
DEAR FUTURE SELF,
#
YOU'RE LOOKING AT THIS FILE BECAUSE
THE PARSE FUNCTION FINALLY BROKE.
#
IT'S NOT FIXABLE. YOU HAVE TO REWRITE IT.
SINCERELY, PAST SELF
DEAR PAST SELF, IT'S KINDA CREEPY HOW YOU DO THAT.
#ALSO, IT'S PROBABLY AT LEAST
2013. DID YOU EVER TAKE
#THAT TRIP TO ICELAND?
STOP JUDGING ME!

My office hours have changed!

In-person (no zoom)

Jacob’s Courtyard (tent)

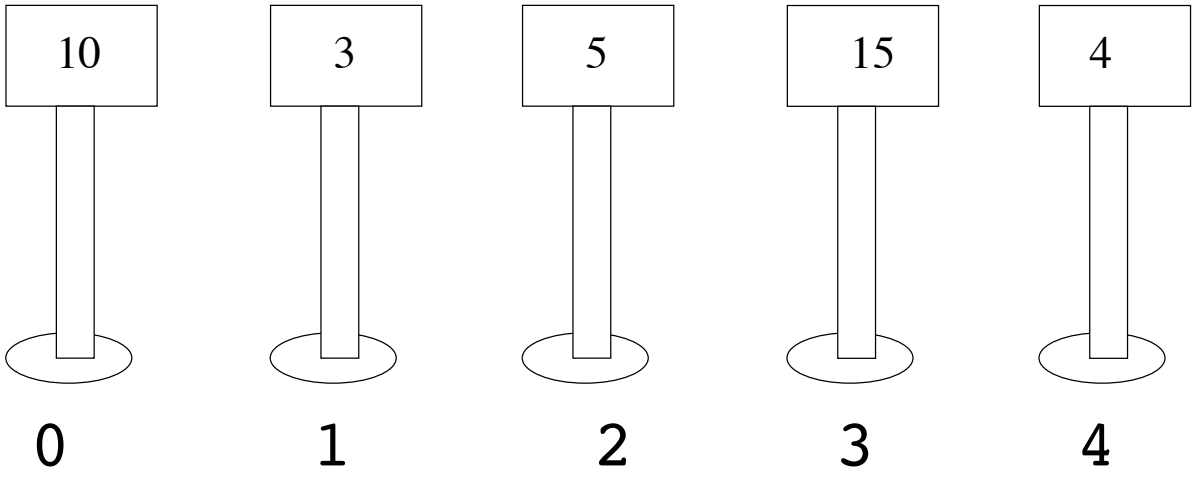
Fridays 3-4pm (same as before)

CS 5 Green

Learning Goals

- Review use-it-or-lose-it
- Explain mutability
- Describe problems with recursion
- Use dictionaries
- Explain how memoization makes recursion faster

The Game of Pegs…



Choose a set of pegs of maximum total value, without choosing two adjacent pegs. Return the max value

```
>>> pegs([10, 3, 5, 15, 4])
???
```

