On your way in...

Pick-Up:
1. HW 8
2. Lecture 22 Notes
3. Graded HW6

Drop-off (2 piles):
1. HW7
Welcome to CS 134!

Introduction to Computer Science

Iris Howley

-Sorting-

Spring 2019
Do you want to be a Computer Science Teaching Assistant this fall?

• TA applications due April 12 (Friday)

• [https://csci.williams.edu/tatutor-application/](https://csci.williams.edu/tatutor-application/)

• You’ll need 2 faculty references
  • Duane or Iris should be at least one of those!
  • Come talk to us!
Suppose we want to create a 4x4 grid on a piece of paper:

If drawing one square is considered one operation, how many operations did this algorithm take?

Is there a faster way?
Suppose we want to create a 4x4 grid on a piece of paper:

1. Take today’s lecture notes, or scrap paper
2. Fold it in half.
   - We made 2 boxes!
3. Fold it in half again!
   - How many boxes? 4
4. Fold in half again!
   - How many boxes? 8
5. Fold in half again!
   - 4 folds total 16
6. Open the paper. How many boxes do we have?

If folding once is considered one operation, how many operations did this algorithm take to make 16 boxes?
Describe Growth of the Folding Algorithm

- 1 Fold = 2 boxes
- 2 Folds = 4 boxes
- 3 Folds = 8 boxes
- 4 folds = 16 boxes

What is the mathematical relationship to predict the number of folds[operations] based on number of boxes[elements]?

\[ f = \log_2 b \]

\[ 2^f = b \]
Logarithms

- The flip of exponents
- \( \log_{10}100 = ?? \)
  - How many 10s do we multiply together to get 100?
  - \( \log_{10}100 = 2 \)
- In computer science “log 8” most often implies “log\(_2\) 8”
  - \( 2^x = 16 \) is equivalent to \( \log_2 16 = x \)
  - \( 2^4 = 16 \) is equivalent to \( \log_2 16 = 4 \)
  - \( 2^6 = 64 \) is equivalent to \( \log_2 64 = 6 \)

O(log n)

- When you double the number of elements, it only increases the number of operations by 1
- 2 items in the list, 1 operation
  - \( \log 2 = 1 \)
- When you have 4 items, increases operations to 2
  - \( \log 4 = 2 \)
- When you have 8 items, only 3 operations
  - \( \log 8 = 3 \)
Big-O Notation

- Tells you how fast an algorithm is / the run-time of algorithms
  - But not in seconds!
- Tells you how fast the algorithm grows in number of operations

$O(\log n)$

“Big O” Number of operations
Common Big-O Run-times

• $O(\log n)$: “log time”
  - Ex: paper folding, binary search

• $O(n)$: “linear time”
  - Ex: linear/sequential search

• $O(n^2)$
  - Ex: ordered insert sort from our LinkedList

• $O(n \log n)$
  - Ex: quicksort

• $O(n!)$
  - Ex: Really, really slow algorithms like traveling salesperson
Common Big-O Run-times
BIG-O NOTATION

Comparing number of operations to select the best algorithm.
Ordered Insert Sort

- We implemented ordered insert sort in LinkedList

```
2 5 3 7 1
```
Ordered Insert Sort

• We implemented ordered insert sort in LinkedList
Ordered Insert Sort

- We implemented ordered insert sort in LinkedList
Ordered Insert Sort

- We implemented ordered insert sort in LinkedList

```
2  5  3  7  1
```

```
2  3  5  7
```
Ordered Insert Sort

• We implemented ordered insert sort in LinkedList

2 5 3 7 1

1 2 3 5 7
Ordered Insert Sort

- We implemented ordered insert sort in LinkedList

What is the worst case for this implementation?
A sorted list.
Ordered Insert Sort

- Our ordered insert sort:
  
  Worst case = Sorted List
  We compare every element of our list to all the other elements of our list.
  What describes this growth rate?
  \[ O(n^2) \]
Ordered Insert Sort

• Our ordered insert sort:

Best case?

Best case = Reverse sorted list

We have to compare every element of our list to one item.

What describes this growth rate?

