><td>SORTED!</td></t<>	YES	IT	IS	SORTED!	YES	IT	IS	SORTED!	YES	IT	IS	SORTED!
YESITISSORTED!YESIT <td< td=""><td>YES</td><td>IT</td><td>IS</td><td>SORTED!</td><td>YES</td><td>IT</td><td>IS</td><td>SORTED!</td><td>YES</td><td>IT</td><td>IS</td><td>SORTED!</td></td<>	YES	IT	IS	SORTED!	YES	IT	IS	SORTED!	YES	IT	IS	SORTED!
YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESYESITISSORTED!YESITISSORTED!YESITISSORTED!YESYESITISSORTED!YESITISSORTED!YESITISSORTED!YESYESITISSORTED!YESITISSORTED!YESITISSORTED!YESYESITISSORTED!YESITISSORTED!YESITISSORTED!YESYESITISSORTED!YESITISSORTED!YESITISSORTED!YESYESITISSORTED!YESITISSORTED!YESITISSORTED!YESYESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORT	YES	IT	IS	SORTED!	YES	IT	IS	SORTED!	YES	IT	IS	SORTED!
YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESYESITISSORTED!YESITISSORTED!YESITISSORTED!YESYESITISSORTED!YESYESITISSORTED!YESITISSORTED!YESYESITISSORTED!YESYESITISSORTED!YESITISSORTED!YESYESITISSORTED!YESYESITISSORTED!YESITISSORTED!YESYESITISSORTED!YESYESITISSORTED!YESITISSORTED!YESYESITISSORTED!YESYESITISSORTED!YESITISSORTED!YESYESITISSORTED!YESYESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITIS	YES	IT	IS	SORTED!	YES	IT	IS	SORTED!	YES	IT	IS	SORTED!
YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESYESITISSORTED!YESITISSORTED!YESITISSORTED!YESYESITISSORTED!YESYESITISSORTED!YESITISSORTED!YESYESITISSORTED!YESYESITISSORTED!YESITISSORTED!YESYESITISSORTED!YESYESITISSORTED!YESITISSORTED!YESYESITISSORTED!YESYESITISSORTED!YESITISSORTED!YESYESITISSORTED!YESYESITISSORTED!YESITISSORTED!YESYESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSOR	YES	IT	IS	SORTED!	YES	IT	IS	SORTED!	YES	IT	IS	SORTED!
YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESYESITISSORTED!YESITISSORTED!YESITISSORTED!YESYESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESYESITISSORTED!YESITISSORTED!YESITISSORTED!YESYESITISSORTED!YESITISSORTED!YESITISSORTED!YESYESITISSORTED!YESITISSORTED!YESITISSORTED!YESYESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!	YES	IT	IS	SORTED!	YES	IT	IS	SORTED!	YES	IT	IS	SORTED!
YESITISSORTED!YESYESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YES <t< td=""><td>YES</td><td>IΤ</td><td>IS</td><td>SORTED!</td><td>YES</td><td>IΤ</td><td>IS</td><td>SORTED!</td><td>YES</td><td>IΤ</td><td>IS</td><td>SORTED!</td></t<>	YES	IΤ	IS	SORTED!	YES	IΤ	IS	SORTED!	YES	IΤ	IS	SORTED!
YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESYESITISSORTED!YESITISSORTED!YESITISSORTED!YESYESITISSORTED!YESITISSORTED!YESITISSORTED!YESYESITISSORTED!YESYESITISSORTED!YESITISSORTED!YESYESITISSORTED!YESYESITISSORTED!YESITISSORTED!YESYESITISSORTED!YESYESITISSORTED!YESITISSORTED!YESYESITISSORTED!YESYESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YE	YES	IT	IS	SORTED!	YES	IT	IS	SORTED!	YES	IT	IS	SORTED!
YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESYESITISSORTED!YESITISSORTED!YESITISSORTED!YESYESITISSORTED!YESITISSORTED!YESITISSORTED!YESYESITISSORTED!YESYESITISSORTED!YESITISSORTED!YESYESITISSORTED!YESYESITISSORTED!YESITISSORTED!YESYESITISSORTED!YESITISSORTED!YESITISSORTED!YESYESITISSORTED!YESITISSORTED!YESITISSORTED!YESYESITISSORTED!YESYESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YE	YES	IT	IS	SORTED!	YES	IT	IS	SORTED!	YES	IT	IS	SORTED!
YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESYESITISSORTED!YESYESITISSORTED!YESITISSORTED!YESYESITISSORTED!YESYESITISSORTED!YESITISSORTED!YESYESITISSORTED!YESYESITISSORTED!YESITISSORTED!YESYESITISSORTED!YESYESITISSORTED!YESITISSORTED!YESYESITISSORTED!YESYESITISSORTED!YESITISSORTED!YESYESITISSORTED!YESYESITISSORTED!YESITISSORTED!YESITISSORTED!YESYESITISSORTED!YESITISSORTED!YESITISSORTED!YESYESI	YES	IΤ	IS	SORTED!	YES	IΤ	IS	SORTED!	YES	IΤ	IS	SORTED!
YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESYESITISSORTED!YESITISSORTED!YESITISSORTED!YESYESITISSORTED!YESYESITISSORTED!YESITISSORTED!YESYESITISSORTED!YESYESITISSORTED!YESITISSORTED!YESYESITISSORTED!YESYESITISSORTED!YESITISSORTED!YESYESITISSORTED!YESYESITISSORTED!YESITISSORTED!YESYESITISSORTED!YESYESITISSORTED!YESITISSORTED!YESYESITISSORTED!YESYESITISSORTED!YESITISSORTED!YESYESITISSORTED!YESYESITISSORTED!YESITISSORTED!YESITISSORTED!Y	YES	IT	IS	SORTED!	YES	IT	IS	SORTED!	YES	IT	IS	SORTED!
YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESYESITISSORTED!YESITISSORTED!YESITISSORTED!YESYESITISSORTED!YESITISSORTED!YESITISSORTED!YESYESITISSORTED!YESYESITISSORTED!YESITISSORTED!YESYESITISSORTED!YESYESITISSORTED!YESITISSORTED!YESYESITISSORTED!YESITISSORTED!YESITISSORTED!YESYESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITIS<	YES	IT	IS	SORTED!	YES	IT	IS	SORTED!	YES	IΤ	IS	SORTED!
YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESYESITISSORTED!YESITISSORTED!YESITISSORTED!YESYESITISSORTED!YESYESITISSORTED!YESITISSORTED!YESYESITISSORTED!YESYESITISSORTED!YESITISSORTED!YESYESITISSORTED!YESYESITISSORTED!YESITISSORTED!YESYESITISSORTED!YESYESITISSORTED!YESITISSORTED!YESYESITISSORTED!YESYESITISSORTED!YESITISSORTED!YESYESITISSORTED!YESYESITISSORTED!YESITISSORTED!YESITISSORTED!YESYESI	YES	IT	IS	SORTED!	YES	IΤ	IS	SORTED!	YES	IΤ	IS	SORTED!
YES IT IS SORTED!YES	YES	IT	IS	SORTED!	YES	IT	IS	SORTED!	YES	IT	IS	SORTED!
YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESYESITISSORTED!YESYESITISSORTED!YESITISSORTED!YESYESITISSORTED!YESYESSORTED!YESITISSORTED!YESYESITISSORTED!YESYESSORTED!YESITISSORTED!YESYESITISSORTED!YESYESSORTED!YESITISSORTED!YESYESITISSORTED!YESYESSORTED!YESITISSORTED!YESYESITISSORTED!YESYESSORTED!YESITISSORTED!YESYESITISSORTED!YESYESSORTED!YESITISSORTED!YESYESITISSORTED!YESYESSORTED!YESITISSORTED!YESYESITISSORTED!YESYESSORTED!YESITISSORTED!YESYESIT<	YES	IT	IS	SORTED!	YES	IΤ	IS	SORTED!	YES	IΤ	IS	SORTED!
YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESYESITISSORTED!YESYESITISSORTED!YESITISSORTED!YESYESITISSORTED!YESYESITISSORTED!YESITISSORTED!YESYESITISSORTED!YESYESSORTED!YESITISSORTED!YESYESITISSORTED!YESYESITISSORTED!YESITISSORTED!YESYESITISSORTED!YESYESITISSORTED!YESITISSORTED!YESYESITISSORTED!YESYESITISSORTED!YESITISSORTED!YESYESITISSORTED!YESYESITISSORTED!YESITISSORTED!YESYESITISSORTED!YESYESITISSORTED!YESITISSORTED!YESITISSORTED!YES <td< td=""><td>YES</td><td>IT</td><td>IS</td><td>SORTED!</td><td>YES</td><td>IΤ</td><td>IS</td><td>SORTED!</td><td>YES</td><td>IΤ</td><td>IS</td><td>SORTED!</td></td<>	YES	IT	IS	SORTED!	YES	IΤ	IS	SORTED!	YES	IΤ	IS	SORTED!
YES IT IS SORTED!YES IT IS SORTED!	YES	IT	IS	SORTED!	YES	IT	IS	SORTED!	YES	IT	IS	SORTED!
YES IT IS SORTED!YES IT IS SORTED!	YES	IT	IS	SORTED!	YES	IΤ	IS	SORTED!	YES	IΤ	IS	SORTED!
YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESITISSORTED!YESYESITISSORTED!YESITISSORTED!YESITISSORTED!YESYESITISSORTED!YESITISSORTED!YESITISSORTED!YESYESITISSORTED!YESYESITISSORTED!YESITISSORTED!YESITISSORTED!YESYESITISSORTED!YESITISSORTED!YESITISSORTED!YESYESITISSORTED!YESITISSORTED!YESITISSORTED!YESYESITISSORTED!YESITISSORTED!YESITISSORTED!YESYESITISSORTED!YESITISSORTED!YESITISSORTED!YESYESITISSORTED!YESIT	YES	IΤ			YES			SORTED!	YES			
YES IT IS SORTED!YES IT IS SORTED!	YES	IT	IS	SORTED!	YES	IT	IS	SORTED!	YES	IT	IS	SORTED!
YES IT IS SORTED!YES IT IS SORTED!	YES	IT	IS	SORTED!	YES	IT	IS	SORTED!	YES	IΤ	IS	SORTED!
YES IT IS SORTED!YES IT IS SORTED!	YES	IΤ	IS	SORTED!	YES	IΤ	IS	SORTED!	YES	IΤ	IS	SORTED!
YES IT IS SORTED!YES IT IS SORTED!	YES	IT	IS	SORTED!	YES	IT	IS	SORTED!	YES	IT	IS	SORTED!
YES IT IS SORTED!YES IT IS SORTED!	YES	IT	IS	SORTED!	YES	IΤ	IS	SORTED!	YES	IΤ	IS	SORTED!
YES IT IS SORTED!YES IT IS SORTED!	YES				YES			SORTED!	YES			SORTED!
YES IT IS SORTED! YES IT IS SORTED! YES IT IS SORTED! YES IT IS SORTED! YES IT IS SORTED! YES IT IS SORTED! YES IT IS SORTED! YES IT IS SORTED! YES IT IS SORTED! YES IT IS SORTED! YES IT IS SORTED! YES IT IS SORTED!	YES	IT	IS	SORTED!	YES	IT	IS	SORTED!	YES	IT	IS	SORTED!
YES IT IS SORTED! YES IT IS SORTED! YES IT IS SORTED! YES IT IS SORTED! YES IT IS SORTED! YES IT IS SORTED!	YES	IT	IS	SORTED!	YES	IΤ	IS	SORTED!	YES	IΤ	IS	SORTED!
YES IT IS SORTED! YES IT IS SORTED! YES IT IS SORTED!	YES	IΤ	IS	SORTED!	YES	IΤ	IS	SORTED!	YES	IΤ	IS	SORTED!
	YES	IT	IS	SORTED!	YES	IT	IS	SORTED!	YES	IΤ	IS	SORTED!
YES IT IS SORTED! YES IT IS SORTED! YES IT IS SORTED!	YES	IT	IS	SORTED!	YES	IΤ	IS	SORTED!	YES	IΤ	IS	SORTED!
	YES	IΤ	IS	SORTED!	YES	IΤ	IS	SORTED!	YES	IΤ	IS	SORTED!

