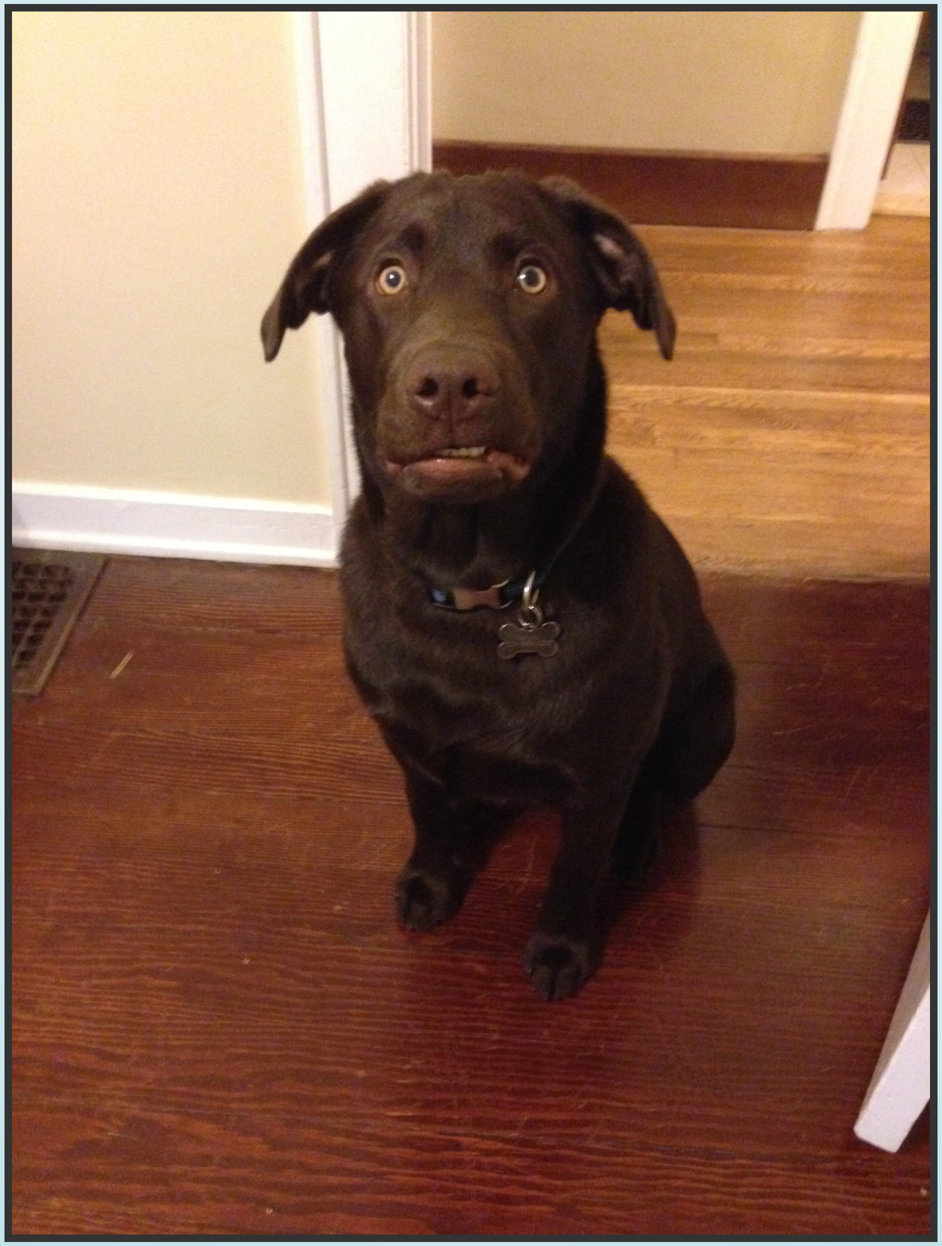
SORTING ALGORITHMS



WHAT IS ABOUT TO HAPPEN?

