CS 5 Black Gazette

CS Professor Rescued from Runaway Recursive Program

Claremont (AP): A CS professor at a small college in Southern California was rescued on Monday morning from a recursive function that was stuck in an infinite loop. “We’re very proud of our rescue team,” said a supervisor at the site of the incident. The professor had forgotten to put a base case in his function and got pulled into the program and was spinning at very high speed when we got there. We were called by an anonymous function. Fortunately, our elite rescue unit, code-named “Control C”, got there just in time. We mapped our way through his office, filtering through the immense debris, and were able to reduce the risk to the team by memoizing the exact right process. The professor was recovering comfortably and the prognosis is that he may regain his sanity in the distant future.
True Story!

Caltech
Pasadena City College
True Story!

Caltech
(Pasadena City College)
Huntington Library

Caltech
Pasadena City College
True Story!

Caltech = Pasadena City College
True Story!

Caltech ∈ Pasadena City College
True Story!

Caltech ⊆ Pasadena City College
No class this Thursday
Homework 3

1. Lab this week is on audio processing and special effects!
2. Spell checking (memoized edit distance)
3. Work Break!

miniWordList = ["a", "am", "amp", "ample", "as", "asp", "at", "ate", "sat", "spa", "spam", "tea", "was", "wasp"]

>>> wordBreak("wasp\nxx\nample\nyyteaz\nzz")
Enter your best solution: was amp
Your score was 6
Best solution is [12, ['wasp', 'ample', 'tea']]

Options for different scoring methods & different word lists!
List Comprehensions

```python
>>> list(map(lambda X: X*2, range(10)))
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18]

>>> [X*2 for x in range(10)]
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
```
```python
>>> list(filter(lambda x: x % 2 == 0, range(10)))
[0, 2, 4, 6, 8]

>>> [x for x in range(10) if x % 2 == 0]
[0, 2, 4, 6, 8]

>>> list(map(lambda x: x**2, filter(lambda y: y%2 == 0, range(10))))
[0, 4, 16, 36, 64]

>>> [x**2 for x in range(10) if x%2 == 0]
[0, 4, 16, 36, 64]
```

What does this do? Try writing a list comprehension equivalent!
You waited until NOW to tell us about this!?
How do we know that memo-i-zed LCS so fast?

```python
def fastLCS(S1, S2, memo):
    if (S1, S2) in memo: return memo[(S1, S2)]
    elif S1 == "" or S2 == "": return 0
    elif S1[0] == S2[0]:
        result = 1 + fastLCS(S1[1:], S2[1:], memo)
        memo[(S1, S2)] = result
        return result
    else:
        option1= fastLCS(S1, S2[1:], memo)
        option2= fastLCS(S1[1:], S2, memo)
        result = max(option1, option2)
        memo[(S1, S2)] = result
        return result
```

Almost everything here is fast. All but…
### Polynomial vs. Exponential

$10^9$ operations/sec

<table>
<thead>
<tr>
<th></th>
<th>$n = 100$</th>
<th>$n = 500$</th>
<th>$n = 1000$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speed</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$n^2$</td>
<td>$10^{-5}$ sec</td>
<td>$0.00025$ sec</td>
<td>$0.001$ sec</td>
</tr>
<tr>
<td>$n^3$</td>
<td>$0.001$ sec</td>
<td>$0.125$ sec</td>
<td>$1.0$ sec</td>
</tr>
<tr>
<td>$2^n$</td>
<td>$4 \times 10^{13}$ YEARS</td>
<td>$10^{34}$ YEARS</td>
<td>$3.3 \times 10^{284}$ YEARS</td>
</tr>
</tbody>
</table>
Insertion Sorting

```python
>>> sort([42, 57, 1, 3])
[1, 3, 42, 57]
```

```python
The idea... Given a list like L = [42, 57, 1, 3]
    • Recursively sort [57, 1, 3] \rightarrow [1, 3, 57]
    • Insert 42 into [1, 3, 57] \rightarrow [1, 3, 42, 57]