What does it mean to "work"?

- Is it enough to be correct on only one input?
- Is it enough to be correct on most inputs?
- In this class, we will use **worst-case analysis**:
 - An algorithm must be correct on **all possible** inputs.
 - The running time of an algorithm is the worst possible running time over all inputs.

Worst-case analysis

Think of it like a game:



Here is my algorithm!

Algorithm: Do the thing Do the stuff Return the answer

Algorithm designer

- Pros: very strong guarantee
- Cons: very strong guarantee



Insertion Sort

1. Does it work?



2. Is it fast?



• Okay, so it's pretty obvious that it works.



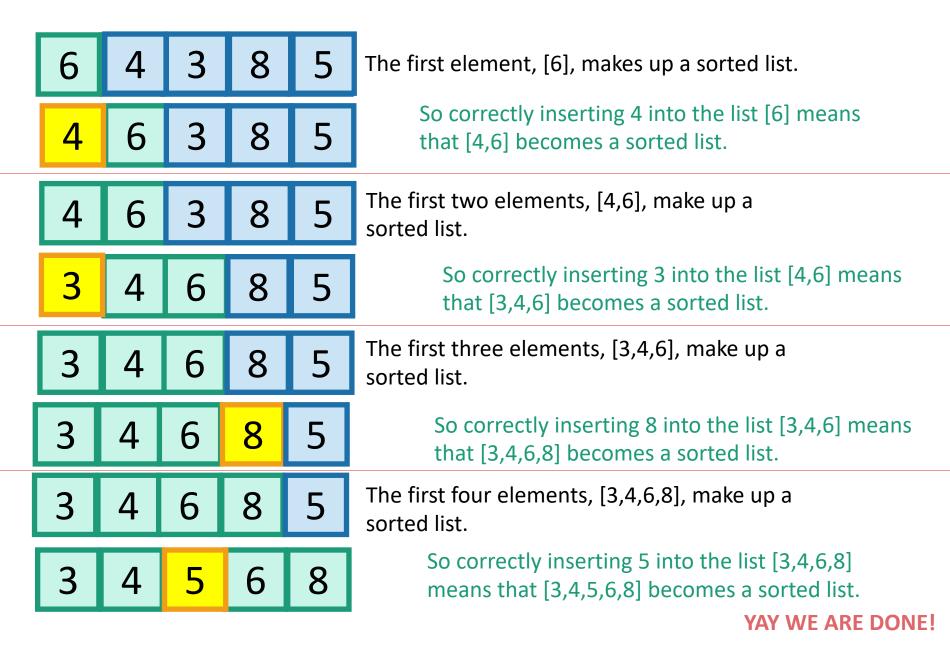
 HOWEVER! In the future it won't be so obvious, so let's take some time now to see how we would prove this rigorously.

Why does this work?

Say you have a sorted list, 3 4 6 8, and another element 5.

• Then you get a sorted list: 3 4 5 6 8

So just use this logic at every step.



This sounds like a job for...

Proof By Induction!

There is a handout with details!

See website!

2 Correctness of InsertionSort

Once you figure out what INSERTIONSORT is doing (see the slides/lecture video for the intuition on this), you may think that it's "obviously" correct. However, if you didn't know what it was doing and just got the above code, maybe this wouldn't be so obvious. Additionally, for algorithms that we'll study in the future, it *won't* always be obvious that it works, and so we'll have to prove it. So in this handout we'll carefully go through a proof that INSERTIONSORT is correct.

We'll do the proof by maintaining a *loop invariant*, in this case that after iteration *i*, then A[:i+1] is sorted. This is obviously true after iteration 0 (aka, before the algorithm begins), because the one-element list A[:1] is definitely sorted. Then we'll show that for any k with 0 < k < n, if this loop invariant holds for k-1, then it holds for k. That is, if it is true that A[:k] is sorted after the k-1'st iteration, then it is true that A[:k+1] is sorted after the k'th iteration. At the end of the day, we'll conclude that A[:n] (aka, the whole thing) is sorted after the n-1'st iteration, and we'll be done.

Formally, we will proceed by induction.

- Inductive hypothesis. After iteration i of the outer loop, A[:i+1] is sorted.
- Base case. After iteration 0 of the outer loop (aka, before the algorithm begins), the list A[:1] contains only one element, and this is sorted.
- Inductive step. Let k be an integer so that 0 < k < n. Suppose that the inductive hypothesis holds for k − 1, so A[:k] is sorted after the k − 1'st iteration. We want to show that A[:k+1] is sorted after the k'th iteration.

Suppose that j^* is the largest integer in $\{0, ..., k-1\}$ such that $A[j^*] < A[k]$. Then the effect of the inner loop is to turn

$$[A[0], A[1], \ldots, A[j^*], \ldots, A[k-1], A[k]]$$

into

$$[A[0], A[1], \dots, A[j^*], A[k], A[j^*+1], \dots, A[k-1]].$$

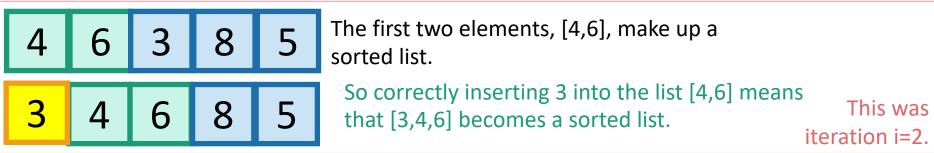
Outline of a proof by induction

Let A be a list of length n

- Inductive Hypothesis:
 - A[:i+1] is sorted at the end of the ith iteration (of the outer loop).
- Base case (i=0):
 - A[:1] is sorted at the end of the 0'th iteration. \checkmark
- Inductive step:
 - For any 0 < k < n, if the inductive hypothesis holds for i=k-1, then it holds for i=k.