- Introduction
- Three types of sort:
  - Bubble Sort
  - Merge Sort
  - Quick Sort(quickly)
- Other considerations
- Resources and exercises

WHY ARE SORTING ALGORITHMS A THING?

"Uhh, well, just take the big numbers and put them at the end until they're in order?

No.

Ok, actually that's kind of insertion sort

In any case, computers need incredibly specific instructions.

ANY IDEAS FOR A SORT STRATEGY?

A SIMPLE SORT

FOR A RANDOM LIST

```
[6, 5, 3, 1, 8, 7, 2, 4]
```

How might we go about sorting this programmatically?

A SIMPLE SORT

```
[6, 5, 3, 1, 8, 7, 2, 4]
```

Okay, start at the beginning....that's a 6

Look at the next number... that's a 5

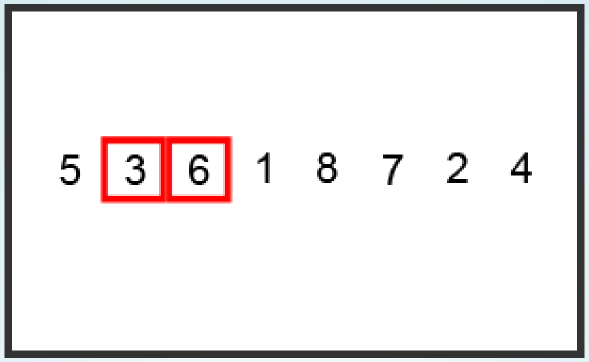
Oh! 5 is less than 6, so switch them!

```
[5, 6, 3, 1, 8, 7, 2, 4]
```

Now compare the next two, 6 to three, switch them!

etc...

A SIMPLE SORT



Source

A SIMPLE SORT

Let's code it!

```
def bubble_sort(l):

    for i in range(len(l) - 1):
      num1, num2 = l[i], l[i+1]

      if num1 > num2:
        l[i], l[i+1] = num2, num1

    return l
```

What's wrong with this?

A SIMPLE SORT

Let's code it better!

```
def bubble_sort(l):
  is_sorted = False

  while is_sorted == False:
    is_sorted=True

    for i in range(len(l) - 1):     # go through each except last
      num1, num2 = l[i], l[i+1]

      if num1 > num2:               # if you have to switch,
        is_sorted = False           # it's not sorted
        l[i], l[i+1] = num2, num1   # switch the numbers
```

```
l = bubble_sort([6, 5, 3, 1, 8, 7, 2, 4]) # => 42 steps!
```

A SIMPLE SORT

Runtime?

Two nested loops, so ....

$O(n^2)$

Bonus: Can you make it run in 21 steps (Twice as fast!)?

Runtime will be $O(n^2/2)$, so unfortunately still $O(n^2)$

BUBBLE SORT

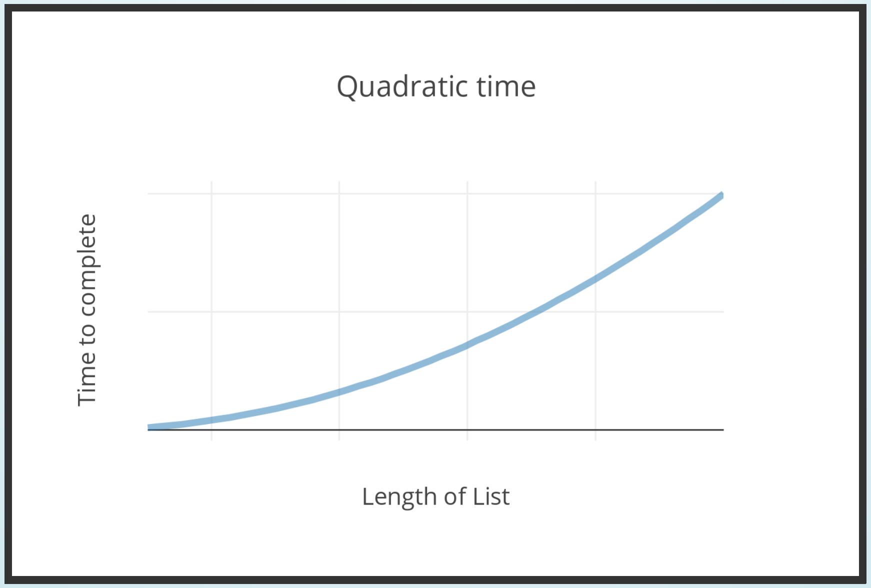
We have just implemented bubble sort!

