Recursion--part 2
Binary search (again)
Binary search (again)

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

3 8 11 14 19 23 30
Binary search (again)

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

| 3 | 8 | 11 | 14 | 19 | 23 | 30 |

If $x$ is equal to the middle element, return the index of this element
Binary search (again)

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

| 3 | 8 | 11 | 14 | 19 | 23 | 30 |

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

| 3 | 8 | 11 |

If $x$ is less than the middle element, search for $x$ in the left half
Binary search (again)

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
Binary search (again)

This is a recursive definition---it defines searching a list in terms of searching smaller lists.
Binary search (again)

This is a recursive definition---it defines searching a list in terms of searching smaller lists.

Where does the recursion bottom out? What is the smallest case?
Binary search (again)

This is a recursive definition---it defines searching a list in terms of searching smaller lists.

Where does the recursion bottom out? What is the smallest case?

If the list is empty, just return -1 (x is not present).
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
Merge Sort (again)
Merge Sort (again)

The crucial step here was merging two small sorted lists into one large one.
Merge Sort (again)

The crucial step here was merging two small sorted lists into one large one.

\[ \begin{array}{ccc} 3 & 8 & 11 \\ & 1 & 2 & 9 & 14 \end{array} \]

\[ \begin{array}{cccc} 1 & 2 & 3 & 8 & 9 & 11 & 14 \end{array} \]
Merge Sort (again)

Earlier we described a 'bottom-up' iterative procedure for using this merge operation to sort a large array. (Merge adjacent sublists of size 1 into sorted lists of size 2, then adjacent sublists of size 2 into sorted lists of size 4, etc.)

It was not so easy to code!
Merge Sort (again)

Here's the recursive definition:

To sort a list:
  Split it in two halves
  Sort each half
  Merge the two halves
Merge Sort (again)

Here's the recursive definition:

To sort a list:
  Split it in two halves
  Sort each half
  Merge the two halves

| 19 | 23 | 11 |  8 |  3 | 30 | 14 |
Merge Sort (again)

Here's the recursive definition:

To sort a list:
   Split it in two halves
   Sort each half
   Merge the two halves
Merge Sort (again)
Here's the recursive definition:

To sort a list:
Split it in two halves
Sort each half
Merge the two halves
Merge Sort (again)

Here's the recursive definition:

To sort a list:
- Split it in two halves
- Sort each half
- Merge the two halves
Merge Sort (again)

Here's the recursive definition:

To sort a list:
  Split it in two halves
  Sort each half
  Merge the two halves

What's the smallest case? The method described only makes sense if the list has at least 2 elements, but lists with 0 or 1 element are already sorted!

The resulting code is MUCH simpler.