(see handout for details)

- Aka, if A[:k] is sorted at step k-1, then A[:k+1] is sorted at step k
- Conclusion:
 - The inductive hypothesis holds for i = 0, 1, ..., n-1.
 - In particular, it holds for i=n-1.
 - At the end of the n-1'st iteration (aka, at the end of the algorithm),
 A[:n] = A is sorted.
 - That's what we wanted! \checkmark



Aside: proofs by induction

- We're gonna see/do/skip over a lot of them.
- I'm assuming you're comfortable with them from CS103.
 - When you assume...
- If that went by too fast and was confusing:
 - GO TO SECTION
 - GO TO SECTION
 - Handout
 - References
 - Office Hours

Make sure you really understand the argument on the previous slide! Check out the handout for a more formal writeup, and go to section for an overview of what we are looking for in proofs by induction.



Siggi the Studious Stork

What have we learned?

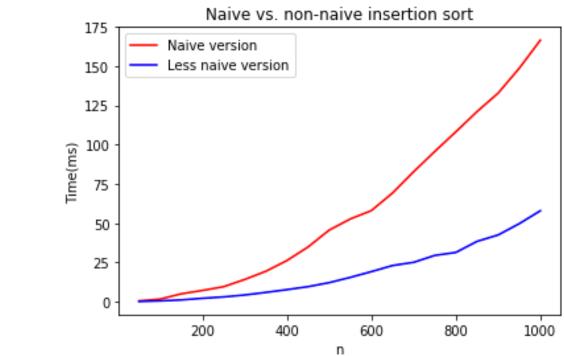
- In this class we will use worst-case analysis:
 - We assume that a "bad guy" comes up with a worst-case input for our algorithm, and we measure performance on that worst-case input.
- With this definition, InsertionSort "works"
 - Proof by induction!

The Plan

- InsertionSort recap
- Worst-case Analysis
 - Back to InsertionSort: Does it work?
- Asymptotic Analysis
 - Back to InsertionSort: Is it fast?
- MergeSort
 - Does it work?
 - Is it fast?

How fast is InsertionSort?

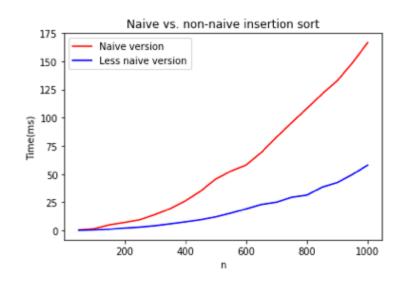
• This fast:





Issues with this answer?

- The "same" algorithm can be slower or faster depending on the implementations.
- It can also be slower or faster depending on the hardware that we run it on.



With this answer, "running time" isn't even well-defined!



How fast is InsertionSort?

Let's count the number of operations!



```
def InsertionSort(A):
    for i in range(1,len(A)):
        current = A[i]
        j = i-1
        while j >= 0 and A[j] > current:
            A[j+1] = A[j]
            j -= 1
            A[j+1] = current
```

By my count*...

- $2n^2 n 1$ variable assignments
- $2n^2 n 1$ increments/decrements
- $2n^2 4n + 1$ comparisons
- ...

*Do not pay attention to these formulas, they do not matter. Also not valid for bug bounty points.

Issues with this answer?

- It's very tedious!
- In order to use this to understand running time, I need to know how long each operation takes, plus a whole bunch of other stuff...

```
def InsertionSort(A):
    for i in range(1,len(A)):
        current = A[i]
        j = i-1
        while j >= 0 and A[j] > current:
            A[j+1] = A[j]
            j -= 1
            A[j+1] = current
```

Counting individual operations is a lot of work and doesn't seem very helpful!



Lucky the lackadaisical lemur

In this class we will use ...

• Big-Oh notation!

 Gives us a meaningful way to talk about the running time of an algorithm, independent of programming language, computing platform, etc., without having to count all the operations.

Main idea:

Focus on how the runtime scales with n (the input size).

Some examples...

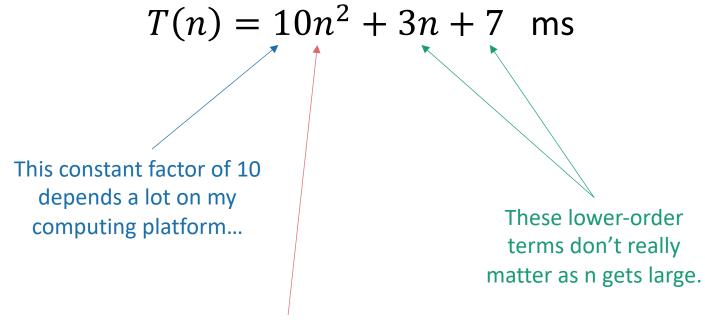
(Only pay attention to the largest function of n that appears.)

Number of operations	Asymptotic Running Time	
$\frac{1}{10} \cdot n^2 + 100$	$O(n^2)$	
$0.063 \cdot n^25 n + 12.7$	$O(n^2)$	We sa "asy
$100 \cdot n^{1.5} - 10^{10000} \sqrt{n}$	$O(n^{1.5})$	
$11 \cdot n \log(n) - 1$	$O(n\log(n))$	

We say this algorithm is "asymptotically faster" than the others.

Why is this a good idea?

• Suppose the running time of an algorithm is:



We're just left with the n² term! That's what's meaningful.

Pros and Cons of Asymptotic Analysis

Pros:

- Abstracts away from hardware- and languagespecific issues.
- Makes algorithm analysis much more tractable.
- Allows us to meaningfully compare how algorithms will perform on large inputs.

Cons:

• Only makes sense if n is large (compared to the constant factors).

100000000 n is "better" than n² ?!?!

pronounced "big-oh of ..." or sometimes "oh of ..."

Informal definition for O(...)



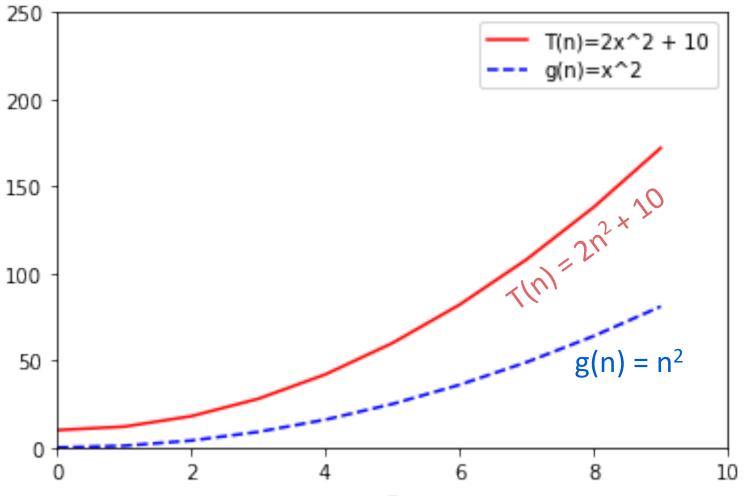
- Let T(n), g(n) be functions of positive integers.
 Think of T(n) as a runtime: positive and increasing in n.
- We say "T(n) is O(g(n))" if:

for large enough n,

T(n) is at most some constant multiple of g(n).

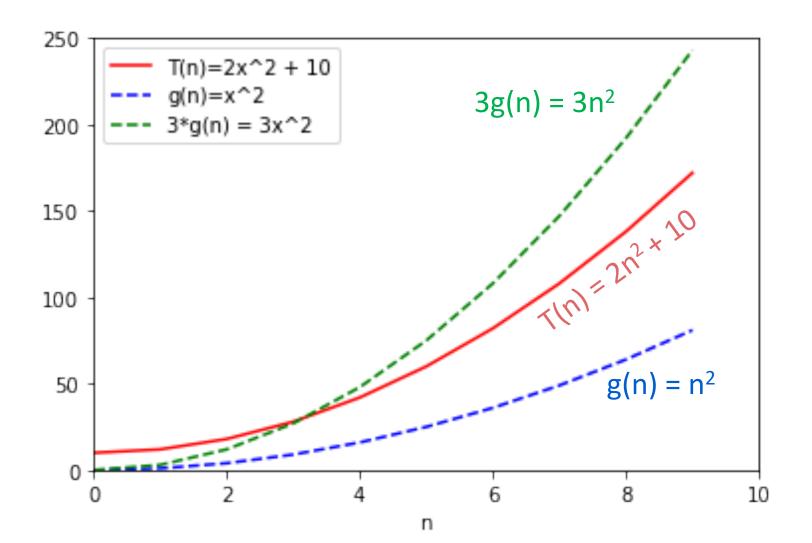
Here, "constant" means "some number that doesn't depend on n."

for large enough n, T(n) is at most some constant multiple of g(n).

