Today:
More sorting

files
Sorting Algorithms

the *fruit flies* of complexity theory...

**Checksort**

check permutations until sorted

```
Check: 3 1 2
   3 2 1
   1 2 3
   1 3 2
   2 1 3
   2 3 1
```

```
SlowSort!
pull # s from a hat
with replacement...

```

Slow:
1 1 1
1 1 2
1 1 3
1 2 1
1 2 2
1 2 3
1 3 1
1 3 2
1 3 3
2 1 1
2 1 2
2 1 3
2 2 1
2 2 2
2 2 3
2 3 1
2 3 2
2 3 3
3 1 1
3 1 2
3 1 3
3 2 1
```

"bogosort"

---

**Bogosort**

From Wikipedia, the free encyclopedia

*Bogosort* is a particularly ineffective sorting algorithm.
Sorting Algorithms: Can we do better?

Yes!

Fun for all ages
Sorting Algorithms

The fruit flies of complexity theory...

Minsort
choose the min each iteration

\[ 7 \ 4 \ 3 \ 2 \ 1 \ 5 \ 0 \ 6 \]
An algorithm of a similar sort...

Insertion Sort
insert elements in order to the left

7 4 3 2 1 5 0 6

vs. minsert?
better than O(N^2)?
< $N^2$ run times… ?

Shell Sort

sort column-sublists of increasing size

7 4 3 2 1 5 0 6 1
Shell Sort

sort column-sublists of increasing size

\(<O(N^2)\) via 2d sorting!

2 elements per column
$O(N^2)$ via 2d sorting!

**Shell Sort**

sort column-sublists of increasing size

<table>
<thead>
<tr>
<th>7</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>0</td>
<td>6</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>5</th>
<th>0</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>2</td>
</tr>
</tbody>
</table>

2 elements per column

<table>
<thead>
<tr>
<th>5</th>
<th>0</th>
<th>3</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>4</td>
<td>6</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>1</th>
<th>0</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>

3 elements per column
<O(N²) via 2d sorting!

Shell Sort
sort column-sublists of increasing size

2 elements per column

3 elements per column

4 el. per col.
O($N^{3/2}$) or better

depending on the skip sequence

Shell Sort

sort column-sublists of increasing size

| 7 4 3 2 1 5 0 6 1 |

2 elements per column

| 7 4 3 2 1 |
| 5 0 6 1 |

| 5 0 3 1 1 |
| 4 6 2 |

3 elements per column

| 1 0 |
| 1 1 7 |
| 4 6 2 |

4 el. per col.

| 1 0 |
| 2 4 |
| 1 3 |

N el. per col.

| 1 0 |
| 2 4 |
| 5 6 |
| 7 |
Divide and Conquer

Mergesort
merge sorted lists together

| 7 | 4 | 3 | 2 | 1 | 5 | 0 | 6 |
How many steps are needed? (usually in the worst case...)

\[ T(N) = \text{running time as a function of problem size, } N \]

**MM algorithm:**

The problem size is the list length, N.
Each comparison (< or >) will count as 1 step

```
for ( i=1 ; i < N ; ++i )
{
    if (L[i] > E) E = L[i];
    if (L[i] < e) e = L[i];
}
```

**Running Times:**

\[ T(N) = \]
Running Time: Loop counting

Nested loops will produce a series …

