Pseudocode: Recursion

Inductive definitions

- Many computations are naturally defined inductively
    - Base case: directly return the value
    - Inductive step: compute value in terms of smaller arguments

Inductive definitions

- Many computations are naturally defined inductively
    - Base case: directly return the value
    - Inductive step: compute value in terms of smaller arguments
- Factorial
    - $n! = n \times (n-1) \times \cdots \times 2 \times 1$
    - $0!$ is defined to be 1
    - `factorial(0) = 1`
    - For `n > 0`, `factorial(n) = n * factorial(n-1)`

Inductive definitions

- Many computations are naturally defined inductively
  - Base case: directly return the value
  - Inductive step: compute value in terms of smaller arguments
- Factorial
  - $n! = n \times (n-1) \times \cdots \times 2 \times 1$
  - $0!$ is defined to be 1
  - `factorial(0) = 1`
  - For `n > 0`, `factorial(n) = n * factorial(n-1)`

```
Procedure Factorial(n)
  if (n == 0) {
    return(1)
  }
  else {
    return(n * factorial(n-1))
  }
End Factorial
```

Inductive definitions

- Many computations are naturally defined inductively
  - Base case: directly return the value
  - Inductive step: compute value in terms of smaller arguments
- Factorial
  - $n! = n \times (n-1) \times \cdots \times 2 \times 1$
  - $0!$ is defined to be 1
  - `factorial(0) = 1`
  - For `n > 0`, `factorial(n) = n * factorial(n-1)`

```
Procedure Factorial(n)
   if (n == 0) {
     return(1)
   }
   else {
     return(n * factorial(n-1))
   }
End Factorial
```

- Recursive procedure
  - `factorial(n)` is suspended till `factorial(n-1)` returns a value

Inductive definitions on lists

- Inductive functions on lists
    - Base case: Empty list
    - Inductive step: Compute value in terms first element and rest

Inductive definitions on lists

- Inductive functions on lists
  - Base case: Empty list
  - Inductive step: Compute value in terms first element and rest
- Sum of numbers in a list
  - If `l == []`, `sum` is `0`
  - Otherwise, add `first(l)` to sum of `rest(l)`

Inductive definitions on lists

- Inductive functions on lists
  - Base case: Empty list
  - Inductive step: Compute value in terms first element and rest
- Sum of numbers in a list
  - If `l == []`, `sum` is `0`
  - Otherwise, add `first(l)` to sum of `rest(l)`

```
Procedure Listsum(l)
   if (l == []) {
     return(0)
   }
   else {
     return(first(l) +
               Listsum(rest(l)))
   }
End Listsum
```

Inductive definitions on lists

- Inductive functions on lists
  - Base case: Empty list
  - Inductive step: Compute value in terms first element and rest
- Sum of numbers in a list
  - If `l == []`, `sum` is `0`
  - Otherwise, add `first(l)` to sum of `rest(l)`
  - Can also add `last(l)` to sum of `init(l)`

```
Procedure Listsum2(l)
   if (l == []) {
     return(0)
   }
   else {
     return(last(l) +
                Listsum2(init(l)))
   }
End Listsum2
```

Insertion sort

- Build up a sorted prefix
- Extend the sorted prefix by inserting the next element in the correct position

Insertion sort

- Build up a sorted prefix
- Extend the sorted prefix by inserting the next element in the correct position

```
Procedure InsertionSort(l)
   sortedList = []
   foreach z in l {
     sortedList =
        SortedListInsert(sortedlList,z)
   }
   return(sortedList)
End InsertionSort
```

Insertion sort

- Build up a sorted prefix
- Extend the sorted prefix by inserting the next element in the correct position

```
Procedure InsertionSort(l)
  sortedList = []
  foreach z in l {
    sortedList =
       SortedListInsert(sortedlList,z)
  }
  return(sortedList)
End InsertionSort
```

```
Procedure SortedListInsert(l,x)
  newList = []
  inserted = False

  foreach z in l {
    if (not(inserted)) {
      if (x < z) {
        newList = newList ++ [x]
        inserted = True
      }
    }
    newList = newList ++ [z]
  }

  if (not(inserted)) {
    newList = newList ++ [x]
  }

  return(newList)
End SortedListInsert
```

Insertion sort, inductively

- List of length 1 or less is sorted
- For longer lists, insert `first(l)` into sorted `rest(l)`

Insertion sort, inductively

- List of length 1 or less is sorted
- For longer lists, insert `first(l)` into sorted `rest(l)`

```
Procedure InsertionSort(l)
    if (length(l) <= 1) {
      return(l)
    }
    else {
      return(
          SortedListInsert(
            InsertionSort(rest(l)),
            first(l)
          )
      )
    }
End InsertionSort
```

```
Procedure SortedListInsert(l,x)
    newList = []
    inserted = False

    foreach z in l {
      if (not(inserted)) {
        if (x < z) {
          newList = newList ++ [x]
          inserted = True
        }
      }
      newList = newList ++ [z]
    }

    if (not(inserted)) {
      newList = newList ++ [x]
    }

    return(newList)
End SortedListInsert
```

Summary

- Many functions are naturally defined in an inductive manner
  - Base case and inductive step
  - For numeric functions, base case is typically 0 or 1
  - For lists, base case is typically length 0 or length 1
- Use recursive procedures to compute such functions
  - Base case: value is explicitly calculated and returned
  - Inductive case: value requires procedure to evaluated on a smaller input
  - Suspend the current computation till the recursive computation terminates
- Warning Without properly defined base cases, recursive procedures will not terminate!