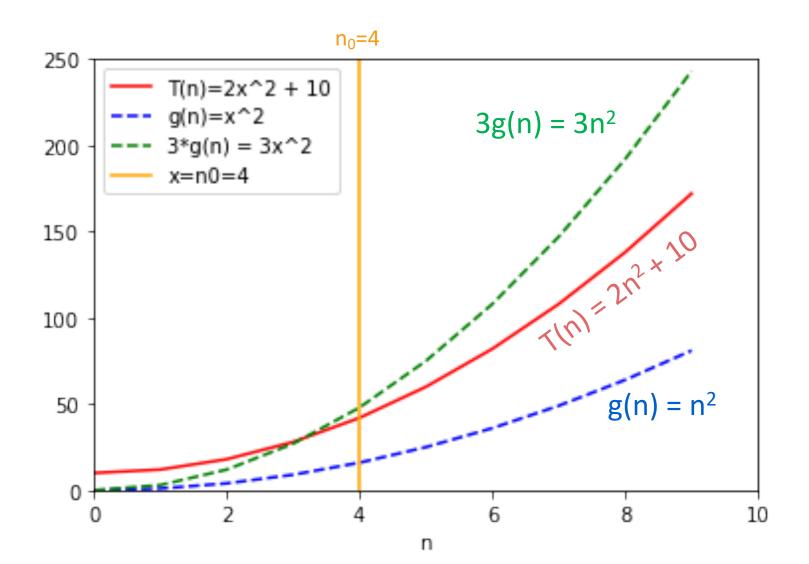


n

for large enough n, T(n) is at most some constant multiple of g(n).

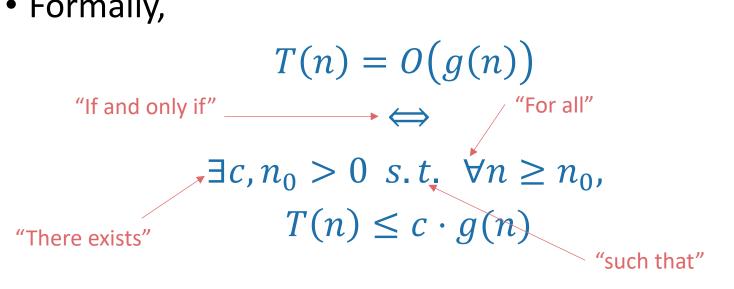


for large enough n, T(n) is at most some constant multiple of g(n).

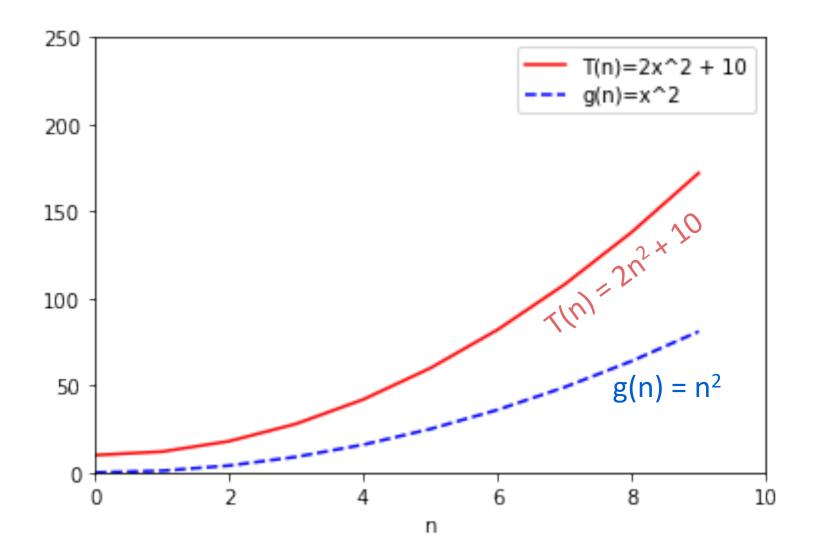


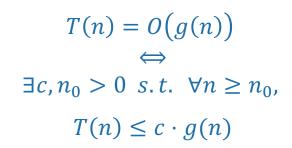
Formal definition of O(...)

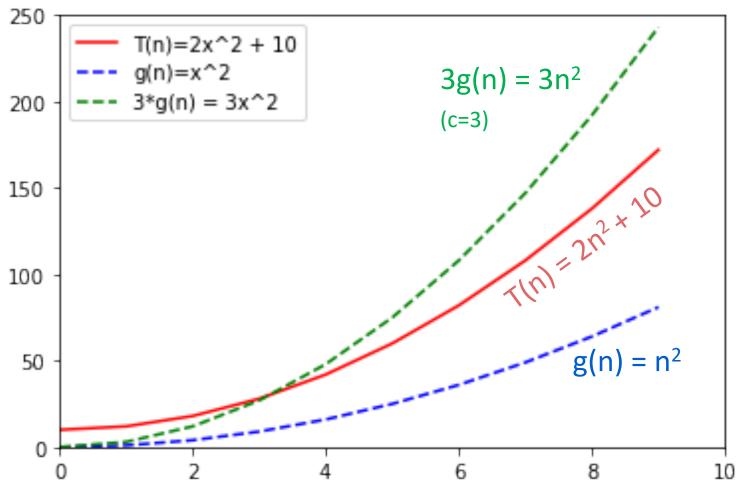
- Let T(n), g(n) be functions of positive integers. • Think of T(n) as a runtime: positive and increasing in n.
- Formally,



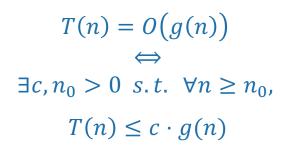
T(n) = O(g(n)) \Leftrightarrow $\exists c, n_0 > 0 \ s.t. \ \forall n \ge n_0,$ $T(n) \le c \cdot g(n)$

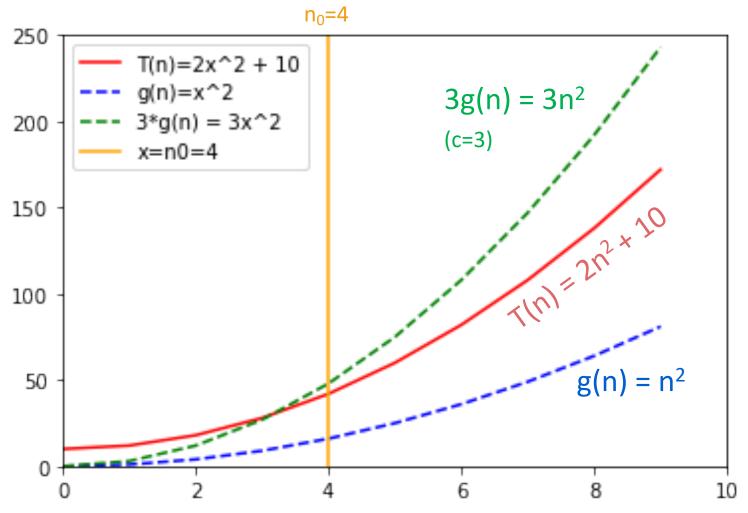






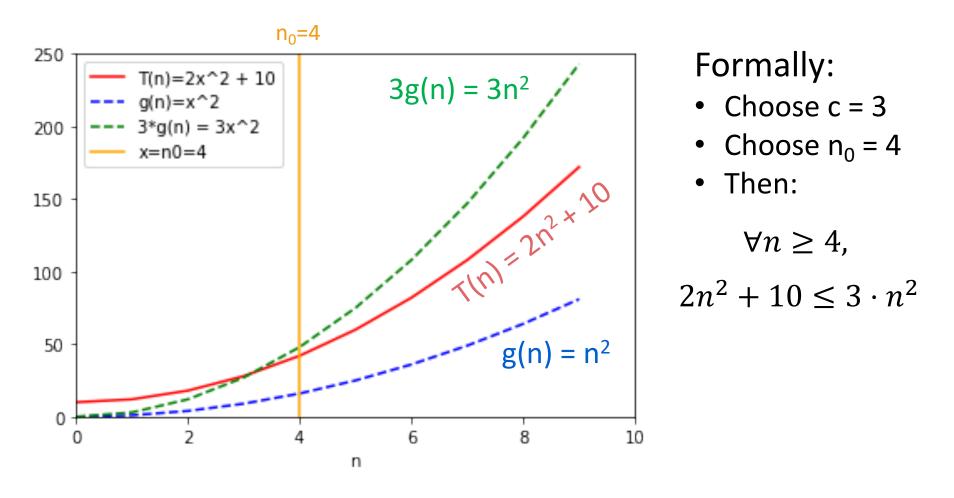
n





n

T(n) = O(g(n)) \Leftrightarrow $\exists c, n_0 > 0 \ s. t. \ \forall n \ge n_0,$ $T(n) \le c \cdot g(n)$