O(n)
Ordered Insert Sort

Best Case
O(n)

We compare each element to only the first element

Average Case
??

We compare each element to half the other elements

Worst Case
O(n^2)

We compare each element to all the other elements

Computer Scientists don’t care about constants, it doesn’t significantly change the growth curve

- O(n * n)
- O(n^2)
- O(n^2) / 2
Ordered Insert Sort

Best Case

\[ O(n) \]

Reverse sorted

Worst Case

\[ O(n^2) \]

This is not ideal.

Sorted
Insertion Sort

• We’ve implemented ordered insert sort with a new list, `newList`
• But you can also implement something similar, Insertion Sort, “in place”
• See this animation:
  ▪ [https://upload.wikimedia.org/wikipedia/commons/0/0f/Insertion-sort-example-300px.gif](https://upload.wikimedia.org/wikipedia/commons/0/0f/Insertion-sort-example-300px.gif)

Typical Insertion Sort

Best Case  
*Sorted*  
Compare to only first element  
$O(n)$

Worst Case  
*Reverse Sorted*  
Compare to all other elements  
$O(n^2)$
SORTING

Organizing data efficiently.
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

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

Is equivalent to (and more pythonic than):

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:
        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, 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, 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, left, right)
        sorted += 1
        done = not swapped

    return True
# 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
  - $O(n^2)$
DIFFERENT IMPLEMENTATION DECISIONS WILL IMPACT RUN-TIME OF THE ALGORITHM
Leftover Slides
Sorting

• Other ways of sorting, we used “Insertion Sort”

• Bubble Sort:
  ▪ Swaps adjacent pairs of elements from a list repeatedly

• Merge Sort:
  ▪ Split the list into 2 sublists, sort the sublists, and then merge them
Linked Lists

- Today, we’re only going to look at sorting our LinkedList

- See example code in shared/examples/04.08!

- Lecture notes from 4/3 and 4/5 are also useful!
Sorting LinkedLists

- **class Element:**
  - **def orderedInsert**(self,v):
    - """Inserts v in ordered list, returns new list."""
    - if v <= self.value:
      - return Element(v,self)
    - elif not self.next:
      - self.next = Element(v)
      - return self
    - else:
      - self.next = self.next.orderedInsert(v)
      - return self

- **class LinkedList:**
  - **def sort**(self):
    - """Sort a list of values."""
    - newList = None
    - for item in self:
      - if newList is None:
        - newList = Element(item)
      - else:
        - newList = newList.orderedInsert(item)
    - self._head = newList
Ordered Insert Sort

```python
ll = [3, 2, 1]

if newList is None:
    newList = Element(item)

if v <= self.value:
    return Element(v, self)
```

ll = [3, 2, 1]

if newList is None:
    newList = Element(item)

if v <= self.value:
    return Element(v, self)

ll = [1, 2, 3]
Ordered Insert Sort

```python
ll = [1, 3, 2]

if newList is None:
    newList = Element(item)

elif not self.next:
    self.next = Element(v)
    return self

else:
    self.next = self.next.orderedInsert(v)
    return self

---

if v <= self.value:
    return Element(v, self)
```
Ordered Insert Sort

```python
newList = None
if newList is None:
    newList = Element(item)

elif not self.next:
    self.next = Element(v)
    return self

elif not self.next:
    self.next = Element(v)
    return self
```

ll = [1, 2, 3]

---

1 2 3

v

1 1

2 2

3 3
Ordered Insert Sort

- How many comparisons are we making to do this sort?

- `ll = LinkedList()`
- `ll.extend(1, 2, 3)`
- `ll.sort()`

\[ n = \text{len}(ll) \]

For each element of `ll`, we have \( n-1 \) comparisons in the worst case.

Computer Science drops the -1
Ordered Insert Sort

• How many comparisons are we making to do this sort?

• ll = LinkedList()
• ll.extend(1,2,3)
• ll.sort()

For each element of ll $\rightarrow n$

We have $\sim n$ comparisons in the worst case $\rightarrow \ast n$

$O(n^2)$ comparisons
HOW DO YOU SORT A DECK OF CARDS?