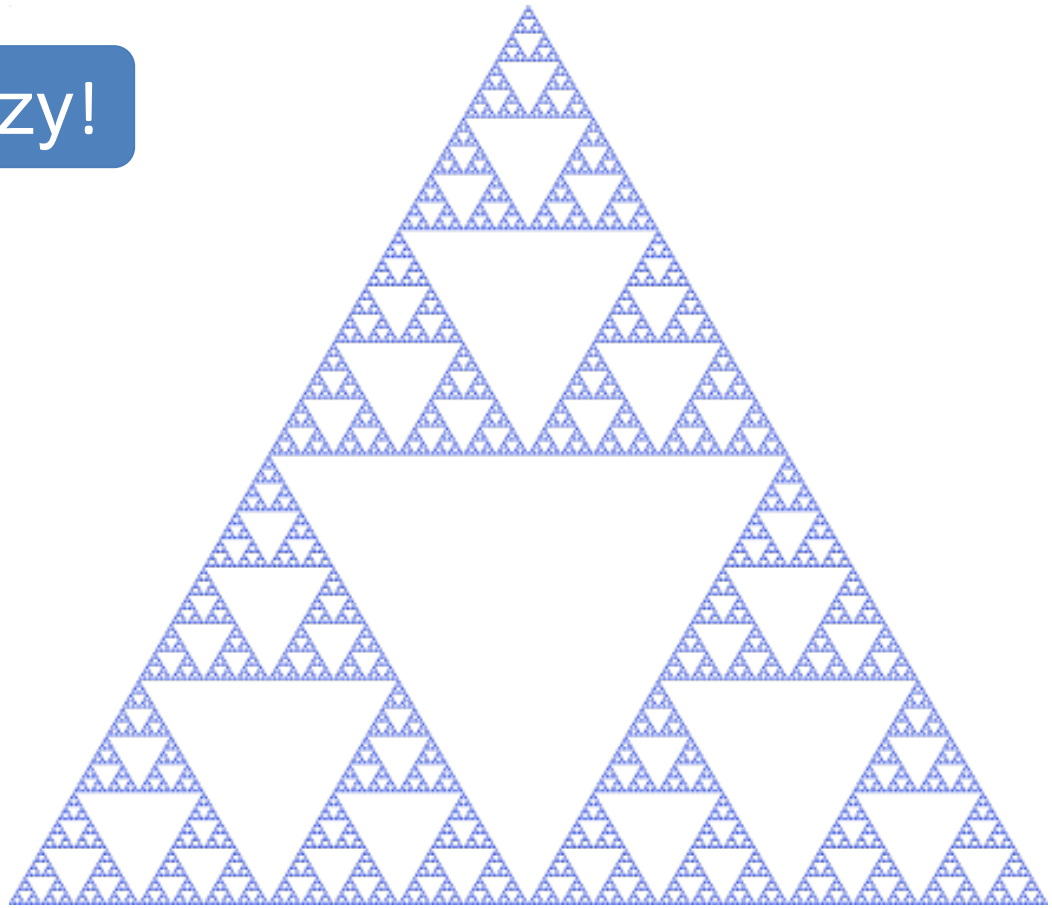
The Game of Pegs

Worksheet

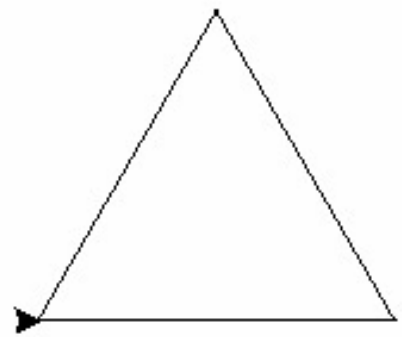

```
def pegs(pegList):
    """Accepts a list of pegs as input. Finds and
    returns the maximum value obtainable without
    using adjacent pegs."""
    if pegList == []: return _____
    else:
        it = pegList[0]

        useIt  = _____

        loseIt = _____

        return _____
```

Courtesy of Allie Russell

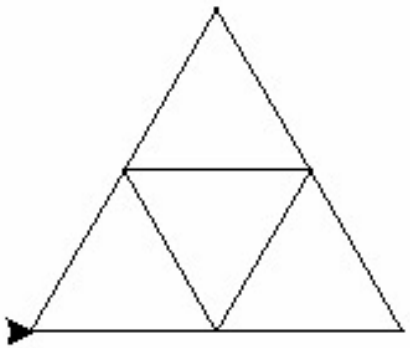
The Sierpinski Triangle!



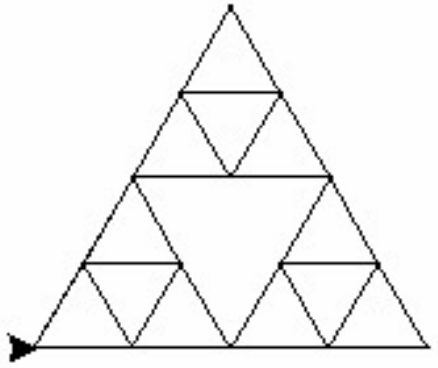
```
>>> fractal(200, 0)
```



```
>>> fractal(200, 1)
```

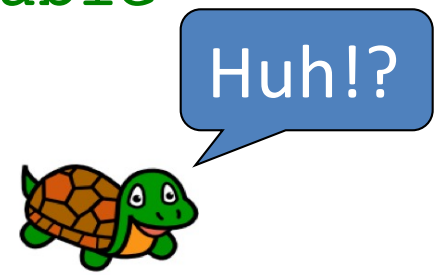


```
>>> fractal(200, 2)
```



An aside on “mutability”

```
>>> L = [29, 47, 17, 23]
>>> L
[29, 47, 17, 23]
>>> L[1] = 42 # mutate the list at index 1
>>> L
[29, 42, 17, 23] # lists are mutable

>>> S = "spam"
>>> S[1] = "c" # strings are immutable
TypeError: 'str' object does not
support item assignment
```