```python
def insert(x, sortedList):
    ''' Takes a number and sorted list as input and returns a new list that has x
       inserted into the right place in the sorted list '''
    if sortedList == []: return [x]
    elif x < sortedList[0]: return [x] + sortedList
    else: return [[sortedList[0]] + \\ 
                   insert(x, sortedList[1:])]  

def sort(myList):
    if myList == []: return []
    else: return insert(myList[0], sort(myList[1:]))
```
What’s the running time?
Mergesort

Can we sort even faster?

\[ \text{msort([42, 3, 1, 5, 27, 8, 2, 7])} \]

Yup! (The secret to all happiness is recursion!)
Mergesort

\[ \text{msort}([42, 3, 1, 5, 27, 8, 2, 7]) \]
Mergesort

\[ \text{msort([42, 3, 1, 5, 27, 8, 2, 7])} \]

\[ \text{msort([42, 3, 1, 5])} / \text{msort([27, 8, 2, 7])} \]

“the magic of recursion!”
Mergesort

\[
\text{msort}([42, 3, 1, 5, 27, 8, 2, 7])
\]

\[
\text{msort}([42, 3, 1, 5]) / \text{msort}([27, 8, 2, 7])
\]

"the magic of recursion!"

\[
[1, 3, 5, 42] \quad [2, 7, 8, 27]
\]
Mergesort

\[
\text{msort([42, 3, 1, 5, 27, 8, 2, 7])}
\]

\[
\text{msort([42, 3, 1, 5])} \quad \text{msort([27, 8, 2, 7])}
\]

“the magic of recursion!”

\[
[1, 3, 5, 42] \quad [2, 7, 8, 27]
\]

\[
\text{merge([1, 3, 5, 42], [2, 7, 8, 27])}
\]
Mergesort

\[ msort([42, 3, 1, 5, 27, 8, 2, 7]) \]

\[ msort([42, 3, 1, 5]) \] \[ msort([27, 8, 2, 7]) \]

\[ merge([1, 3, 5, 42], [2, 7, 8, 27]) \]

[ ]
Mergesort

\[ \text{msort}([42, 3, 1, 5, 27, 8, 2, 7]) \]

\[ \text{msort}([42, 3, 1, 5]) \quad \text{msort}([27, 8, 2, 7]) \]

\[ \text{merge}([1, 3, 5, 42], [2, 7, 8, 27]) \]

[1,
Mergesort

\[\text{msort}([42, 3, 1, 5, 27, 8, 2, 7])\]

\[\text{msort}([42, 3, 1, 5]) \quad \text{msort}([27, 8, 2, 7])\]

\[\text{merge}([1, 3, 5, 42], [2, 7, 8, 27])\]

[1, 2]
Mergesort

\[ \text{msort([42, 3, 1, 5, 27, 8, 2, 7])} \]

\[ \text{msort([42, 3, 1, 5])} \quad \text{msort([27, 8, 2, 7])} \]

\[ \text{merge([1, 3, 5, 42], [2, 7, 8, 27])} \]

[1, 2, 3]
Mergesort

\[
\text{msort}([42, 3, 1, 5, 27, 8, 2, 7])
\]

\[
\text{msort}([42, 3, 1, 5]) \quad \text{msort}([27, 8, 2, 7])
\]

\[
\text{merge}([1, 3, 5, 42], [2, 7, 8, 27])
\]

[1, 2, 3, 5]
Mergesort

\[ \text{msort}([42, 3, 1, 5, 27, 8, 2, 7]) \]

\[ \text{msort}([42, 3, 1, 5]) \quad \text{msort}([27, 8, 2, 7]) \]

\[ \text{merge}([1, 3, 5, 42], [2, 7, 8, 27]) \]

\[ [1, 2, 3, 5, 7] \]
Mergesort

\[
\text{msort}([42, 3, 1, 5, 27, 8, 2, 7])
\]

\[
\text{msort}([42, 3, 1, 5]) \quad \text{msort}([27, 8, 2, 7])
\]

\[
\text{merge}([1, 3, 5, 42], [2, 7, 8, 27])
\]

\[
[1, 2, 3, 5, 7, 8]
\]
Mergesort

\[ \text{msort}([42, 3, 1, 5, 27, 8, 2, 7]) \]

\[ \text{msort}([42, 3, 1, 5]) \quad \text{msort}([27, 8, 2, 7]) \]

\[ \text{merge}([1, 3, 5, 42], [2, 7, 8, 27]) \]