```c
for ( int i=0 ; i<N ; ++i )
{
    for ( int j=0 ; j<(i*i) ; ++j )
    {
        1 time step of work here...
    }
}
```

but not always $O(N^2)!$
### Algorithms and Impatience

**Problem:** Search through an array to find max and min

| 5 | 2 | 3 | 8 | 4 | 1 | 6 | 7 |

- **Divide-and-conquer!**
- **E**
- **e**
- **N=8**
- **N** is the problem size

- **hubris**
- **laziness**

- better or worse than looping?
- in absolute terms? / big-O terms?
Running Time: Recurrences

T(N) = running time as a function of problem size, N

Recursive algorithms lead to *recurrence relationships*:

base case: \( \text{MMRec}(L) : \)

if \( L == [x,y] \) and \( x > y \) then
return \([x,y]\) else return \([y,x]\);

recursion: \( \text{MMRec}(L) : \)

\([E_1,e_1] = \text{MMRec}( \text{half1}(L) ) , \)
\([E_2,e_2] = \text{MMRec}( \text{half2}(L) ) , \)
if \( E_1 > E_2 \) then \( E=E_1 \) else \( E=E_2 \),
if \( e_1 < e_2 \) then \( e=e_1 \) else \( e=e_2 \),
return \([E,e]\);

Running Time: \( T(2) = 1 \)
\( T(N) = \)
Recurrences

Recurrence relation from the recursive max/min finding program...

\[ T(2) = 1 \]
\[ T(N) = 2 \ T\left(\frac{N}{2}\right) + 2 \]

what is \( T(N/2) \)?

recursive vs. looping?
big-O?
Mergesort

Mergesort merges sorted lists together

7 4 3 2 1 5 0 6

Recurrence relation for mergesort:

Big-O running time:
Any *Comparison Sort* is at best $O(n \cdot \log(n))$

Start with all of the permutations of an n-element list.

$[e_1, e_2, e_3, \ldots, e_n]$ 

Sorting is equivalent to determining *which* of the permutations we started with…

The "adversary" doesn't seem to do all that much…
Any *Comparison Sort* is at best $O(n \cdot \log(n))$

Start with all of the permutations of an n-element list. $[e_1, e_2, e_3, \ldots, e_n]$

Sorting is equivalent to determining *which* of the permutations we started with…

The Adversary

```
Is e1<e2?
```

```
OK, is e1<e3?
```

“Adversary Argument”
Any \textit{Comparison Sort} is at best $O(n \cdot \log(n))$

Start with all of the permutations of an $n$-element list. $[e_1, e_2, e_3, \ldots, e_n]$

Sorting is equivalent to determining \textit{which} of the permutations we started with…

The Adversary

“Adversary Argument”

Ah, this adversary is tricky after all!
Any *Comparison Sort* is at best $O(n \cdot \log(n))$

Start with all of the permutations of an $n$-element list. $[e_1, e_2, e_3, \ldots, e_n]$

```
[1,2,3]      [1,3,2]      [2,3,1]      [3,2,1]
[2,1,3]      [3,1,2]      [1,2,3]      [2,1,3]
[3,1,2]      [3,2,1]      [3,1,2]      [3,2,1]
```

**The Adversary**

