Recursion

Announcements

- Lab 9 (sorting) posted; due Saturday at midnight
- Quiz 5 on Friday
  - Study guide posted

Today's plan

- Review Friday's lecture
- Merge sort, *O(n log n)* sort that can be defined using recursion
- Idea of recursion
- What the stack looks like during recursion

Review: insertion sort

- "Insert" `n`th value into position so that the first `n+1` values in the list are in sorted order.

```
def insertionSort(L):
    n = len(L)
    for i in range(1, n):
        position = i
        while position > 0 and L[position-1] > L[position]:
            swap(L, position, position-1)
            position -= 1
```

Review

- Put call to `main()` in protected block so it gets called if the program is run from command line, but not if it's imported:

```
if __name__ == "__main__":
  main()
```

- Put `assert` statements in `main()` to test an algorithm with a variety of inputs

- Use the `time()` function to figure out how much time elapsed during an algorithm's execution

Merge sort

- Observation: if we have two sorted lists, combining them into one sorted list is a linear-time, *O(n)*, algorithm.

- Algorithm: split the list in half, sort each half separately, merge them back together

- Run time: we can split the list in half a logarithmic number of times, each merge is linear, so the overall algorithm is *O(n log n)*

- Note: merge sort is not in place

```
def merge(L1, L2):
    L = []
    index1 = 0
    index2 = 0
    while index1 < len(L1) and index2 < len(L2):
        if L1[index1] < L2[index2]:
            L.append(L1[index1])
            index1 += 1
        else:
            L.append(L2[index2])
            index2 += 1
    if index1 == len(L1):
        L += L2[index2:]
    else:
        L += L1[index1:]
    return L
```

```
def mergeSort(L):
    if len(L) <= 1:
        return L
    else:
        middleIndex = len(L)/2
        firstHalf = mergeSort(L[:middleIndex])
        secondHalf = mergeSort(L[middleIndex:])
        return merge(firstHalf, secondHalf)
```

