Introduction to Computer Science

Recursion and Wrap-up

Iain Styles
6 December 2016

Recap

- With Dave:
 - Efficiency
 - Complexity in time and space
 - Complexity classes
 - Correctness
 - Ways in which programs can be incorrect
 - Specifications
 - Invariants
- Today
 - Recursion
 - Not always very efficient
 - But easier to prove correctness
 - NOT EXAMINABLE

Recursion

- Iteration (loops) typically starts with a simple case and "adds complexity"
 - Example: sum a list of numbers x:

```
thesum = 0;
for(i=0, i<N; i++) {
   thesum += x[i];
}</pre>
```

 In recursion, we will try to do the opposite and break the problem down into simpler problems

Recursion

```
 x = [1,3,5,7,11,13,17,19] 
thesum = sum([1,3,5,7,11,13,17])+19
= sum([1,3,5,7,11,13]) + 17 + 19
= sum([1,3,5,7,11]) + 13 + 17 + 19
= ...
= sum([1]) + 3 + 5 + 7 + 11 + 13 + 17 + 19
= 1 + 3 + 5 + 7 + 11 + 13 + 17 + 19
```

Algorithmically....

```
int sum(x) {
    if(x.length==1) {
        return x[0];
    }
    else{
        return last(x) + sum(x[0:length(x)-2]);
    }
}
```

Recursion

- Recursive functions are those that call themselves
 - Usually on a "simpler" version of the problem
- Why do it?
 - Recursion is expensive
 - Have to store a "call stack" that records arguments, return address, local variables etc for each function call.
 - Iteration is "obvious"
 - Or is it?
 - Mathematical tools for "proving" things about iteration are underdeveloped
 - Recursion much easier to "reason" about
 - Some problems much easier to express recursively

General Pattern

- Two parts:
 - A "base case"
 - Simplest possible situation that cannot be further reduced
 - The recursive call

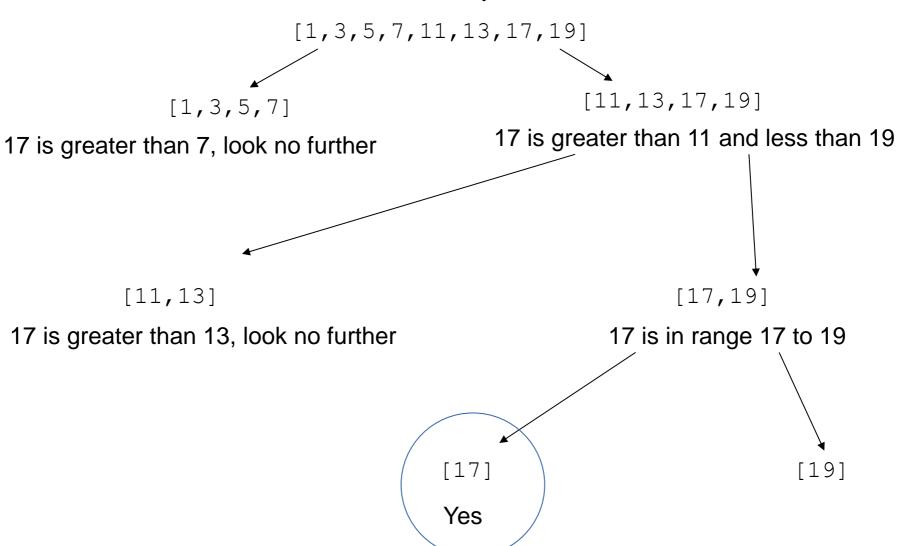
```
int sum(x) {
  if(x.length==1) {
    return x[0];
  }
  else{
    return last(x) + sum(x[0:length(x)-2]);
  }
}
```

Binary Search

- Problem: is an item in a list?
- Unsorted list: have to look at all the elements
 - Linear search easy to iterate
 - O(N)
- Sorted list: much more efficient: O(log N)
 - Binary Search
 - Split the list in half
 - Look at the upper/lower bounds
 - If item is within bounds, search appropriate sublist
 - "Natural" to do recursively
 - Dave showed you an iterative version

Binary Search

Is 17 in my list?



Binary Search

```
int binarySearch(x,item) {
   int middle = (start + end) / 2;
   if(end < start) {</pre>
      return -1;
   if(item==x[middle]) {
      return middle;
   } else if(item < x[middle]) {</pre>
      return binarySearch(x, start, middle - 1, item);
   } else {
      return binarySearch(a, middle + 1, end, item);
```

Why recursion?

- Prove correctness by "induction"
 - Prove true for simplest case (list of length 1)
 - Assume true for general case (list of length r)
 - Prove true for list of length r+1
 - "Structural Induction" is general proof method
- Allows rigorous proofs of program correctness
 - No equivalent tools for iterative algorithms
- Many algorithms operating on "tree-like" data structures can be expressed elegantly recursively
 - This will be studied in detail next term
- But not usually efficient
 - Building the "Call stack" costs time and memory
 - Machines work iteratively

Module Summary

- Number representations
 - Whole numbers
 - Fixed point
 - Negative numbers using two's complement
- Organisation of Computers
 - von Neuman architecture
 - Instruction sets
 - Translation of code into machine instructions (by hand!)

Module Summary

- Java and its virtual machine
 - Compilation and interpretation
 - Virtual machines combine them
 - Portability of bytecode (write once, run anywhere)
 - Stack-based computing stacks and frames
 - Mechanisms for subroutines (methods)
 - Bytecode
 - Objects and the heap
 - Garbage collection

Module Summary

- Efficiency
 - Time complexity
 - Complexity classes
- Correctness
 - Types of errors (syntax, run-time, etc)
 - Specification & correctness
 - Programming by contract
 - Invariants
- Recursion

Assessment Reminder

- MSc and YiCS
 - Two quizzes (one currently open)
 - Exam in May
 - 90 minutes
- MSc only
 - Reflection assignment
 - Currently being marked