Same example $2n^2 + 10 = O(n^2)$

T(n) = O(g(n)) \Leftrightarrow $\exists c, n_0 > 0 \ s. t. \ \forall n \ge n_0,$ $T(n) \le c \cdot g(n)$

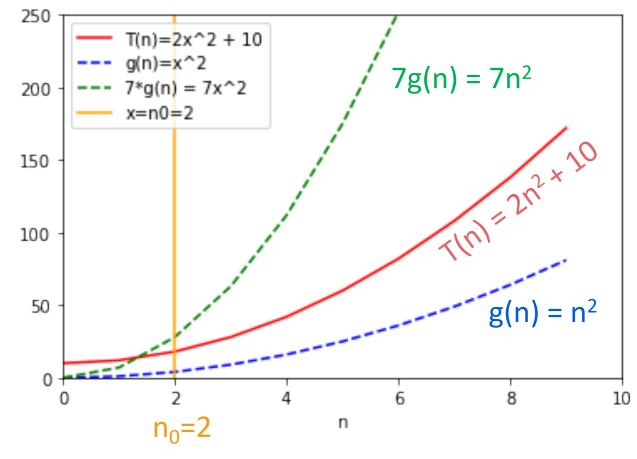
Formally:

- Choose c = 7
- Choose $n_0 = 2$

• Then:

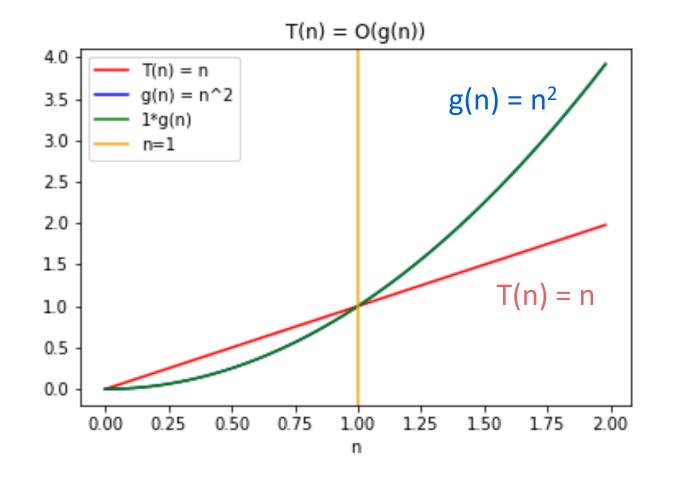
 $\forall n \ge 2,$ $2n^2 + 10 \le 7 \cdot n^2$





O(...) is an upper bound: $n = O(n^2)$

T(n) = O(g(n)) \Leftrightarrow $\exists c, n_0 > 0 \ s.t. \ \forall n \ge n_0,$ $T(n) \le c \cdot g(n)$



• Choose $n_0 = 1$

• Then

 $\forall n \geq 1$,

 $n \leq n^2$

Break

$\Omega(...)$ means a lower bound

- We say "T(n) is $\Omega(g(n))$ " if, for large enough n, T(n) is at least as big as a constant multiple of g(n).
- Formally,

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

$$\Leftrightarrow$$

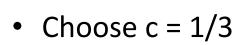
$$\exists c, n_0 > 0 \quad s. t. \quad \forall n \ge n_0,$$

$$c \cdot g(n) \le T(n)$$

$$\swarrow$$
Switched these!!

Example $n \log_2(n) = \Omega(3n)$

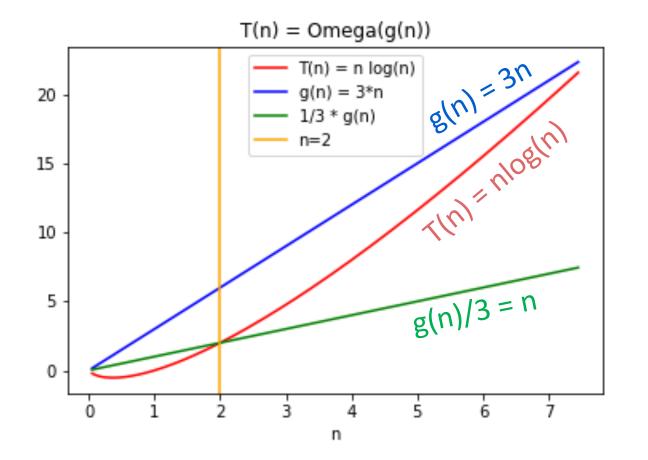
 $T(n) = \Omega(g(n))$ \Leftrightarrow $\exists c, n_0 > 0 \ s.t. \ \forall n \ge n_0,$ $c \cdot g(n) \le T(n)$



- Choose $n_0 = 2$
- Then

 $\forall n \geq 2$,

 $\frac{3n}{3} \le n \log_2(n)$



$\Theta(...)$ means both!

• We say "T(n) is $\Theta(g(n))$ " iff both:

$T(n) = O\bigl(g(n)\bigr)$

and

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

Non-Example: n^2 is not O(n)

- Proof by contradiction:
- Suppose that $n^2 = O(n)$.
- Then there is some positive c and n₀ so that:

$$\forall n \ge n_0, \qquad n^2 \le c \cdot n$$

• Divide both sides by n:

$$\forall n \ge n_0, \qquad n \le c$$

- That's not true!!! What about, say, $n_0 + c + 1$?
 - Then $n \ge n_0$, but , n > c
- Contradiction!

$$T(n) = O(g(n))$$

$$\Leftrightarrow$$

$$\exists c, n_0 > 0 \ s.t. \ \forall n \ge n_0,$$

$$T(n) \le c \cdot g(n)$$

Take-away from examples

- To prove T(n) = O(g(n)), you have to come up with c and n₀ so that the definition is satisfied.
- To prove T(n) is NOT O(g(n)), one way is proof by contradiction:
 - Suppose (to get a contradiction) that someone gives you a c and an n_0 so that the definition *is* satisfied.
 - Show that this someone must by lying to you by deriving a contradiction.

Another example: polynomials

- Say $p(n) = a_k n^k + a_{k-1} n^{k-1} + \dots + a_1 n + a_0$ is a polynomial of degree $k \ge 1$.
- Then:
 - 1. $p(n) = O(n^k)$ 2. p(n) is **not** $O(n^{k-1})$
- See the notes/references for a proof.

Try to prove it yourself first!



Siggi the Studious Stork

More examples

- $n^3 + 3n = O(n^3 n^2)$
- $n^3 + 3n = \Omega(n^3 n^2)$
- $n^3 + 3n = \Theta(n^3 n^2)$
- 3ⁿ is NOT O(2ⁿ)
- $\log_2(n) = \Omega(\ln(n))$
- $\log_2(n) = \Theta(2^{\log\log(n)})$

Work through these on your own! Also look at the examples in the reading!



Siggi the Studious Stork

Some brainteasers

- Are there functions f, g so that NEITHER f = O(g) nor f = Ω(g)?
- Are there non-decreasing functions f, g so that the above is true?



Ollie the Over-achieving Ostrich

Recap: Asymptotic Notation

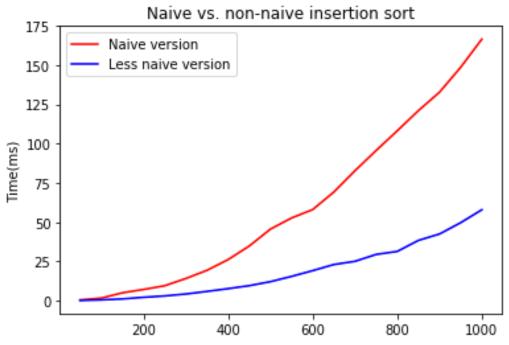
- This makes both Plucky and Lucky happy.
 - Plucky the Pedantic Penguin is happy because there is a precise definition.
 - Lucky the Lackadaisical Lemur is happy because we don't have to pay close attention to all those pesky constant factors.
- But we should always be careful not to abuse it.
- In the course, (almost) every algorithm we see will be actually practical, without needing to take $n \ge n_0 = 2^{10000000}$.



This is my happy face!

Back Insertion Sort

- 1. Does it work?
- 2. Is it fast?