OTHERS?

- Insertion Sort
- Select Sort

Worst case: $O(n^2)$

Average case: $O(n^2)$



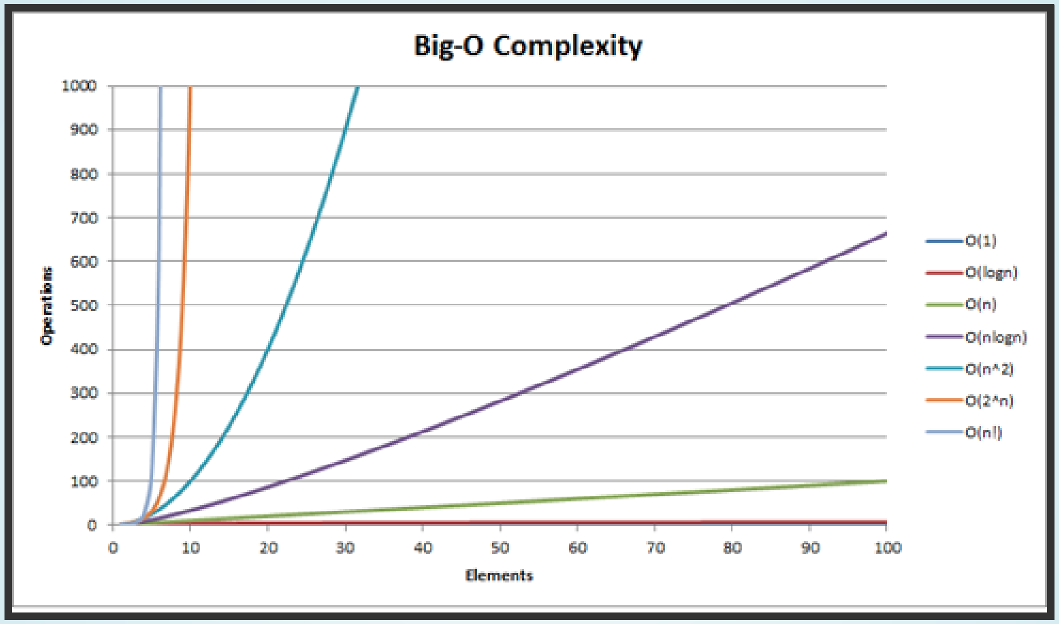
: (

WE CAN DO BETTER!

ENTER MERGE SORT

MERGE SORT

Merge sort is awesome because it runs in O(n log(n)) time



MERGE SORT

BUT FIRST: MERGE

Let's say we have **two already sorted arrays** and we want to **make one sorted array**.

```
lst1 = [1, 2, 4, 7]
lst2 = [3, 5, 6, 8]
```

How can we merge them?

```
#output => [1, 2, 3, 4, 5, 6, 7, 8]
```

MERGE SORT

MERGE: STRATEGY

1. Start with two sorted lists
2. Initialize new, empty list for results
3. Compare first element of each sorted list
4. Remove whichever element is lower and add it to the results list.
5. Continue comparing the first elements of each list until one of them has no more items.
6. Append the remaining items from the other list to results list

