On your way in...

Pick-Up:
1. Nothing!
Welcome to CS 134!

Introduction to Computer Science
Iris Howley

- Recursive Sorting-

Spring 2019
DOING FOR ROBOTS WHAT NATURE DID FOR US

LESLIE KAELBLING

Strategies for intelligent robot design.

Today at 2:35p in Wege (here)
Kate Bouman, PhD

One of the creators of the algorithm to combine the data of 8 radio telescopes around the world

Photographing a Black Hole

• Data processing, algorithm discussed in paper “First M87 Event Horizon Telescope Results. III. Data Processing and Calibration”

• Cites some interesting python modules:
  ▪ Numpy (van der Walt et al. 2011)
  ▪ Scipy (Jones et al. 2001)
  ▪ Pandas (McKinney 2010)
  ▪ Jupyter (Kluyver et al. 2016)
  ▪ Matplotlib (Hunter 2007)

• Reddit is “pretty sure” it’s python & matplotlib in the previous Facebook photo

https://iopscience.iop.org/article/10.3847/2041-8213/ab0c57
SORTING

Organizing data efficiently.
Bubble Sort

https://youtu.be/lyZQPjUT5B4?t=40

(note: they implemented some shortcuts for handling sorted elements at the end of the list, this is not typical of bubble sort)

https://en.wikipedia.org/wiki/Bubble_sort#/media/File:Bubble-sort-example-300px.gif

6 5 3 1 8 7 2 4
Let’s make it stop when we run out of things to sort!

```python
def bubbles(l):
    n = len(l)
    sorted = 0
    done = False
    while (sorted < n-1) and (not done):
        swapped = False
        for left in range(0, n-sorted-1):
            right = left + 1
            if l[left] > l[right]:
                swapped = True
                swap(l, left, right)
        sorted += 1
        done = not swapped

    return l
```

Not done yet! Haven’t started

Stop when we went thru list w/o swapping!

When we begin sorting pairs, no swaps yet

We’ve swapped, set True

We’re done if we’ve not swapped values
def bubbles(l, key=None):
    n = len(l)
    sorted = 0
    done = False
    while (sorted < n-1) and (not done):
        swapped = False

        for left in range(0, n-sorted-1):
            right = left+1
            if l[left] > l[right]:
                swapped = True
                swap(l, left, right)

        sorted += 1
        done = not swapped

Let’s implement a key function for our sort!
Sorting Tools

- `def byRank` is a simple, one-expression function with this one purpose!
- ...lambda functions (i.e. anonymous functions)

- `rl = sorted(ranks, key=lambda pair:pair[1])`
- Compare to:
  - `rs = sorted(ranks, key=byRank)`
  - `def byRank(pair):
      return pair[1]`

Reminder!

Can provide a lambda function
...or any named function
Functions as Objects

- `dogs = ['pixel', 'tally', 'linus', 'wally']`

- `def justDog(d):
  return d + " dog"

- `def printDog(dList, strFunction):
  for d in dList:
    print(strFunction(d))

- `>>> printDog(dogs, justDog)`

...because functions are objects

They can be passed like objects

And stored in variables

And called by their variable name
Key for Bubblesort

- hockey = [('williams', 14, 9, 2), ('amherst', 15, 7, 4), ('middlebury', 8, 13, 4)]

- bubble(hockey, key=lambda tup: tup[1])

- print(hockey)
  - [('middlebury', 8, 13, 4), ('williams', 14, 9, 2), ('amherst', 15, 7, 4)]
def bubbles(l, key=None):
    n = len(l)
    sorted = 0
    done = False
    while (sorted < n-1) and (not done):
        swapped = False
        for left in range(0, n-sorted-1):
            right = left+1
            if l[left] > l[right]:
                swapped = True
                swap(l, left, right)
        sorted += 1
        done = not swapped