An aside on “mutability”

```
>>> L = [29, 47, 17, 23]
>>> L.append(42)
>>> L
[29, 47, 17, 23, 42]

>>> S = "spam"
>>> S.append("q")
BARF!

>>> S = S + "q"
>>> S
"spamq"
```

+ (add operator)

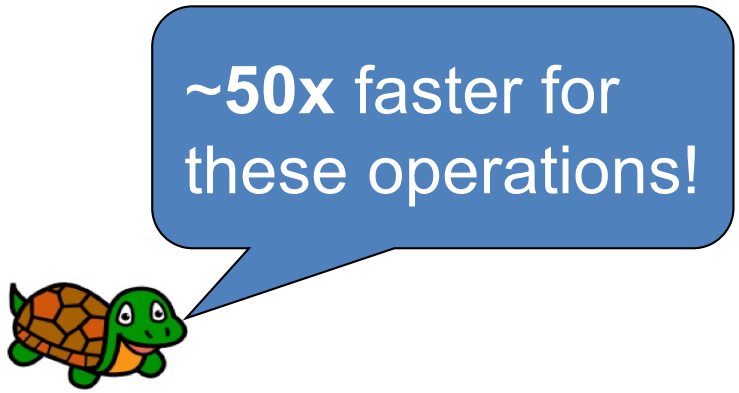
```
>>> L = [6, 3]
>>> L
[6, 3]
>>> L + [9,11]
[6, 3, 9, 11]
>>> L
[6, 3]
```

```
>>> L = [6,3]
>>> L
[6, 3]
>>> L = L + [9, 11]
>>> L
[6, 3, 9, 11]
```

add returns a new list! We must assign it!

Why append and extend?

```
>>> L = list(range(10**8))
>>> L = L + [42] # 0.8s

>>> L = list(range10**8))
>>> L.append(42) # 0.02s

>>> L = L + [5, 6, 7, 8] # 1.2s
>>> L.extend([5, 6, 7, 8]) # 0.03s
```



append

```
>>> L = [6, 3]
>>> L
[6, 3]
>>> L.append([9,11])
>>> L
[6, 3, [9, 11]]
```

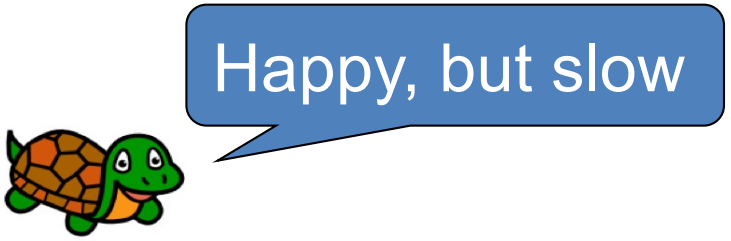
extend

```
>>> L = [6,3]
>>> L
[6, 3]
>>> L.extend([9,11])
>>> L
[6, 3, 9, 11]
```

Nothing is returned! L is **modified** instead!

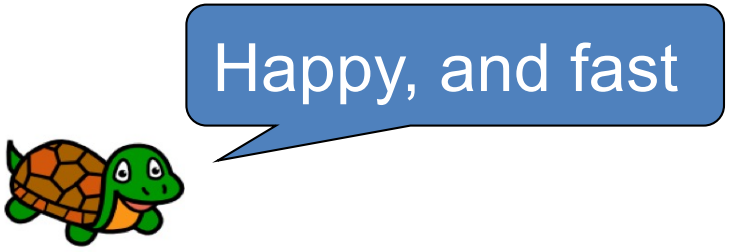
A subtle point…

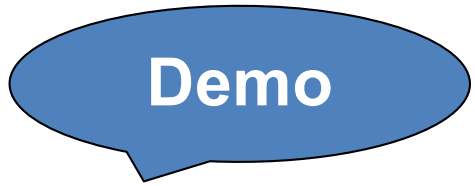
```
def initely():
    L = [1, 2, 3]
    M = [4, 5, 6]
    return L + M
```




```
def unct():
    L = [1, 2, 3]
    M = [4, 5, 6]
    return L.extend(M)
```