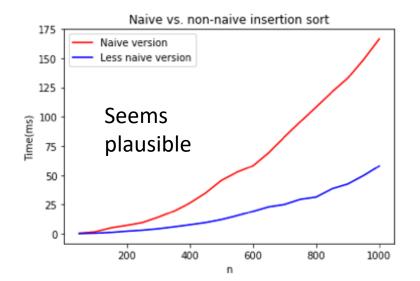
Insertion Sort: running time

Operation count was:

- $2n^2 n 1$ variable assignments
- $2n^2 n 1$ increments/decrements
- $2n^2 4n + 1$ comparisons
- ...
- The running time is $O(n^2)$

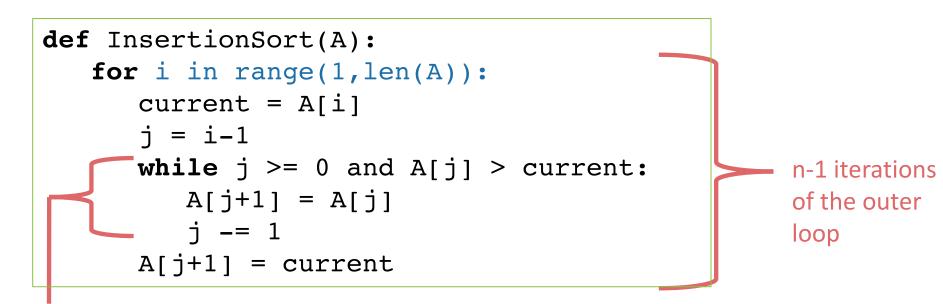


Go back to the pseudocode and convince yourself of this!



Insertion Sort: running time

As you get more used to this, you won't have to count up operations anymore. For example, just looking at the pseudocode below, you might think...



In the worst case, about n iterations of this inner loop

"There's O(1) stuff going on inside the inner loop, so each time the inner loop runs, that's O(n) work. Then the inner loop is executed O(n) times by the outer loop, so that's O(n^2)."



What have we learned?

InsertionSort is an algorithm that correctly sorts an arbitrary n-element array in time $O(n^2)$.

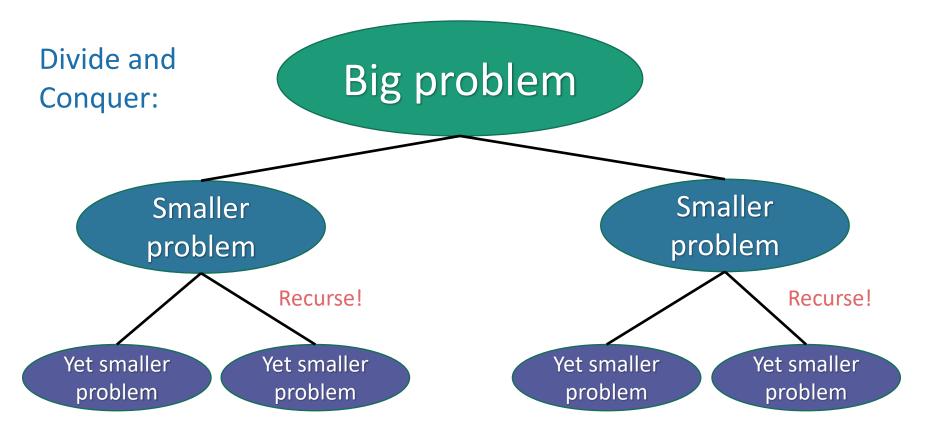
Can we do better?

The Plan

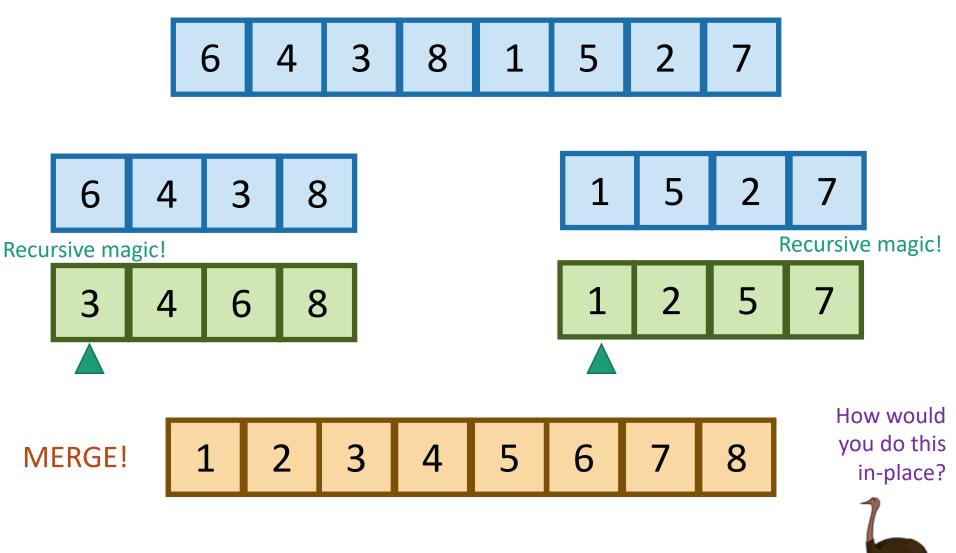
- InsertionSort recap
- Worst-case analyisis
 - Back to InsertionSort: Does it work?
- Asymptotic Analysis
 - Back to InsertionSort: Is it fast?
- MergeSort
 - Does it work?
 - Is it fast?

Can we do better?

- MergeSort: a divide-and-conquer approach
- Recall from last time:



MergeSort



Code for the MERGE step is given in the Lecture2 IPython notebook, or the textbook

Ollie the over-achieving Ostrich

MergeSort Pseudocode

MERGESORT(A):

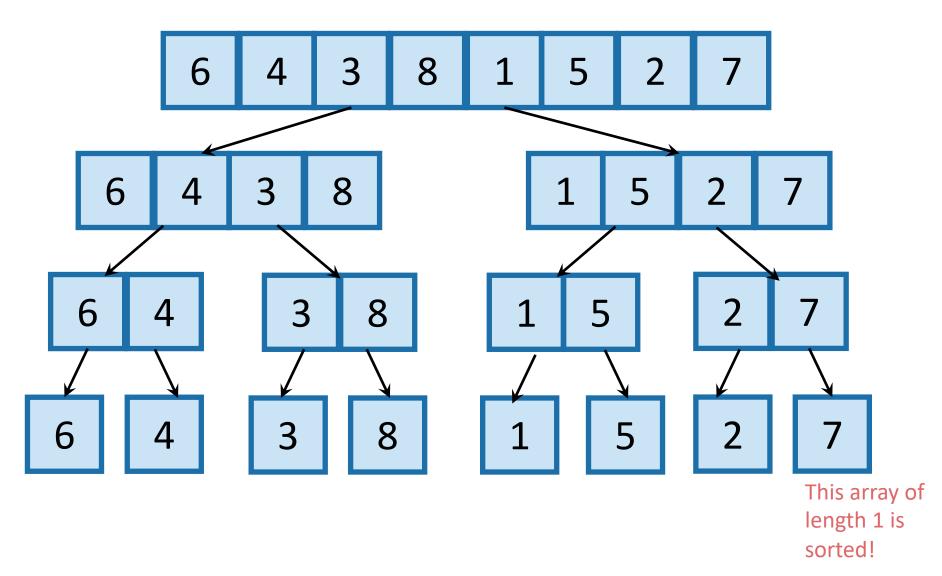
- n = length(A)
- if n ≤ 1: If A has length 1,
 return A
- L = MERGESORT(A[0 : n/2])
- R = MERGESORT(A[n/2 : n])

Sort the left half

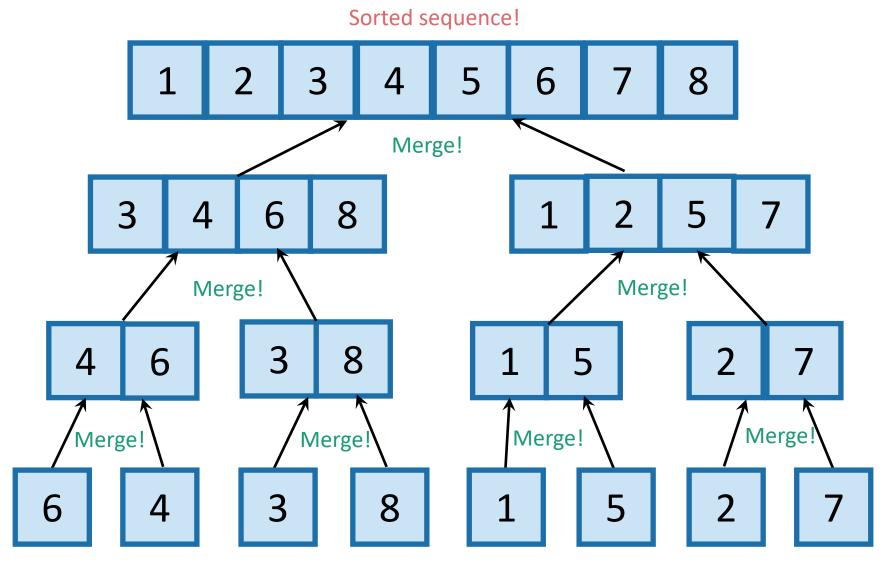
- Sort the right half
- return MERGE(L,R) Merge the two halves

What actually happens?