How do we implement this?
Where should we use this key?
Key for Bubblesort

def bubble(l, key=None):
    if key is None:
        key = lambda x: x
    n = len(l)
    sorted = 0
    while sorted < n-1:
        swapped = False
        # perform a pass of finding the max:
        for left in range(0, n-sorted-1):
            # number of values: n-sorted; nu. of comparisons
            right = left + 1
            if key(l[left]) > key(l[right]):
                swapped = True
                swap(l, left, right)  # l[left], l[right] = l[right], l[left]
            # the maximum value is now clear to the right
            # we can reduce the problem
            sorted += 1
        if not swapped:
            break

Use key instead of relying on [left].__gt__(l[right])
Setting a Default Lambda Function

- `def bubble(l, key=None):
  - if key is None:
    - key = lambda x : x
  - ...

This approach is safer, better.
Comparison

• We need to be able to compare values in order to sort them!

• If you’re sorting a list of your own data types (like, say, Color) you’re going to want to implement some special methods for your class:
  ▪ `def __gt__(self):`  # greater than operator >
  ▪ `def __eq__(self):`  # equal to operator ==

• There’s another way to do this, we’ll talk about it on Friday.
  ▪ Provide a key function that produces a value that can be compared!
Comparing Colors with a Key Function

```python
def byBrightness(c):
    """ given a color, c, return the brightness of the color """
    return sum(c.rgb()) / 3

if __name__ == "__main__":
    from sort import bubble
    from random import randint, random

    # Two color lists
    rgblist = [ Color( (randint(0,1), randint(0,1), randint(0,1)) ) for _ in range(0,5) ]
    colorlist = [ Color( (random(), random(), random()) ) for _ in range(0,5) ]

    # colorlist is just 0s and 1s
    print("before", rgblist)
    bubble(rgblist, byBrightness)
    print("\nafter", rgblist)
```
SELECTION SORT

Another simple way to sort.
Selection Sort

1. Will place sorted items at front of list
2. Goes through list, and finds the lowest number [comparison]
3. Place lowest object at front of list
4. Goes through list (except first/sorted) item, and finds the next lowest number
5. Places it at front of list

Selection Sort

- Red is lowest, blue it’s comparing to, yellow is sorted
- With a partner, answer the following questions:
  1. When does Selection Sort stop sorting?
  2. Does Selection Sort have an outer loop? And an inner loop?
  3. What do these loop(s) do?
  4. What is the run-time for Selection Sort?
Selection Sort

• With a partner, answer the following questions:

1. When does Selection Sort stop sorting?
   - When it runs out of unsorted elements, when it’s found an item for every index

2. Does Selection Sort have an outer loop? And an inner loop?
   - Yes.

3. What do these loop(s) do?
   - Inner – looks for lowest number, swaps w. whatever’s at our next sorted spot
   - Outer – Runs inner loop for each index in our list

4. What is the run-time for Selection Sort? \( O(n^2) \) [both Best & Worst case]
Selection Sort

def selectionSort(d, key=None):
    if key is None:
        key = lambda x: x  # key is naturally ordered value
    n = len(d)
    sorted = 0

    while sorted < n-1:
        first = sorted  # location of first possibly unsorted item
        minLocation = first
        Inner loop – start at the first unsorted, look until the end
        for i in range(first, n):
            if d[minLocation] >= d[i]:  # update the location of lowest value, if we found a new low value
                minLocation = i
            swap(d, minLocation, first)
        sorted += 1

        Outer loop – ends when we’ve found a sorted item for each index
If we got this far, then swap our first unsorted element with the low value
Would you use selection sort to sort a deck of cards?
Sorting Algorithm Run-times

- **Insertion Sort**
  - Best Case: $O(n)$
  - Worst Case: $O(n^2)$

- **Bubble Sort**
  - Best Case: $O(n)$
  - Worst Case: $O(n^2)$

- **Selection Sort**
  - Best Case: $O(n^2)$
  - Worst Case: $O(n^2)$
QUICK SORT

More [recursive] sorting!
Quicksort

• Sort these lists:

7

• if $n < 2$:
  ▪ Stop!