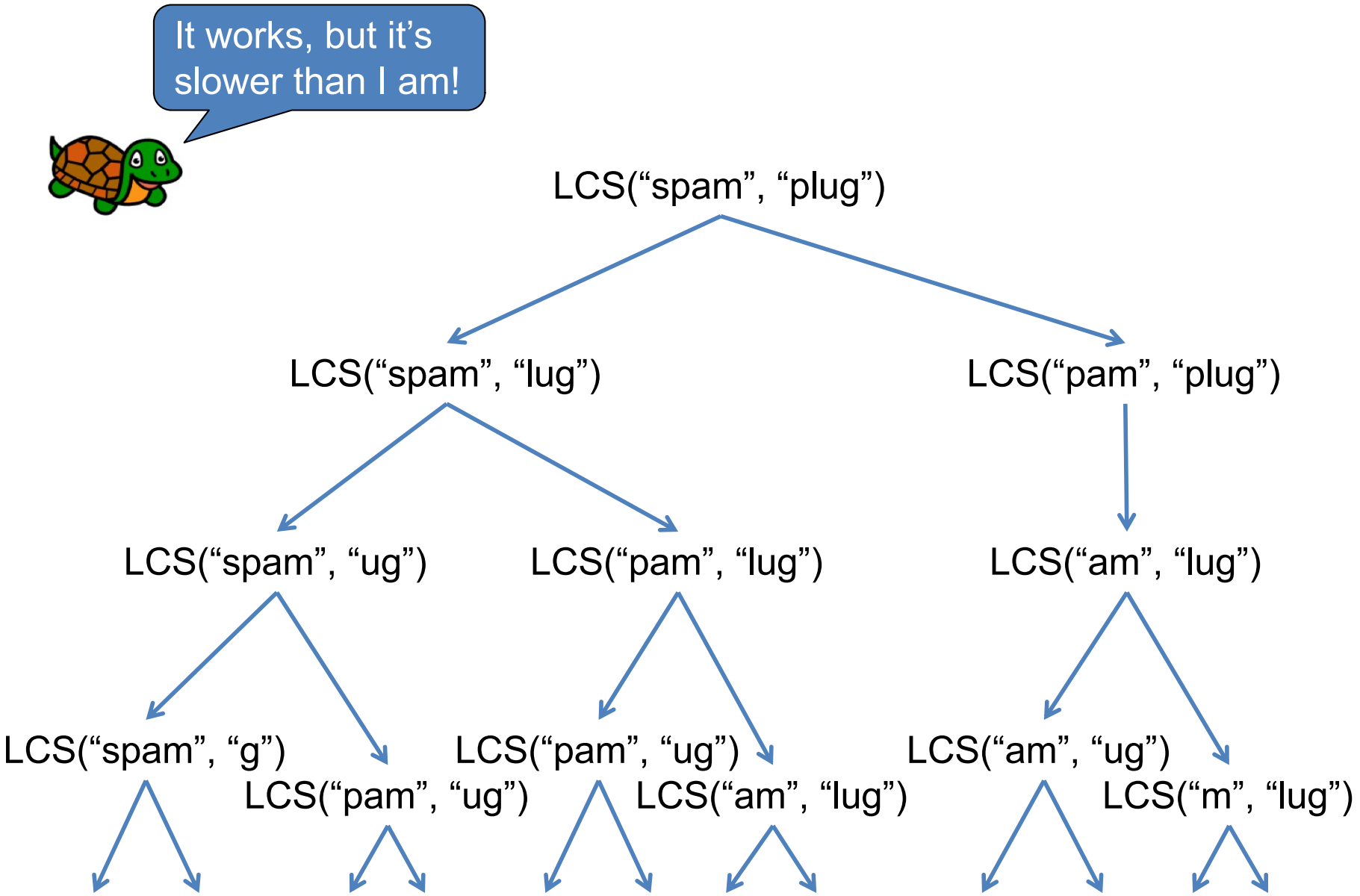

```
def initive():
    L = [1, 2, 3]
    M = [4, 5, 6]
    L.extend(M)
    return L
```





LCS Revisited…

```
def LCS(stringA, stringB):
    """Accepts two strings as input. Returns the length of
    the Longest Common Substring."""
    # base case
    if stringA == "" or stringB == "": return 0

    # add 1 if first characters match; check remaining
    elif stringA[0] == stringB[0]:
        return 1 + LCS(stringA[1:], stringB[1:])

    # try dropping 1 char from each string; return max
    else:
        option1 = LCS(stringA, stringB[1:])
        option2 = LCS(stringA[1:], stringB)
        return max(option1, option2)
```


