Functional Decomposition, Again
Lists of compound data can be useful
- The type \([X]\)
- The flatten function
- Using types to help us guess operations

The simplest recursive type is "Natural"
- Patterns of induction

What can we compute with Natural?
Today's Lecture

- More on how to analyze problems
- We've already seen
  - Analyze data dependence
    - Make functions to capture such dependence
  - Analyze data organization
    - Define types to capture data organization
- Today
  - Using case analysis
  - Creating auxiliary functions can be useful
A Simple Way to Sort

- Sort: \([X] -> [X]\)

- Example:
  - Given \([3,5,4,2,1]\)
  - Return \([1,2,3,4,5]\)

- Example:
  - Given \([100,3,400]\)
  - Return \([3,100,400]\)

- Can we devise a Method for doing this?
Functional Decomposition

- Small functions come up often
  - With Lists:
    - `append` used in `flatten`, and many others
  - With Naturals:
    - `+` used in `*`,
    - `*` used in `exp`, and
    - `-` used in `/`
    - `+, -, exp` all used in polynomials, linear algebra, and many others
Sorting

- We derived \textit{sort} and \textit{insert}
- We followed the recipe very systematically to derive both of them
- (Resulting code is in book)
Possible Orderings

- Example:
  - Given [1,2,3]
  - Return[[1,2,3], [2,1,3], [2,3,1], [1,3,2], [3,1,2], [3,2,1]]

- Can the natural induction pattern help us figure out how to implement this function?