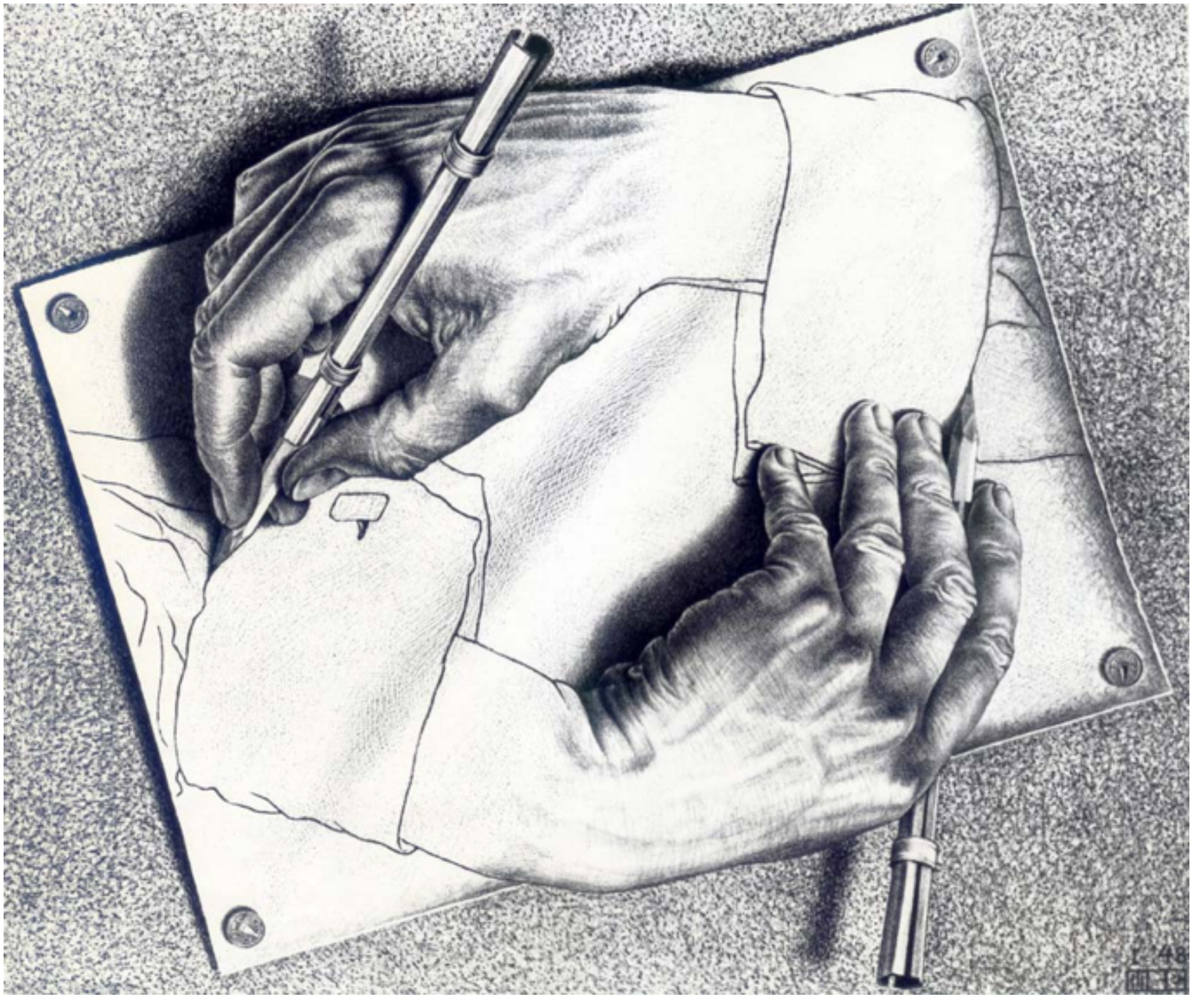
Back to timesorts.py

Algorithm run-times

- *O(1)*: indexing, arithmetic
- *O(log n)*: binary search
- *O(n)*: linear search, merging two lists
- *O(n log n)*: merge sort
- $O(n^2)$: insertion sort, selection sort, bubble sort

Idea of Recursion

- A function that calls itself in its own definition!
  - This doesn't work with definitions for words
  - But it does (miraculously) work for code—we'll see how
  - "To understand recursion, you must understand recursion"




MEMENTO
SOME MEMORIES ARE BEST FORGOTTEN.
GUY PEARCE CARRIE-ANNE MOSS JOE PANTOLIANO
MEMENTO

Idea of Recursion

- A **recursive function** has one (or more) base case and one (or more) general case.

  - The **base case** is a version of the problem that can be solved immediately.

  - The **general case** can be solved by using the answer to a smaller version of the same problem. We're not solving the problem right away, but *we are getting closer to a solution*

  - Like mathematical induction (or falling dominoes)

Another example

```
def sumToNum(n):
    """
    Purpose: Return the sum of the integers from 1 to n
    Paramters: n - a positive integer
    Returns: the sum
    """
    if n == 1:
        return 1
    else:
        return n + sumToNum(n-1)
```

How does this work?

```
def sumToNum(n):
    if n == 1:
        return 1
    else:
        return n + sumToNum(n-1)
```

```
sumToNum(4)
4 + sumToNum(3)
4 + 3 + sumToNum(2)
4 + 3 + 2 + sumToNum(1)
4 + 3 + 2 + 1
10
```

How does this work?

- Essentially we are working with multiple “copies” of the same function.

- We have seen how a function can be called more than once with different parameters

- We have seen that you can call a function which itself calls a function

- Recursion puts these two ideas together. Let’s see how it plays out on the stack…

More than once with different parameters

```
def add5(n):
    return n + 5

def main():
    a = 7
    print(add5(a))
    b = 6
    print(add5(b))

main()
```

Function that calls a function

```
def add5(n):
    return n + 5

def add5List(L):
    for i in range(len(L)):
        L[i] = add5(L[i])

def main():
    L = [1, 2, 3]
    add5List(L)
    print(L)

main()
```

Recursive Function

Draw stack when base case is reached:

```
def sumToNum(n):
    if n == 1:
        return 1
    else:
        return n + sumToNum(n-1)

def main():
    result = sumToNum(4)
    print(result)
```