Wellington (AP): Two HMC penguins were missing after their spaceship lost power and crashed into the southern ocean. “The CS penguin had kindly offered a ride to her friend,” blubbered a distraught professor, “and apparently he was fiddling with the flight computer just before takeoff. I don’t know what I’ll do for classroom examples now.”

With weather worsening, there is little hope for rescue. A memorial service will be held Sunday in the Hoch-Shanahan freezer.
Mergesort

```plaintext
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])
```

[...

Let’s try it out - and let’s not even make n a power of 2!

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

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

How “Efficient” Is Mergesort?

How big a deal is this?

The Meder-O-Matic Supercomputer:
100 billion steps/second

\[
n^2 \text{ algorithm} \quad n \log_2 n \text{ algorithm}
\]

\[
n = 10^8 \quad 11.5+ \text{ days}
\]
This Week

Homework 3:
• Reading
• Black lab (Problem 1) is same as Gold this week
• Problem 2: Spel Chekking
• Problem 3: Word Break game

Aside: Another Way to map

```python
def doubleList1(L):
    return list(map(lambda x: 2*x, L))
def doubleList2(L):
    return [2*x for x in L]
def doubleListFiltered1(L):
    return list(map(lambda x: 2*x, filter(lambda x: x != 42, L)))
def doubleListFiltered2(L):
    return [2*x for x in L if x != 42]
```

Conditionalizing lambda

```python
>>> list(map(lambda x: "nice" if x == 42 else "blech!", [42, 7, 6, 42, 3]))
["nice", "blech!", "blech!", "nice", "blech!"]

>>> list(map(lambda x: "HM" if x == 42
else "PO" if x == 47
else x,
    [42, 7, 6, 47, 3]))
['HM', 7, 6, 'PO', 3]
```

How Efficient is LCS?

```python
def LCS(S1, S2):
    if S1 == "" or S2 == "": return 0
    else:
        if S1[0] == S2[0]:
            return 1+LCS(S1[1:], S2[1:]))
        else:
            return max(LCS(S1, S2[1:]), LCS(S1[1:], S2))
```

Demo LCS, fastLCS

Two strings of length 100 nucleotides each...

```python
>>> steps = 2**100
>>> speed = 3 * 10**9
>>> seconds = steps / speed
>>> years = seconds / (60*60*24*365.25)
```

13 trillion years!
How Dictionaries Work: Hashing

```python
>>> D = {'Julie': 'chocolate', ... }
>>> x = (1, 2)
>>> D[x] = 'my tuple'
>>> D['Julie': 'chocolate', (1, 2): 'my tuple']
```

Memory locations

```python
D['Julie'] — 5999
D[x] =
D[(1, 2)]
3
5999 'chocolate'
```

Imagine that we now changed `x[0]=42`

```python
def LCS(S1, S2):
    if S1 == '' or S2 == '': return 0
    elif S1[0] == S2[0]:
        return 1+LCS(S1[1:], S2[1:])
    else:
        return max(LCS(S1, S2[1:]), LCS(S1[1:], S2))
```

Old slow version

```python
def fastLCS(S1, S2):
    if (S1, S2) in memo: return memo[(S1, S2)]
    elif S1 == '' or S2 == '':
        answer = 0
    elif S1[0] == S2[0]:
        answer = 1+LCS(S1[1:], S2[1:])
    else:
        answer = max(LCS(S1, S2[1:]), LCS(S1[1:], S2))
    memo[(S1, S2)] = answer
    return answer
```

New fast "memoized" version

Dictionaries Revisited

```python
>>> D = {'spam': 'yummy!', (42, 42): 'an important point'}
>>> D['spam']
'yummy!'
>>> D[(42, 42)]
'an important point'
>>> D['zaster!', 'putrid', 'smoke!']
BARF!
```
**Beyond LCS: Edit Distance**

```python
>>> ED("ATTATCG", "ACATTC")
4
ATTAT-CG
A-CATTC-

The lower the edit distance the better!
```

```python
def ED(S1, S2):
    if S1 == '': return len(S2)
    elif S2 == '': return len(S1)
    elif S1[0] == S2[0]: return ED(S1[1:], S2[1:])
    else:  # substitute, insert, or delete!
        return 1 + min(ED(S1, S2[1:]), ED(S1[1:], S2))
```

**Beyond LCS: Change**

```python
def change(value, coins):
    if value <= 0:
        return 0
    elif coins == []:
        return float("inf")
    loseIt = change(value, coins[1:])
    if value < coins[0]:
        return loseIt
    else:
        useIt = 1 + change(value - coins[0], coins)
        return min(useIt, loseIt)
```

```python
>>> change(5, [1, 2, 5, 6])
3
```

```python
>>> change(7, [1, 2, 5, 6])
4
```

```python
>>> change(0, [1, 2, 5, 6])
0
```

```python
>>> change(1, [1, 2, 5, 6])
1
```