Any volunteers? :)

MERGE SORT

MERGE

```
def make_one_sorted_list(lst1, lst2):
    result_list = []
    while len(lst1) > 0 and len(lst2) > 0:  #if items left in both lists
        #compare first items of each list
        if lst1[0] < lst2[0]:
            result_list.append(lst1.pop(0))  #append and rm first item of lst1
        else:
            result_list.append(lst2.pop(0))  #append and rm first item of lst2

    result_list.extend(lst1)
    result_list.extend(lst2)

    return result_list

#input lists must be already sorted themselves
print make_one_sorted_list([1, 2, 4, 7], [3, 5, 6, 8])
#output => [1, 2, 3, 4, 5, 6, 7, 8]
```

MERGE SORT

BUT FIRST: MERGE

This merge solution is generalizeable for any two pre-sorted lists!

AWESOME!!

But how can we guarantee a pre-sorted list??

MERGE SORT

BASE CASE

What kind of list can we KNOW is already pre-sorted?

A list with one item is always sorted.

MERGE SORT

How to get down to where every list is one item long?

RECURSION

RECURSION

RECURSION

RECURSION

RECURSION

RECURSION

Any volunteers?

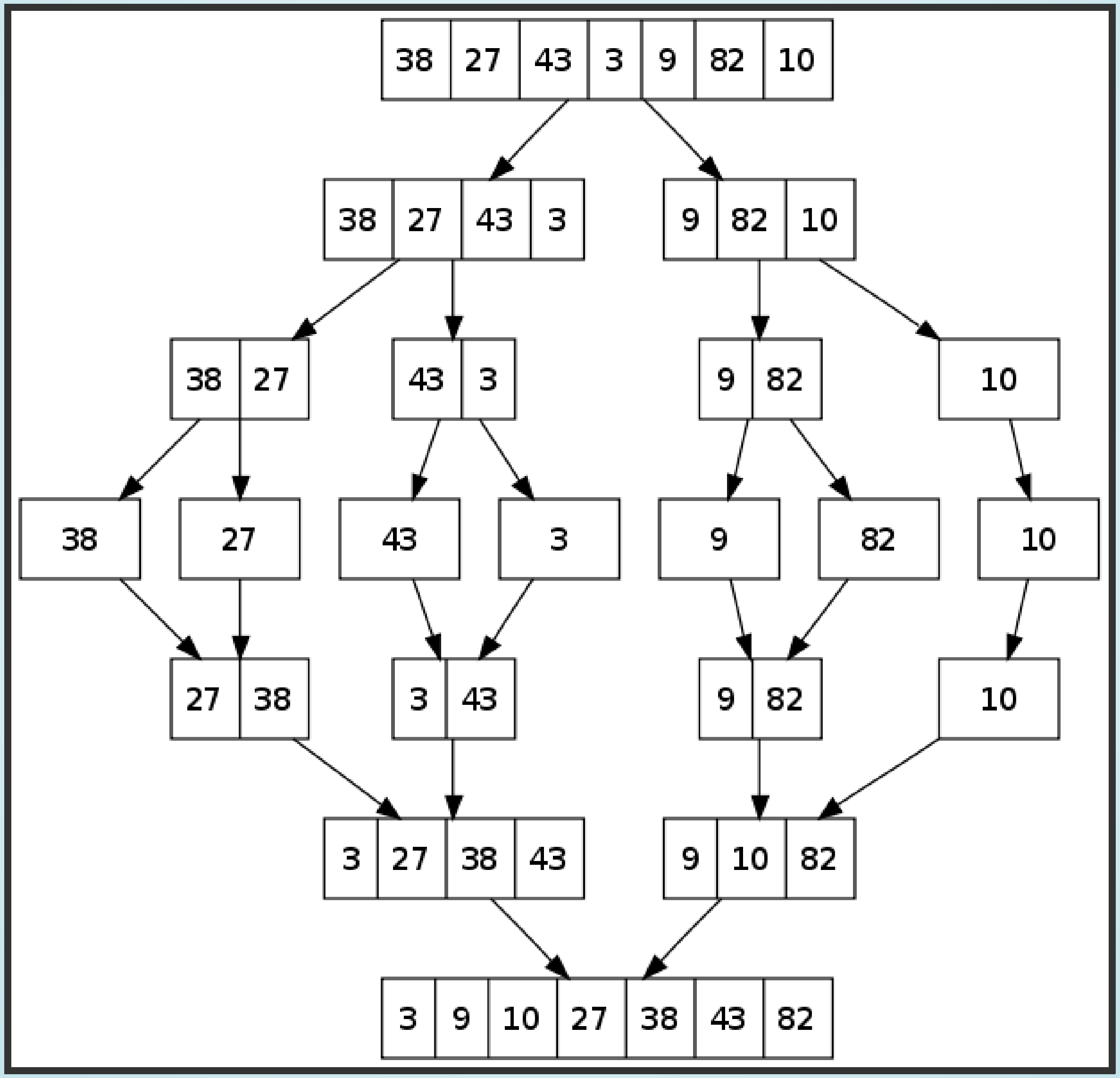
MERGE SORT

MAKE EVERYTHING A LIST OF ONE

```python
def make_everything_a_list_of_one(lst):
    if len(lst) < 2:      #if length of lst is 1, return lst
        print lst,
        return lst
    mid = int(len(lst)/2)     #index at half the list
    make_everything_a_list_of_one(lst[:mid])      #divide list in half
    make_everything_a_list_of_one(lst[mid:])      #assign other half


lst2 = [3, 5, 6, 8]
make_everything_a_list_of_one(lst2)      #outputs => [3] [5] [6] [8]
```

MERGE SORT



38 27 43 3 9 82 10
38 27 43 3
9 82 10
38 27
43 3
9 82
10
38
27
43
3
9
82
10
27 38
3 43
9 82
10
3 27 38 43
9 10 82
3 9 10 27 38 43 82

MERGE SORT

YOUR MISSION:

Combine these to make merge sort

Bonus: Don't decrease length of merging lists (don't use pop)

MERGE SORT

STATS

Runtime?

O(n log(n))

Runspace?

O(n)

This means that there's n (the length of the list) extra space needed to complete the algorithm. This takes the form of the results list.

OTHERS WITH O(N LOG(N))?