It works, but it's slower than I am!
LCS("spam", "plug")
LCS("spam", "lug")
LCS("pam", "plug")
LCS("spam", "ug")
LCS("pam", "lug")
LCS("am", "lug")
LCS("spam", "g")
LCS("pam", "ug")
LCS("pam", "ug")
LCS("am", "lug")
LCS("am", "ug")
LCS("m", "lug")

Jumping off paper…

1 sheet of regular paper

Folds (n)	Layers (2^n)	Height
0	1	1 paper thick
1	2	2 sheets thick
… skip a few …		
7	2^7	notebook
13	2^{13}	1 meter
17	2^{17}	2 story house
20	2^{20}	1/4 Sears Tower
30	2^{30}	Outer limit of atmosphere
50	2^{50}	Distance to sun

2^n is really bad!

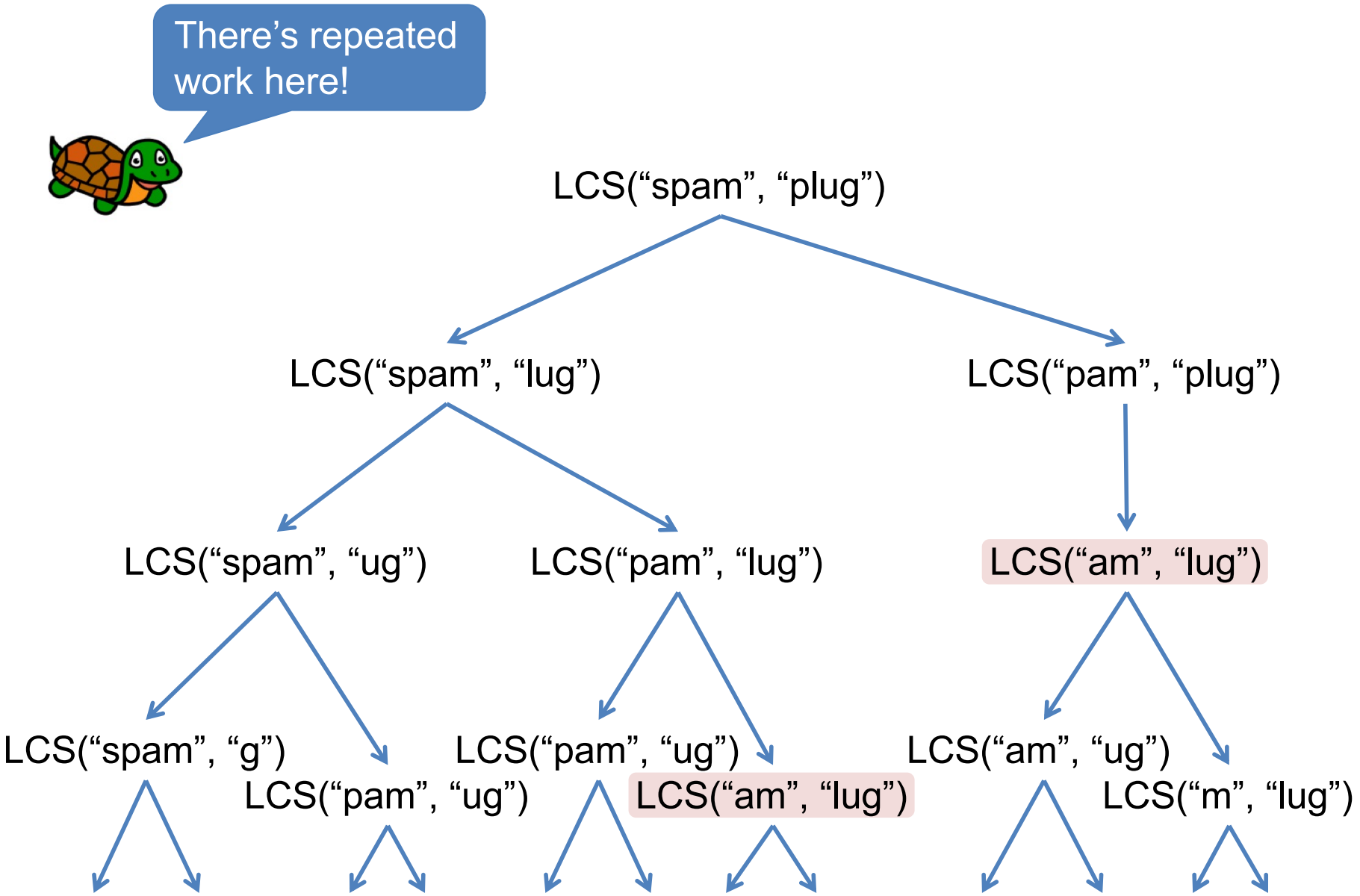
1 billion operations per second!