When $n = 0$, or $n = 1$
Quicksort $n = 2$ *

- Check if first element is smaller than the second
- If it isn’t, swap them

*After next slide: note that this step is actually picking the leftmost item to be the pivot, and then partitioning into a [3] “less-than” sublist and an empty “greater-than” sublist
Quicksort $n = 3$

- Pick an element, we call it the *pivot*
  - Let’s start with the leftmost item as the pivot
Quicksort $n = 3$

- Find all the elements smaller than the pivot, and all the elements larger than the pivot
  - We call this *partitioning*

- We now have a sub-list of all elements smaller, and a sub-list of all elements larger

- The two sub-lists are not sorted! Just partitioned based on our pivot
Quicksort \( n = 3 \)

- If they were sorted, then we could just combine the lowerList + pivot + greaterList (but we haven’t sorted them yet)
- We call quicksort on our lower-list [3] and it’s only one element! We know how to sort that!
- We call quicksort on our greater list [7], and ...ditto
- Now we can combine lowerList + pivot + greaterList
Quicksort \( n > 2 \)

- Pivot is 5
- Partition!
- We have a lower list of 2 and a greater list of 1, we know how to sort those!
  - Call quicksort again, on each
- Sorted!
Quicksort

After each quicksort call:

We know our pivot is sorted, but nothing else!

Combine all sub-lists at the end!
How many levels of quicksort calls?
How many levels for a list of length 8?
Pre-sorted Quicksort

• 2 elements = 2 levels of calls to quicksort
• 4 elements = 4
• 8 elements = 8
• 16 elements = 16

• What is this growth rate?
  ▪ Number of levels grows by $O(n)$

Levels of recursive calls to quicksort works a little like our “outer loop” for iterative sorting methods.
Quicksort

During partitioning, has to look at each element to see if it’s less than or greater than the pivot!

For each call to quicksort, how many operations?

As always, this is technically: $n$

But we drop the constants!
Quick Sort Operations

• n operations for each call

• **What is this growth rate?**
  ▪ Number of operations at each level grows by $O(n)$
  ▪ Add another level? Add another n operations

• What is the run-time of Quick Sort when the list is sorted?
  ▪ $O(\text{Number of calls} \times \text{Number of operations/call})$
  ▪ $O(n \times n)$
  ▪ $O(n^2)$

$O(n^2)$ isn’t great.

(Especially for a recursive algorithm)
Let’s pick a random pivot!

How many levels quicksort calls?
How many call levels for a list of length 8?

1 2 3 4 5 6 7 8

1 2 3 4 5 6 7 8

1 2 3 5 6 7 8

8?

4!
Pre-sorted Quicksort

• 2 elements = 1 levels of calls to quicksort
• 4 elements = 2
• 8 elements = 3
• 16 elements = 4

• What is this growth rate similar to?
  ▪ Paper folding from Wednesday!
  ▪ Number of levels grows by $O(\log n)$
Quick Sort Operations

• Number of operations?
  ▪ n operations for each call (still)

• What is this growth rate?
  ▪ Number of operations at each level grows by $O(n)$
  ▪ Add another level? Add another n operations

• What is the run-time of Quick Sort when the list is sorted?
  ▪ $O($Number of calls * Number of operations/call$)$
  ▪ $O(n \times \log n)$
  ▪ $O(n \log n)$ **Best case scenario!**

**Worst Case?** We already saw it, sorted list with a bad pivot selected repeatedly: $O(n^2)$
Sorting Algorithm Run-times

- **Insertion Sort**
  - Best Case: $O(n)$
  - Worst Case: $O(n^2)$
- **Bubble Sort**
  - Best Case: $O(n)$
  - Worst Case: $O(n^2)$
- **Selection Sort**
  - Best Case: $O(n^2)$
  - Worst Case: $O(n^2)$
- **Quick Sort**
  - Best Case: $O(n \log n)$
  - Worst Case: $O(n^2)$
Quick Sort

https://www.youtube.com/watch?v=ywWB6J5gz8
If you like this topic (i.e., algorithms)...