First, recursively break up the array all the way down to the base cases



Then, merge them all back up!

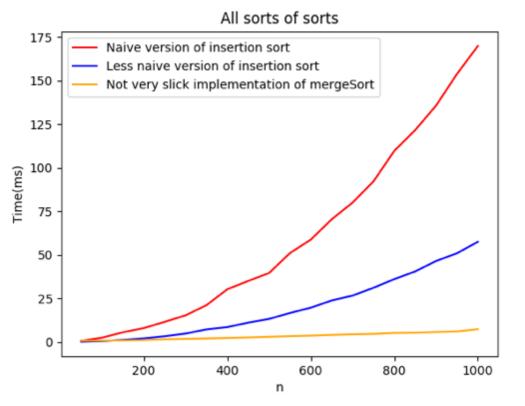


A bunch of sorted lists of length 1 (in the order of the original sequence).

Two questions

- 1. Does this work?
- 2. Is it fast?

IPython notebook says...



Empirically:

- 1. Seems to work.
- 2. Seems fast.

It works

• Yet another job for...

Proof By Induction!

Work this out! There's a skipped slide with an outline to help you get started.



lt works

Assume that n is a power of 2 for convenience.

Inductive hypothesis:

"In every the recursive call on an array of length at most i, MERGESORT returns a sorted array."

- Base case (i=1): a 1-element array is always sorted.
- Inductive step: Need to show: if the inductive hypothesis holds for k<i, then it holds for k=i.
- Aka, need to show that if L and R are sorted, then MERGE(L,R) is sorted.
- Conclusion: In the top recursive call, MERGESORT returns a sorted array.

- MERGESORT(A):
 - n = length(A)
 - **if** $n \le 1$:
 - return A
 - L = MERGESORT(A[1 : n/2])
 - R = MERGESORT(A[n/2+1 : n])
 - return MERGE(L,R)

Fill in the inductive step! HINT: You will need to prove that the MERGE algorithm is correct, for which you may need...another proof by induction!

Assume that n is a power of 2 for convenience.

CLAIM:

It's fast

MergeSort runs in time $O(n \log(n))$

- Proof coming soon.
- But first, how does this compare to InsertionSort?
 - Recall InsertionSort ran in time $O(n^2)$.

$O(n \log(n))$ vs. $O(n^2)$?

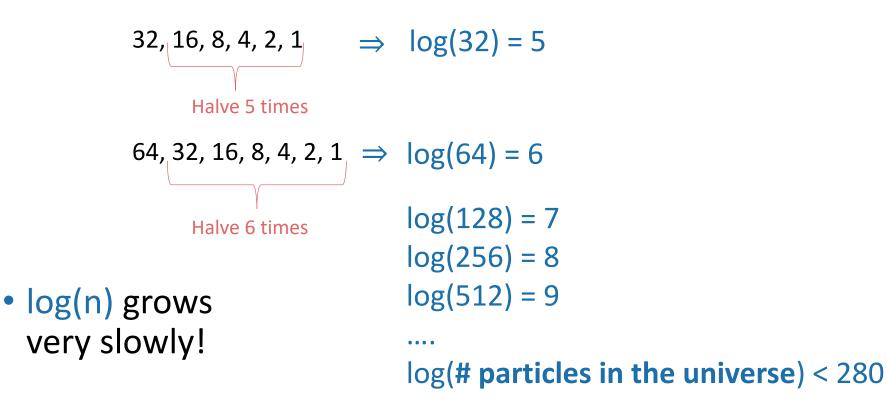
All logarithms in this course are base 2

Aside:

Quick log refresher



- Def: log(n) is the number so that $2^{\log(n)} = n$.
- Intuition: log(n) is how many times you need to divide n by 2 in order to get down to 1.



$O(n \log n)$ vs. $O(n^2)$?

- log(n) grows much more slowly than n
- $n \log(n)$ grows much more slowly than n^2

Punchline: A running time of O(n log n) is a lot better than O(n²)!

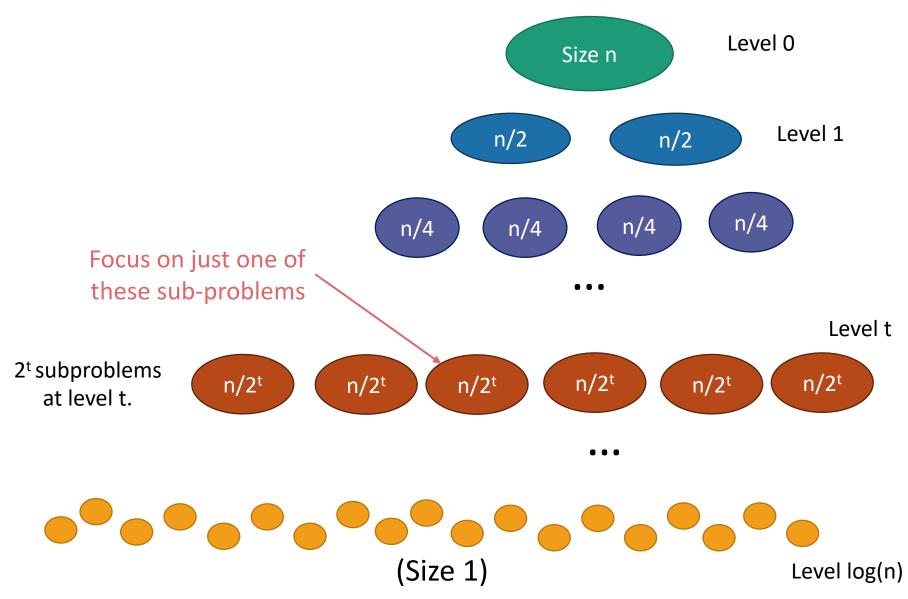
Assume that n is a power of 2 Now let's prove the claim

CLAIM:

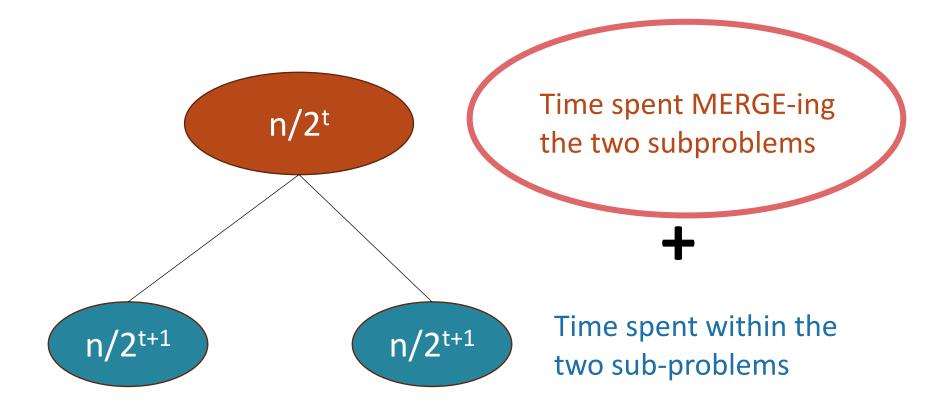
MergeSort runs in time $O(n \log(n))$

for convenience.