n = 20: 0.001 seconds

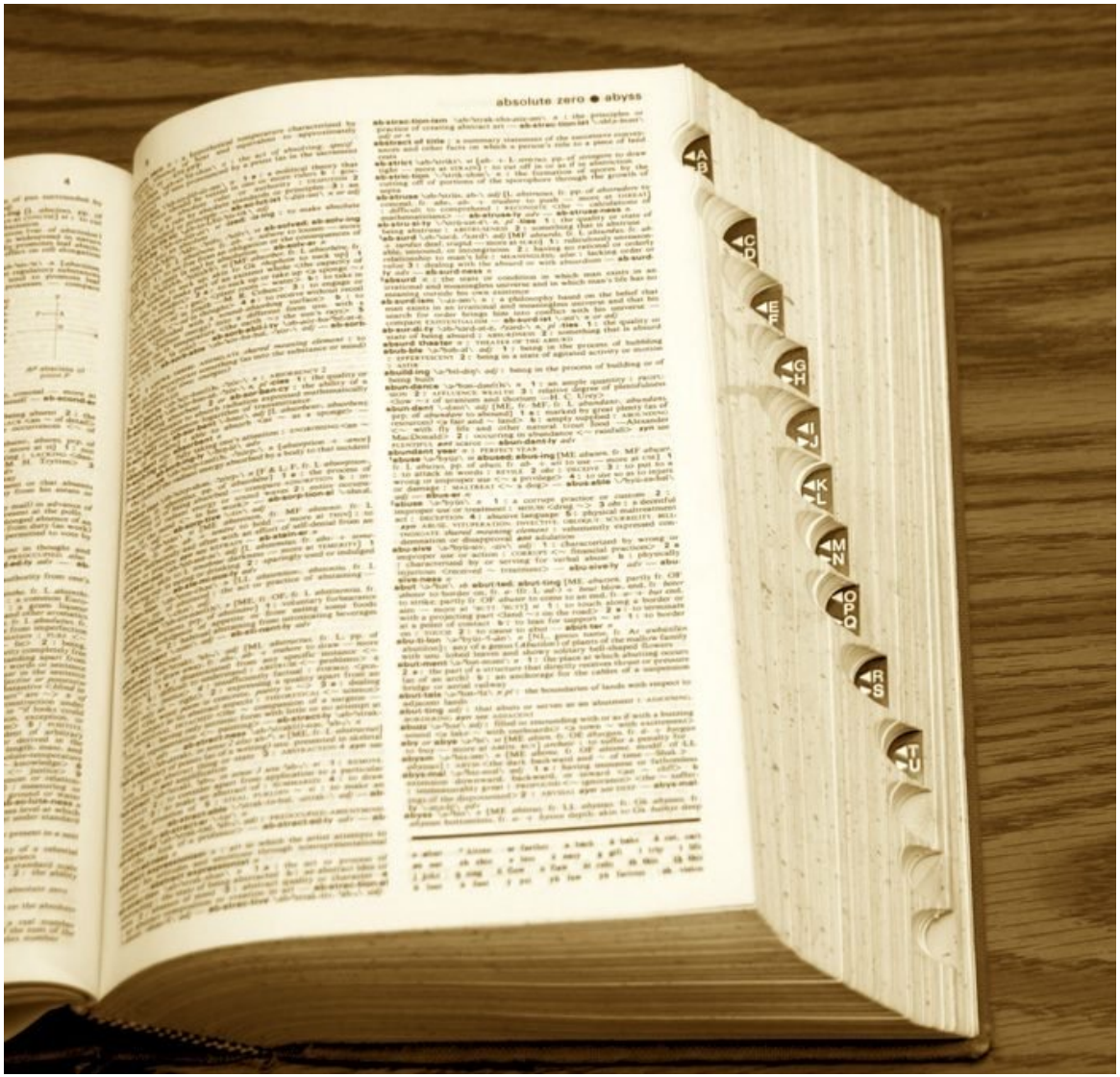
n = 50:

n = 100:

n = 300:


There's repeated work here!
LCS("spam", "plug")
LCS("spam", "lug")
LCS("pam", "plug")
LCS("spam", "ug")
LCS("pam", "lug")
LCS("am", "lug")
LCS("spam", "g")
LCS("pam", "ug")
LCS("pam", "ug")
LCS("am", "lug")
LCS("am", "ug")
LCS("m", "lug")

Dictionaries!



the *key*

Each word in a dictionary has a definition

the *value*

Words are stored in a way that enables easy look-up.