\[ [1, 2, 3, 5, 7, 8, 27] \]
Mergesort

\[
\text{msort([42, 3, 1, 5, 27, 8, 2, 7])}
\]

\[
\text{msort([42, 3, 1, 5])} \quad \text{msort([27, 8, 2, 7])}
\]

\[
\text{merge([1, 3, 5, 42], [2, 7, 8, 27])}
\]

\[
[1, 2, 3, 5, 7, 8, 27, 42] \quad \text{Done!}
\]
msort([42, 3, 1, 6, 5, 2, 7])

msort([42, 3, 1])  msort([6, 5, 2, 7])
\texttt{msort([42, 3, 1, 6, 5, 2, 7])}

\texttt{msort([42, 3, 1])}

\texttt{msort([6, 5, 2, 7])}

\texttt{msort([42])}

\texttt{msort([3, 1])}

\texttt{msort([6, 5])}

\texttt{msort([2, 7])}
msort([42, 3, 1, 6, 5, 2, 7])

msort([42, 3, 1])

msort([6, 5])

msort([2, 7])

msort([42])

msort([3])

msort([1])
msort([42, 3, 1, 6, 5, 2, 7])

msort([42, 3, 1])

[42]

msort([42])

[42]

msort([3, 1])

[3]

msort([3])

[3]

msort([6, 5, 2, 7])

msort([6, 5])

[6, 5]

msort([6])

[6]

msort([2, 7])

[2, 7]
msort([42, 3, 1, 6, 5, 2, 7])

msort([42, 3, 1])
  msort([42])
  msort([3, 1])
    msort([3]) msort([1])

msort([6, 5, 2, 7])
  msort([6, 5])
  msort([2, 7])
msort([42, 3, 1, 6, 5, 2, 7])

<table>
<thead>
<tr>
<th>msort([42, 3, 1])</th>
<th>msort([6, 5, 2, 7])</th>
</tr>
</thead>
<tbody>
<tr>
<td>msort([42])</td>
<td>msort([6, 5])</td>
</tr>
<tr>
<td>msort([3])</td>
<td>msort([2, 7])</td>
</tr>
<tr>
<td>msort([1])</td>
<td></td>
</tr>
</tbody>
</table>

[1, 3, 42]
msort([42, 3, 1, 6, 5, 2, 7])

msort([42, 3, 1])

msort([42])

msort([3, 1])

msort([6, 5, 2, 7])

msort([6, 5])

msort([2, 7])
def mergesort(myList):
    '''mergesort the given list'''
    if len(myList) <= 1: return myList
    else:
        midpoint = len(myList)//2  # Why //

def merge(sortedList1, sortedList2):
    '''merge two sorted arrays'''
    if sortedList1 == []: return sortedList2
    elif sortedList2 == []: return sortedList1
    elif sortedList1[0] < sortedList2[0]:
        return [sortedList1[0]] + merge(sortedList1[1:], sortedList2)
    else:
        return [sortedList2[0]] + merge(sortedList1, sortedList2[1:])
def mergesort(myList):
    ''' mergesort the given list '''
    if len(myList) <= 1: return myList
    else:
        midpoint = len(myList)//2  # Why //
        return merge(mergesort(myList[0:midpoint]),
                      mergesort(myList[midpoint:]))

def merge(sortedList1, sortedList2):
    ''' merge two sorted arrays '''
    if sortedList1 == []: return sortedList2
    elif sortedList2 == []: return sortedList1
    elif sortedList1[0] < sortedList2[0]:
        return [sortedList1[0]] + merge(sortedList1[1:], sortedList2)
    else:
        return [sortedList2[0]] + merge(sortedList1, sortedList2[1:])
What’s the running time of mergesort?

To keep things tidy, let’s assume that the length of the array is a power of 2...
```python
import time
import random
import sys
import matplotlib.pyplot as plt