Let's prove the claim

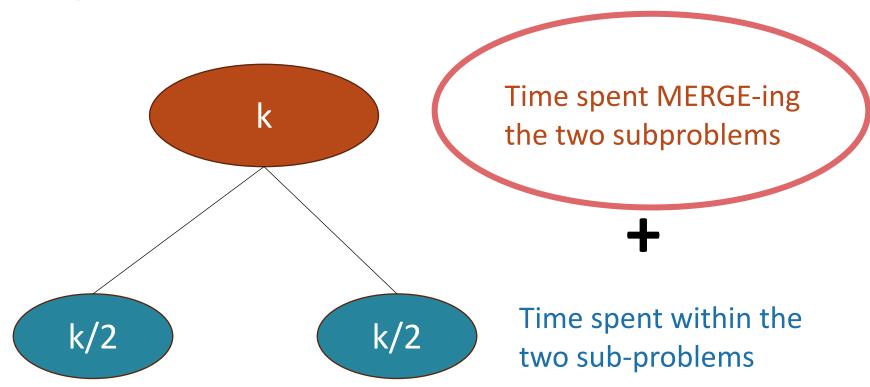


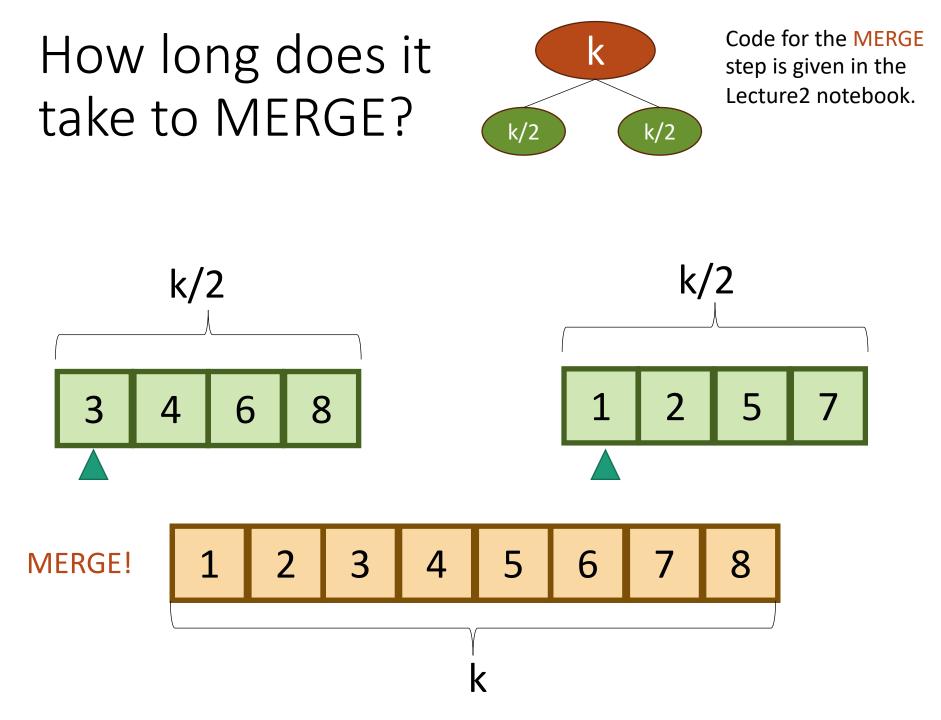
How much work in this sub-problem?



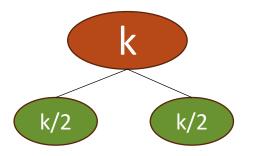
How much work in this sub-problem?

Let $k=n/2^t$...



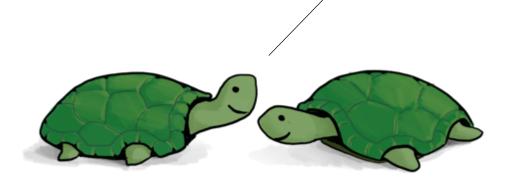


How long does it take to MERGE?



Code for the MERGE step is given in the Lecture2 notebook.

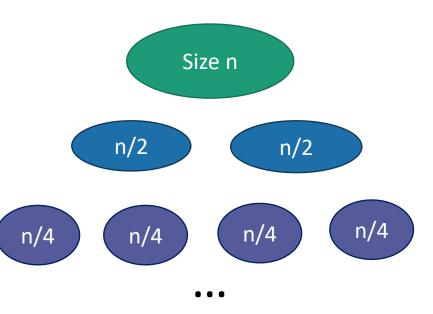
How long does it take to run MERGE on two lists of size k/2?

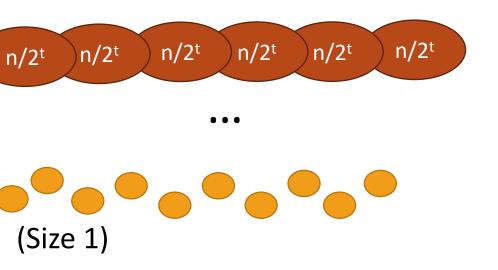


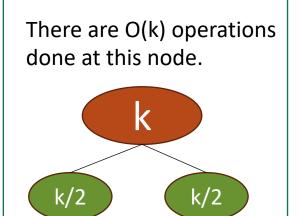
Think-Pair-Share Terrapins

Answer: It takes time O(k), since we just walk across the list once.

Recursion tree



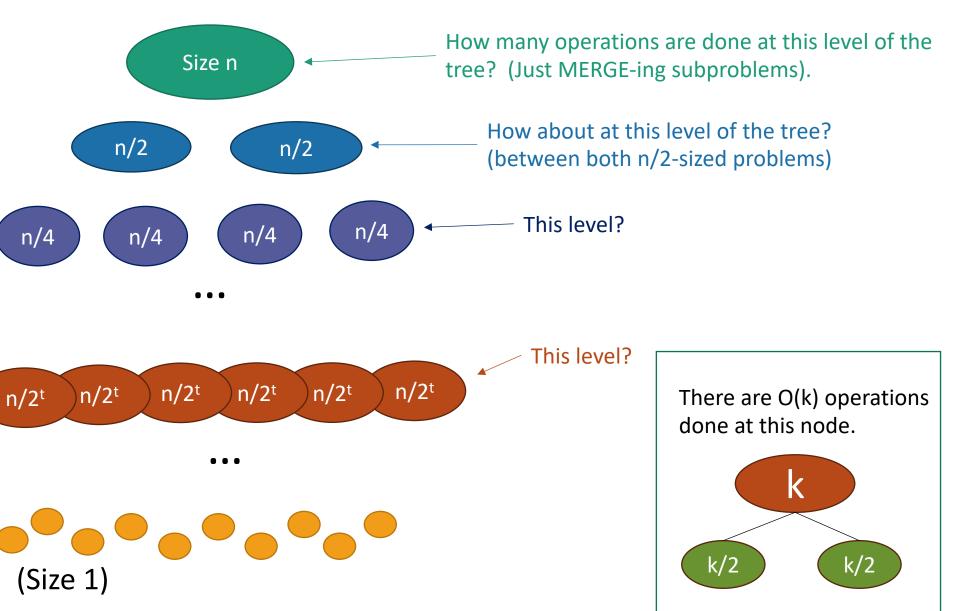


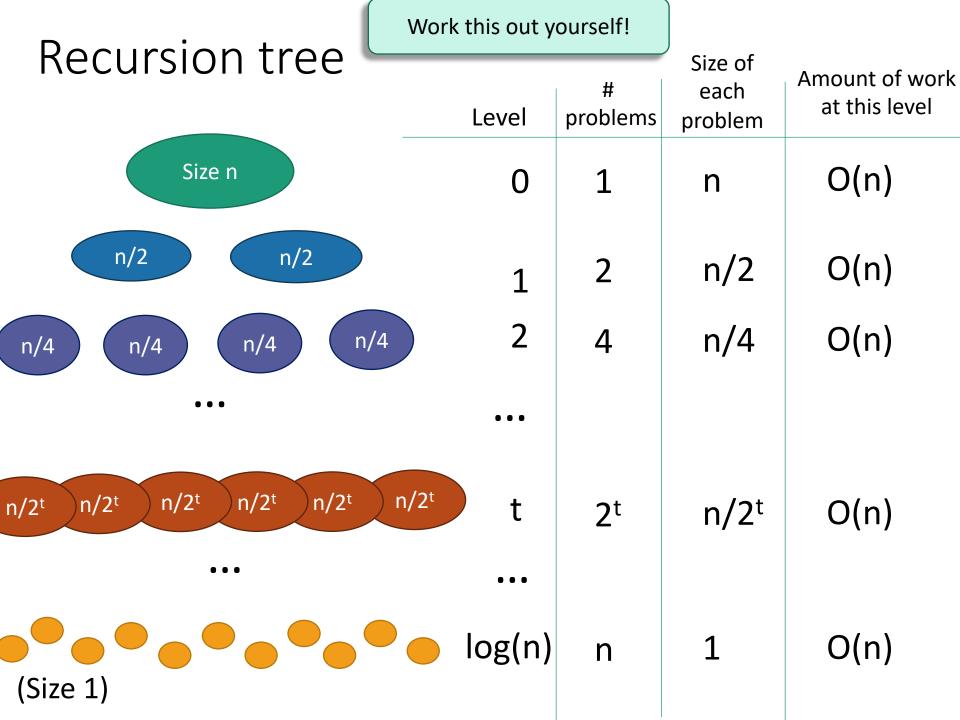


Recursion tree



Think, Pair, Share!





Total runtime...

- O(n) steps per level, at every level
- log(n) + 1 levels
- O(n log(n)) total!

That was the claim!

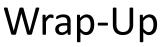
What have we learned?

- MergeSort correctly sorts a list of n integers in time O(n log(n)).
- That's (asymptotically) better than InsertionSort!

The Plan

- InsertionSort recap
- Worst-case analyisis
 - Back to InsertionSort: Does it work?
- Asymptotic Analysis
 - Back to InsertionSort: Is it fast?
- MergeSort
 - Does it work?
 - Is it fast?





Recap

- InsertionSort runs in time O(n²)
- MergeSort is a divide-and-conquer algorithm that runs in time O(n log(n))
- How do we show an algorithm is correct?
 - Today, we did it by induction
- How do we measure the runtime of an algorithm?
 - Worst-case analysis
 - Asymptotic analysis
- How do we analyze the running time of a recursive algorithm?
 - One way is to draw a recursion tree.

Next time

• A more systematic approach to analyzing the runtime of recursive algorithms.

Before next time

- Pre-Lecture Exercise:
 - A few recurrence relations (see website)