Midterm 2

- Friday, April 17, 11:00
- Contact me if you need a different time
- Administered as Canvas quiz with live Zoom meeting
- Topics:
  - Sorting, searching, efficient algorithms
  - Binary, hex
  - Complex lists (lists of lists of lists)
  - Tuples, and dictionaries
A recursive process (formula, algorithm, function) is defined in terms of smaller instances of the same process.

In Python, a recursive function is a function that calls itself.
A recursive process (formula, algorithm, function) is defined in terms of smaller instances of the same process.

In Python, a recursive function is a function that calls itself.
Recursive definition of n!

If $n = 0$, then $n! = 1$

otherwise, $n! = n \times (n-1)!$
Recursive definition of n!

If n = 0, then n! = 1
otherwise, n! = n \times (n-1)!

def factorial(n):
    if n == 0:
        return 1
    else:
        return n*factorial(n-1)
Binary search

To search for $x$ in a sorted list......

If $x$ is equal to the middle element, return the index of this element

If $x$ is less than the middle element, search for $x$ in the left half

If $x$ is greater than the middle element, search for $x$ in the right half
This gives simple recursive implementation of binary search

```python
def rec_binary(x, L, low, high):
    if low<=high:
        mid=(low+high)//2
        if L[mid]==x:
            return mid
        elif L[mid]<x:
            return rec_binary(x, L, mid+1, high)
        else:
            return rec_binary(x, L, low, mid-1)
    else:
        return -1
```
These examples are at least as easy to code, and easier to understand, using iteration (for and while loops). So why bother with recursion at all?
Merge Sort

Here's the recursive definition:
To sort a list:
  Split it in two halves
  Sort each half
  Merge the two halves
Simple recursive implementation of merge sort

def rec_mergesort(s,low,high):
    if low<high:
        mid=(low+high)//2
        rec_mergesort(s,low,mid)
        rec_mergesort(s,mid+1,high)
        merge(s,low,mid+1,high+1)
When not to use recursion---Fibonacci disaster
When not to use recursion---Fibonacci disaster

Recall Fibonacci sequence 0,1,1,2,3,5,8,13,21,34,55,89,...

\[ F_0 = 0 \]
\[ F_1 = 1 \]
\[ F_N = F_{N-1} + F_{N-2} \text{ if } N > 1 \]
When not to use recursion---Fibonacci disaster

Recall Fibonacci sequence 0,1,1,2,3,5,8,13,21,34,55,89,...

\[ F_0 = 0 \]
\[ F_1 = 1 \]
\[ F_N = F_{N-1} + F_{N-2} \] if \( N > 1 \)
When not to use recursion---Fibonacci disaster

Recall Fibonacci sequence 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ...

\[ F_0 = 0 \]
\[ F_1 = 1 \]
\[ F_N = F_{N-1} + F_{N-2} \text{ if } N > 1 \]

```python
def fibonacci(n):
    if n==0:
        return 0
    elif n==1:
        return 1
    else:
        return fibonacci(n-1)+fibonacci(n-2)
```
When not to use recursion---Fibonacci disaster

```python
def fibonacci(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fibonacci(n-1) + fibonacci(n-2)
```

It took 0.4 seconds to compute `fibonacci(30)` this way, 4.5 seconds for `fibonacci(35)`, 49 seconds for `fibonacci(40)`. The running time grows exponentially.
When not to use recursion---Fibonacci disaster

It took 0.4 seconds to compute fibonacci(30) this way, 4.5 seconds for fibonacci(35), 49 seconds for fibonacci(40). The running time grows exponentially.

A call to fibonacci(10) produces calls to fibonacci(8) and fibonacci(9), which in turn produce a call to fibonacci(6), two calls to fibonacci(7), and another call to fibonacci(8), which in turn....

The result is that the exact same function call is repeated many times, and the total number of calls to fibonacci grows exponentially.
Tower of Hanoi Puzzle

Move all the disks from the left-hand post to center post, one disk at a time. You cannot put a larger disk on top of a smaller one.
If you have only two disks, you can solve the puzzle in three moves....
To solve the puzzle with five disks, you first have to solve it for four disks (moving from left post to right post).
...then move the largest disk......
...then solve it again for four disks, moving from the right post to the center post.
def hanoi(k, i, j):
    if k == 0:
        return []
    else:
        other = 3 - i - j
        return hanoi(k - 1, i, other) + [(i, j)] + hanoi(k - 1, other, j)
How many moves?

- If number of disks = 0, number of moves = 0
- If number of disks = 1, number of moves = 1
- If number of disks = 2, number of moves = 1 + 1 + 1 = 3
- .....3 disks,......3+1+3=7
- .....4 disks, 7+1+7=15
- .....5 disks, 15+1+15=31
How many moves?

N disks: $2^{N-1}$

Check: $2^{N-1} + 1 + 2^{N-1} = 2 \times 2^N - 1 = 2^{N+1} - 1$

For a tower of 10 disks, this is 1023 moves.