Dictionaries!

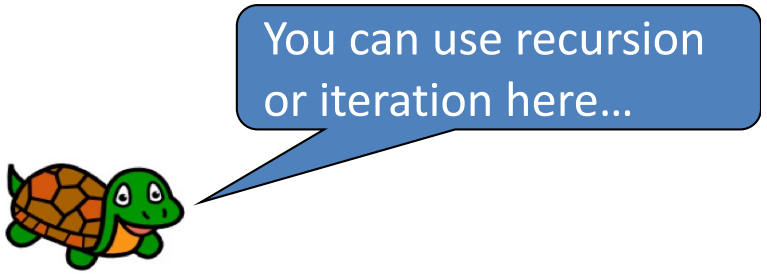
the *key*

the *value*

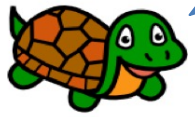
```
>>> D = {}
>>> D["spam"] = "a health food product"
>>> D[42] = "an important number"
>>> D["joe"] = ["jwirth@hmc.edu", "olin 1253"]
>>> D
{'spam': 'a health food product', 42: 'an important number', 'joe': ['jwirth@hmc.edu', 'olin 1253']}
>>> D.keys()
dict_keys(['spam', 42, 'joe']) # can also do D.values()
>>> D[42]
'an important number'
>>> 42 in D
True
>>> 43 in D
False
>>> "bjorn" in D
False
>>> D["bjorn"]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'bjorn'
```

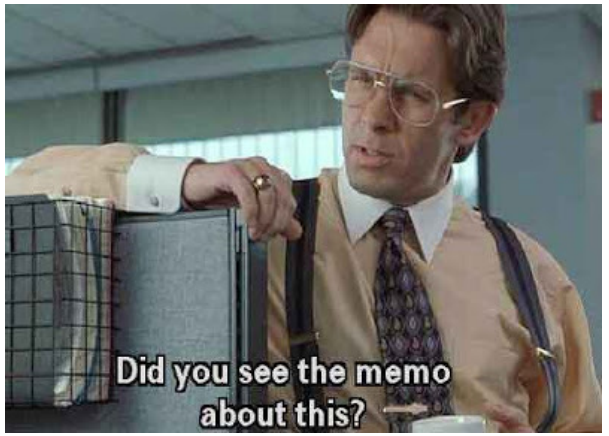
Try This…


```
>>> scrabbleDict = {"a":1, "b":3, "c":3, "d":2, …}
>>> score("spam", scrabbleDict)
8
```




```
def score(myString, myDict):
    """Returns the score of a word in scrabble"""
```


There's repeated work here!


Did you see the memo about this?

LCS("spam", "plug")
LCS("spam", "lug")
LCS("pam", "plug")
LCS("spam", "ug")
LCS("pam", "lug")
LCS("am", "lug")
LCS("spam", "g")
LCS("pam", "ug")
LCS("pam", "ug")
LCS("am", "lug")
LCS("am", "ug")
LCS("m", "lug")

Modify LCS to be fastLCS


```
def fastLCS(stringA, stringB, memoD):
    """Accepts two strings and a dictionary as inputs. Returns
    the length of the Longest Common Substring."""



    # base case: either string empty => no LCS
    if stringA == "" or stringB == "": return 0

    # char match => +1 to count; check remaining strings
    elif stringA[0] == stringB[0]:
        return   1 + fastLCS(stringA[1:], stringB[1:])

    # try removing one char from each string
    else:
        option1 = fastLCS(stringA, stringB[1:])
        option2 = fastLCS(stringA[1:], stringB)
        return   max(option1, option2)
```

Extra Practice!

(implement a recursive sorting algorithm)

Mergesort

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

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

"the magic of recursion!"