```
[1,3,2]      [2,3,1]      [3,1,2]
[1,2,3]      [1,2,3]      [1,3,2]
```

```
[1,3,2]      [2,3,1]
[1,2,3]      [1,2,3]
```

**“Adversary Argument”**

```
[1,3,2]      [2,3,1]
[1,2,3]      [2,3,1]
```

$h \geq \log_2(n!)$
Fun with Logs and Big-Omega

\[
\left(\frac{n}{2}\right)^\frac{n}{2} < n!
\]

\[
\log\left(\frac{n}{2}\right)^\frac{n}{2} < \log n! \leq h
\]

So I guess omega is sort of like an upside-down O.
HW7: Loops, loops and more loops!

- **for**
  - definite iteration
  - For a *known* number of iterations

- **while**
  - indefinite iteration
  - For an *unknown* number of iterations
Problem 1: the Mandelbrot Set

Consider the following update rule for all complex numbers c:

\[ z_0 = 0 \]

\[ z_{n+1} = z_n^2 + c \]

If \( z \) does not diverge, then \( c \) is in the M. Set.
Consider the following update rule for all complex numbers $c$:

$$z_0 = 0$$

$$z_{n+1} = z_n^2 + c$$

If $z$ does not diverge, then $c$ is in the M. Set.

**Problem 1: the Mandelbrot Set**
Problem 1: the Mandelbrot Set

Consider the following update rule for all complex numbers $c$:

$z_0 = 0$

$z_{n+1} = z_n^2 + c$

The shaded area are points that do not diverge.
Python and images

from bmp import *  

for creating and saving images

image = BitMap( 300, 200, Color.GREEN )

creates a bitmap object and names it image
Python and images and objects

from bmp import *

image = BitMap( 300, 200, Color.GREEN )

here, a bitmap object named image
is calling an internal method named saveFile

image.saveFile( "test.bmp" )
from bmp import *

image = BitMap( 300, 200, Color.GREEN )

image.setPenColor( Color.Red )

image.plotPixel( 150, 100 )

image.saveFile( "test.bmp" )

**objects** are variables that can contain their own functions, often called **methods**
from bmp import *

def test():
    """ image demonstration """
    width = 200
    height = 200
    image = BitMap( width, height )

    # a 2d loop over each pixel
    for col in range(width):
        for row in range(height):
            if col == row:
                image.plotPoint( col, row )

    image.saveFile( "test.bmp" )
def menu():
    choice = 1  # Anything except 9
    while choice != 9:
        print("I see danger in your future...")
        printMenu()
        choice = input("Make your choice ")

    print("The inner eye can see no more")

def printMenu():
    print("What would you like to do?")
    print("\t 0: See another prediction")
    print("\t 9: Quit")

"\t" represents a tab
def menu():
    while True:
        print "I see danger in your future..."
        printMenu()
        choice = input("Make your choice ")
        if choice == 9:
            break
        print "The inner eye can see no more"

def printMenu():
    print "What would you like to do?"
    print "\t 0: See another prediction"
    print "\t 9: Quit"

break stops the execution of the current loop

I'll figure out later how to get out of this loop!

OK - I'll stop the loop now and continue with the code after the loop
'Cause somethin' like he left knee and a harp," said he had to the whole school? The shouting and then some strange and Mrs. "Well, I know Hagrid; they spotted handkerchief and get him get rid of course, had a gigantic beet with her," he knew what to all he's

All the sky with the sun in the sun in the church where you're gone Lucy in my eyes. There beneath the girl with an hourglass And then the banker never wears a lot to hold your hand. Can't buy me tight, tight Owww! Love is love I can't hide,

**Who is the author?**

This is but ourselves. No, faith, My uncle! O royal bed of confession Of your rue for leave to nature; to this time I should weep for thy life is rotten before he is. have sworn 't. Or my blood. I have closely sent for nine; and unprofitable,

**What is the work?**

The Senators and the date of a written declaration that Purpose, they shall consist of nine States, shall not, when he shall have such Vacancies. The President pro tempore, in the Desire of a Qualification to the Speaker of the Senate. Article 6. When vacancies by the office upon probable
HW7 Pr 2: Read some text and automatically generate new (reasonable?) text

"Cause somethin' like he left knee and a harp," said he had to the whole school?
The shouting and then some strange and Mrs. "Well, I know Hagrid; they spotted
handkerchief and get him get rid of
course, had a gigantic beet with her," he
knew what to all he's

Who is the author? What is the work? What is going on?

This is but ourselves. No, faith, My uncle!
O royal bed of confession Of your rue
for leave to nature; to this time I should
weep for thy life is rotten before he is.
have sworn it. Or my blood. I have
closely sent for nine; and unprofitable,

The Senators and the date of a written
declaration that Purpose, they shall consist
of nine States, shall not, when he shall have
such Vacancies. The President pro tempore,
in the Desire of a Qualification to the
Speaker of the Senate. Article 6. When
vacancies by the office upon probable
Markov Model

Technique for modeling any sequence of natural data
Each item depends on only the item immediately before it.

The text file:
I like spam. I like toast and spam.
I eat ben and jerry's ice cream too.

For each word, keep track of the words that can follow it (and how often)

The Model:

| I: like, like, eat           | ben: and          |
| like: spam., toast          | jerry's: ice      |
| spam.: I, I                | ice: cream        |
| toast: and                 | too.             |
| eat: ben                   | cream: too.       |
| and: spam., jerry's        | too.:            |

- We can repeat words to indicate frequency
- Where to begin??
1st order Generative Markov Model

Technique for modeling any sequence of natural data
Each item depends on only the item immediately before it.
A key benefit is that the model can generate feasible data!

I like spam. I like spam. I like toast and jerry's ice cream too.

Generating text (1st order):

1) start with the first word in the file. Call it w
2) choose a word following w, at random. Call it w
3) choose a word following w, at random. And so on…
HW7 Pr 2: Need to be able to…

- Read text from a file
- Compute and store the model
- Generate the new text
In Python reading files is no problem…

```python
>>> f = file( 'a.txt' )

>>> text = f.read()

>>> text
'This is a file.\nLine 2\nLast line!\n'

>>> f.close()
```
In Python reading files is no problem...