- Quick Sort

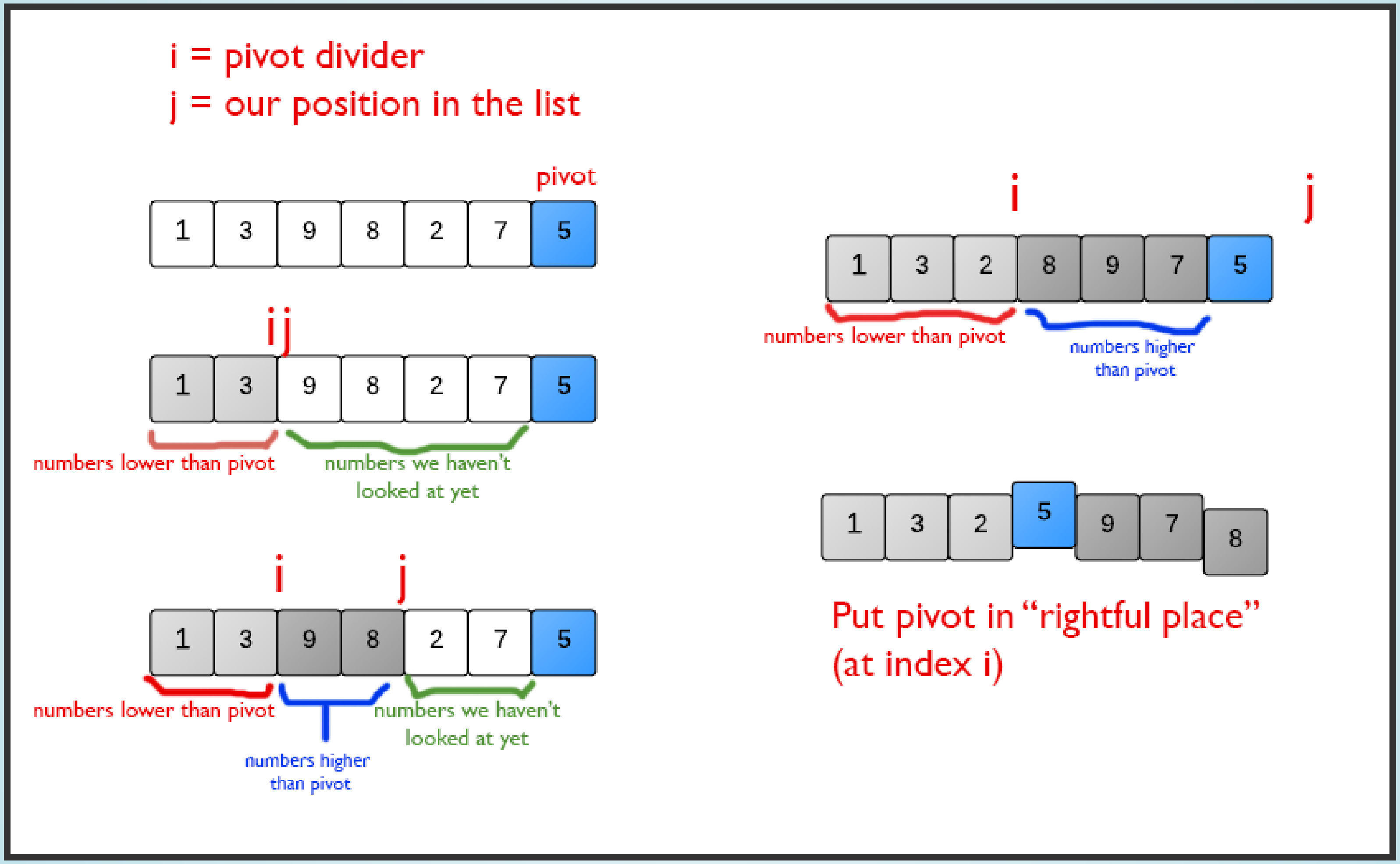
PLUS, operates in place

QUICK SORT (QUICKLY)

Operates on the idea of a partition

That is, there is a 'pivot' and we can move all the numbers lower than the pivot number to the beginning of the list and move all the numbers bigger than the pivot to the right of the pivot number.

PIVOT


i = pivot divider
j = our position in the list

Put pivot in "rightful place" (at index i)

FURTHER RESOURCES:

- Quicksort intro: https://www.youtube.com/watch?v=aQiWF4E8flQ (6 min)
- Tim Roughgarden Quicksort: https://class.coursera.org/algo-006/lecture (Quicksort-Algorithm, first two lectures)
- "An Intuitive Explanation of Quicksort" http://www.quora.com/What-is-an-intuitive-explanation-of-QuickSort

WHY WOULD YOU CHOOSE ONE OVER THE OTHER?

- Runtime
- Space requirements (how much more space does it need)
- Likely structure of your data:
  - Random?
  - Almost reversed?
  - Almost sorted?
  - Likely Duplicates?

EXERCISES

- Write bubble sort
  - BONUS: Write bubble sort in O(n^2/2) time
- Write merge sort
  - BONUS: Don't change length of sublists (don't use pop)
- Write quick sort

:)

RESOURCES

- Animated Sort Algorithms (compare with gifs!)
- Hungarian Dance sorting:
    - Bubble Sort
    - Quick Sort