sys.setrecursionlimit(30000)

def run(algorithm, length, trials):
    ''' Takes a function name, array length, and number trials and reports average time for algorithm to sort arrays of that length. '''
    print("Starting ", algorithm, "array length ", length, "for ", trials, " trials")
    L = list(range(length))  # Get a list of the right length
    totalTime = 0
    for iteration in range(trials):
        random.shuffle(L)  # Permute list at random
        start = time.time()  # Start timer
        algorithm(L)  # Run sorting algorithm on list
        end = time.time()  # End timer
        totalTime += end-start  # Update total time
    averageTime = totalTime/trials
    print("Average run time: ",averageTime)
    return averageTime

def test(start, end, step, trials):
    itime = []  # insert sort times
    mtime = []  # mergesort times
    for length in range(start, end, step):
        itime = itime + [run(insertionsort, length, trials)]
        mtime = mtime + [run(mergesort, length, trials)]
    plt.plot(list(range(start, end, step)), mtime, "ro")
    plt.plot(list(range(start, end, step)), itime, "bx")
    plt.show()

    test(100, 1000, 100, 10)
```
“Hard” Problems

Snowplows of Northern Minnesota (aka the “Vertex Cover Problem”)
“Hard” Problems

Snowplows of Northern Minnesota (aka the “Vertex Cover Problem”)

Brute-force?  Greed?  DP?
# n² and n³ versus 2ⁿ

The Ran-O-Matic performs $10^9$ operations/sec

<table>
<thead>
<tr>
<th></th>
<th>n = 10</th>
<th>n = 30</th>
<th>n = 50</th>
<th>n = 70</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n^2$</td>
<td>100</td>
<td>900</td>
<td>2500</td>
<td>4900</td>
</tr>
<tr>
<td></td>
<td>&lt; 1 sec</td>
<td>&lt; 1 sec</td>
<td>&lt; 1 sec</td>
<td>&lt; 1 sec</td>
</tr>
<tr>
<td>$n^3$</td>
<td>1000</td>
<td>27000</td>
<td>125K</td>
<td>343K</td>
</tr>
<tr>
<td></td>
<td>&lt; 1 sec</td>
<td>&lt; 1 sec</td>
<td>&lt; 1 sec</td>
<td>&lt; 1 sec</td>
</tr>
<tr>
<td>$2^n$</td>
<td>1024</td>
<td>$10^9$</td>
<td>13 days</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt; 1 sec</td>
<td>1 sec</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
n² and n³ versus 2ⁿ

The Ran-O-Matic performs $10^9$ operations/sec

<table>
<thead>
<tr>
<th></th>
<th>n = 10</th>
<th>n = 30</th>
<th>n = 50</th>
<th>n = 70</th>
</tr>
</thead>
<tbody>
<tr>
<td>n²</td>
<td>100</td>
<td>900</td>
<td>2500</td>
<td>4900</td>
</tr>
<tr>
<td></td>
<td>&lt; 1 sec</td>
<td>&lt; 1 sec</td>
<td>&lt; 1 sec</td>
<td>&lt; 1 sec</td>
</tr>
<tr>
<td>n³</td>
<td>1000</td>
<td>27000</td>
<td>125K</td>
<td>343K</td>
</tr>
<tr>
<td></td>
<td>&lt; 1 sec</td>
<td>&lt; 1 sec</td>
<td>&lt; 1 sec</td>
<td>&lt; 1 sec</td>
</tr>
<tr>
<td>2ⁿ</td>
<td>1024</td>
<td>$10^9$</td>
<td>13 days</td>
<td>37 thousand years</td>
</tr>
<tr>
<td></td>
<td>&lt; 1 sec</td>
<td>1 sec</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Assuming computers double in speed every year, let’s just wait 10 years!
### $n^2$ and $n^3$ versus $2^n$

The Ran-O-Matic performs $10^9$ operations/sec

<table>
<thead>
<tr>
<th></th>
<th>$n = 10$</th>
<th>$n = 30$</th>
<th>$n = 50$</th>
<th>$n = 70$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n^2$</td>
<td>100</td>
<td>900</td>
<td>2500</td>
<td>4900</td>
</tr>
<tr>
<td></td>
<td>&lt; 1 sec</td>
<td>&lt; 1 sec</td>
<td>&lt; 1 sec</td>
<td>&lt; 1 sec</td>
</tr>
<tr>
<td>$n^3$</td>
<td>1000</td>
<td>27000</td>
<td>125K</td>
<td>343K</td>
</tr>
<tr>
<td></td>
<td>&lt; 1 sec</td>
<td>&lt; 1 sec</td>
<td>&lt; 1 sec</td>
<td>&lt; 1 sec</td>
</tr>
<tr>
<td>$2^n$</td>
<td>1024</td>
<td>$10^9$</td>
<td>13 days</td>
<td>37 thousand years</td>
</tr>
<tr>
<td></td>
<td>&lt; 1 sec</td>
<td>1 sec</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

37 thousand years -> 37 years
The Traveling Salesperson Problem

Brute Force?  Greed?  DP?
“Hard” Problems

The Travelling Salesperson Problem

![Graph showing the travelling salesperson problem with cities Claremont, Montclare, Montclear, and Clearmont connected by edges with distances.]
“Hard” Problems

The Travelling Salesperson Problem

Claremont

Montclare

Montclear

10^{42}

2

2

1

1

Clearmont
The Hamiltonian Cycle Problem
Here as he walked by
on the 16th of October 1843
Sir William Rowan Hamilton
in a flash of genius discovered
the fundamental formula for
quaternion multiplication
\[ i^2 = j^2 = k^2 = ijk = -1 \]
& cut it on a stone of this bridge
Snowplows and Travelling Salesperson Revisited!

Tens of thousands of other known problems go in this cloud!!

- Snowplow Problem
- Travelling Salesperson Problem
- Protein Folding
- ... (other problems)
- Sodoku

NP-hard problems
I can’t find an efficient algorithm, I guess I’m just too dumb.

I can’t find an efficient algorithm, because no such algorithm is possible!

I can’t find an efficient algorithm, but neither can all these famous people!