```python
>>> f = file( 'a.txt' )
    opens the file and calls it `f`

>>> text = f.read()
    reads the whole file and calls it `text`

>>> text
'This is a file.\nLine 2\nLast line!\n'

>>> f.close()
    `text` is a single string containing all the text in the file
    But how to process the text from here…?

    closes the file (closing
    Python does the same)
String Manipulation

```python
>>> text
'This is a file.\nLine 2\nLast line!\n'
>>> print text
This is a file.
Line 2
Last line!

>>> text.split()
['This', 'is', 'a', 'file.', 'Line', '2', 'Last', 'line!']
>>> text
'This is a file.\nLine 2\nLast line!\n'
>>> lines = text.split('\n')
>>> lines
['This is a file.', 'Line 2', 'Last line!', '']
```

*Returns a list of the words in the string (splitting at spaces, tabs and newlines)*

*Returns a list of the lines in the string (splitting at newlines)*
Objects, objects, everywhere!

```python
>>> L = []  # create a list, L
>>> dir(L)  # see all of L's methods

all list methods appear -- lots of them including append, index, remove, and sort

>>> help(L.sort)  # I wonder...

>>> help(L.index)  # What does this do?
```
List methods

not including list operators

>>> dir([])
append
count
extend
index
insert
pop
remove
reverse
sort

String methods

not including string operators

>>> dir(''
capitalize
center
count
find
index
isalpha
lower
replace
split
strip
title
upper
and ~10 more…

help can help
but there's a fundamental difference… what and why?
Mutable vs. immutable objects

Lists are *mutable* objects.

```python
>>> L = [2,1,3]
>>> L.sort()
>>> L
[1,2,3]
```

Strings are *immutable* objects.

(So are numbers.)

```python
>>> s = 'string'
>>> s.replace('st','')
'ring'
>>> s
'string'
```

*L* has changed.

No return value.

*S* has NOT changed.

Returns a NEW string.
WMSCI 2005

Rooter: A Methodology for the Typical Unification of Access Points and Redundancy

Jeremy Stribling, Daniel Aguayo and Maxwell Krohn

Randomly-generated submission
accepted to WMSCI 2005

http://pdos.csail.mit.edu/scigen/

No end to the WMSCI emails…
The Price Is Right!

Goal: Buy from a set of 5 items (as many of each as you want) while spending between $9.25 and $10.00.
Demo
Step 1: Identify Information To Store

- What information does this program need to keep track of?
Step 1: Identify Information To Store

- How much has the user spent?
  float: money
- Which items are available?
  list of strings: items
- How much does each available item cost?
  list of floats: prices
- Which item did the user currently choose?
  int: choice
- How many of the currently chosen item does the user want?
  int: number
Step 2: Break Down Functionality

- What are the things that this program needs to do?
Step 2: Break Down Functionality

- Control the overall game play
- Prompt the user to enter a choice and number of items
- Calculate the total spent
- Figure out when the game is over
- Figure out win or lose
- Print a welcome message
- Remove the purchased item from the lists
def playTry1():
    print "Welcome to the price is right!"
    print "Your goal is to buy 5 or fewer items (any number of each)"
    print "and spend between $9.25 and $10"

    items = ['bleach', 'coke', 'ramen', 'ice cream', 'super ball']
    prices = [1.35, .75, .25, 3.00, 1.75]
    money = 0.0

    while money < 9.25 and len(items) > 0:
        print "You have spent $", money
        printItems( items )
        choice = input( "Which item would you like to buy? " )
        number = input( "How many " + items[choice] + " would you like?" )
        print items[choice], "is $", prices[choice]
        money += prices[choice]*number

        prices = prices[0:choice] + prices[choice+1:]
        items = items[0:choice] + items[choice+1:]

    print "You have spent $", money
    if money >= 9.25 and money <= 10.0:
        print "You win!"
    else:
        print "You lose!"