- Enroll in CS136 Data Structures!
  - ...take CS256 Algorithm Design & Analysis
  - ...take CS361 Theory of Computation

- See “Grokking Algorithms” by Bhargava

- “Algorithms to Live By: The Computer Science of Human Decisions” is a lighter read by Christian and Griffiths
QUESTIONS?
Leftover Slides
3 4 2 5
Bubble Sort

• If you watch carefully, you’ll notice two loops here (much like insertion sort):

1. An *inner loop* that goes through the unsorted portions of the list and compares pairs all the way across

2. An *outer loop* that keeps calling the inner loop, until there aren’t any unsorted portions left
Bubble Sort

https://youtu.be/lyZQPjUT5B4?t=40
(note: they implemented some shortcuts for handling sorted elements at the end of the list, this is not typical of bubble sort)

https://en.wikipedia.org/wiki/Bubble_sort#/media/File:Bubble-sort-example-300px.gif
Swapping

```python
def swap(l, left, right):
    l[left], l[right] = l[right], l[left]
```

Is equivalent to (and more pythonic than):

```python
def swap(l, lt, rt):
    tmp = l[lt]
    l[lt] = l[rt]
    l[rt] = tmp
```
Comparison

• We need to be able to compare values in order to sort them!

• If you’re sorting a list of your own data types (like, say, Color) you’re going to want to implement some special methods for your class:
  • `def __lt__(self):` # less than operator <
  • `def __eq__(self):` # equal to operator ==

• There’s another way to do this, we’ll talk about it on Friday.
def bubble(l):
    n = len(l)
    sorted = 0

    while sorted < n-1:  # Outer loop
        for left in range(0, n-sorted-1):
            right = left+1
            if l[left] > l[right]:
                swap(l, left, right)

        sorted += 1
def bubble(l):
    n = len(l)
    sorted = 0

    while sorted < n-1:
        for left in range(0, n-sorted-1):
            right = left+1
            if l[left] > l[right]:
                swap(l, left, left, right)

        sorted += 1

1. Stops when we’ve checked every element
2. Goes thru the list, looking at pairs.
3. Swap if needed.
4. Increase our count of sorted elements, so we don’t check it (the black shaded box in the example)
Something’s missing here...

When does this stop trying to sort?

```python
def bubble(l):
    n = len(l)
    sorted = 0
    while sorted < n-1:
        for left in range(0, n-sorted-1):
            right = left+1
            if l[left] > l[right]:
                swap(l, left, right)
        sorted += 1
```

What’s the best/average/worst case for run-time?

$O(n^2)$

It’s all the same!
def bubbles(l):
    n = len(l)
    sorted = 0
    done = False
    while (sorted < n-1) and (not done):
        swapped = False
        for left in range(0, n-sorted-1):
            right = left + 1
            if l[left] > l[right]:
                swapped = True
                swap(l, left, left, right)
        sorted += 1
        done = not swapped

    Not done yet! Haven’t started
    Stop when we went thru list w/o swapping!
    When we begin sorting pairs, no swaps yet
    We’ve swapped, set True
    We’re done if we’ve not swapped values
Bubble Sort

If stopping when swaps are done:

<table>
<thead>
<tr>
<th>Best Case</th>
<th>Average Case</th>
<th>Worst Case</th>
</tr>
</thead>
<tbody>
<tr>
<td>$O(n)$</td>
<td>$O(n^2)$</td>
<td>$O(n^2)$</td>
</tr>
</tbody>
</table>

We compare each element to only the next element (sorted) We compare each element to half the other elements We compare each element to all the other elements (reverse sorted)

If running the outer loop $n-1$ times:

- **Best Case** is the same as **Worst Case**!
- We compare each element to all the other elements $O(n^2)$