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

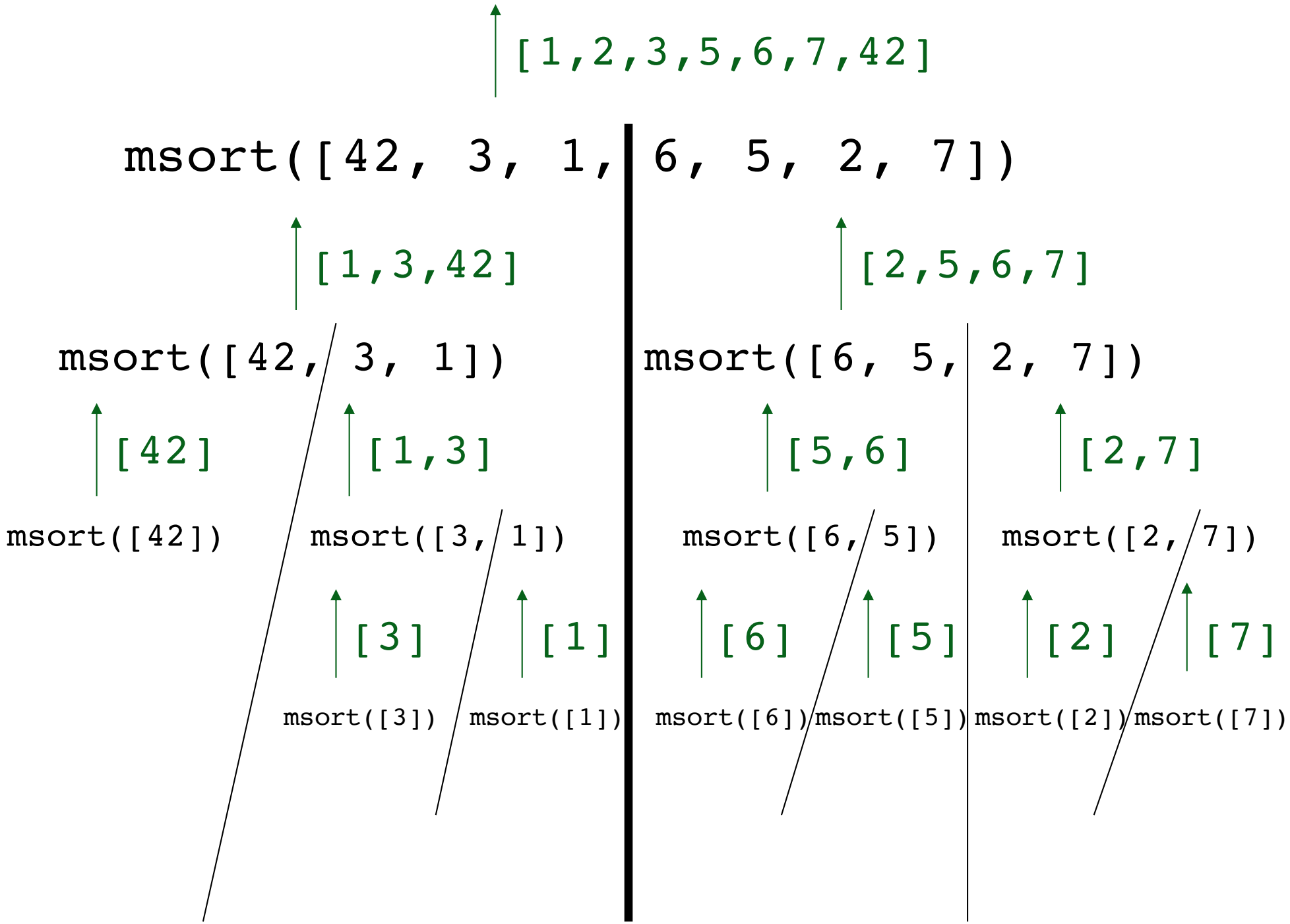
Mergesort

```
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])
```

```
[1] + merge([3, 5, 42], [2, 7, 8, 27])
        [2] + merge([3, 5, 42], [7, 8, 27])
                [3] + merge([5, 42], [7, 8, 27])
                                …
                        [27] + merge([42], [])
                                [42]
```


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


```
def mergesort(my_list):
    """mergesort the given list"""
    if len(my_list) <= 1: return my_list
    else:
```

```
def merge(sorted_list1, sorted_list2):
    """merge two sorted arrays"""
    if sorted_list1 == []: return sorted_list2
    elif sorted_list2 == []: return sorted_list1
    elif sorted_list1[0] < sorted_list2[0]:
        return ???
    else:
        return ???
```

Reminder:

- Lecture feedback form (https://forms.gle/aPmkpXDUTp4Xo4CV7)