How to Design Recursive Algorithms

Show termination and correctness. Show that the algorithm produces a correct solution.

- *Termination.* Show that the recursion — and thus the algorithm — always terminates.
  - *Making progress.*
    Explain why what each of your friends get is a smaller instance of the problem.
  - *The end is reached.*
    Explain why a base case is always reached.

- key things to address
  - making progress – show that your friends are guaranteed subproblems at least one smaller than you got
    - check for special cases or bugs that mean subproblems aren't really smaller
  - reaching the end – show that base cases can't be skipped
    - e.g. fib(n) = fib(n-1)+fib(n-2) with base case fib(1) = 1
      - fib(2) = fib(1)+fib(0)

How to Design Recursive Algorithms

- *Correctness.* Show that the algorithm is correct.
  - *Establish the base case(s).*
    Explain why the solution is correct for each base case.
  - *Show the main case.*
    Assume that the friends return the correct results for their subproblem, and explain why the correct answer is then produced from those results.
  - *Final answer.*
    Explain why the top level — the setup plus a correct solution to the initial subproblem followed by the wrapup — means that the final result is a correct answer to the problem.

- proof by induction
  - explain why the base case(s) correctly solve subproblems of that size
  - for the recursive case(s), assume that the friends return correct solutions for their subproblems – address
    - why you correctly split the problem into subproblems
    - why you correctly combine the friends' solutions into your solution

How to Design Recursive Algorithms

Determine efficiency. Evaluate the running time and space requirements of the algorithm.

- *Implementation.*
  Identify data structures and, as necessary, specific implementations of those data structures to efficiently support the algorithm. Also fill in any algorithmic details that are needed in order to establish the running time.
- *Time and space.*
  Assess the running time and space requirements of the algorithm given the implementation identified.

Recursive algorithms tend to lead to recurrence relations in one of two forms:

split off *b* elements
$T(n) = a\,T(n-b) + f(n)$ where $f(n) = 0$ or $\Theta(n^c \log^d n)$

divide into subproblems of size *n/b*
$T(n) = a\,T(n/b) + f(n)$ where $\Theta(n^c \log^d n)$

- *Room for improvement.*
  Are the targets met? Is it necessary to do better? If improvements in running time and/or space are needed, identify possible avenues for improvement.