CSE 3101 Design and Analysis of Algorithms
Meta Steps for Unit 2

Jeff Edmonds

This contains the **most important** concepts in this unit. You will not be able to pass this course without knowing and understanding these. The steps provided **must** be followed on all assignments and tests in this course. Do **not** believe that because you know the material, you can answer the questions in your own way. Though this material is necessary, it does not contain everything that you need. You must read the book, go to class, review the slides, and ask lots of question.

Chapters 8-12: Recursion & Friends

People do have a hard time getting recursion. But then a light goes off and they get it. I always feel that it is a matter of them trusting the friends and quite frankly trusting me. (See page 104 of "How to Think about Algorithms".)

The Steps: The following are the steps that needs to be *considered* (but not necessarily written) when you are designing a recursive algorithm within the friends paradigm. (Be sure that the code that you produce does not indicate that you missed any of these steps.)

1. Carefully write the specifications for the problem.
   (a) Precondition: Define the set of legal instances (inputs)
   To be sure that we solve the problem for every legal instance.
   So that we know what we can give to a friend.
   (b) Postcondition: Required output
   So that we know what is expected of us.
   So that we know what we can expect from our friend.
2. Focus on only one step. Do not trace out the entire computation.
3. Consider your input instance
   (a) Remember that you know nothing other than what is given to you by the preconditions.
   (b) Be sure each path in your code returns what is required by the postcondition.
4. If your instance is sufficiently small, solve it yourself as a base case.
5. Construct one or more subinstances:
   (a) It must be an instance to the same problem, i.e. meet the precondition.
   (b) It must be smaller according to some definition of size.
6. Assume by magic (strong induction) your friends give you a correct solution for each of these.
7. Use their solutions for their subinstances to help you find a solution to your own instance.
8. Do not worry about who your boss is or how your friends solve their instance. No global variable or effects (unless I say so)
9. If you want more from your friends, change the pre and/or postconditions, but
   (a) Be sure to document it.
   (b) Be sure you and all of your friends are using the same pre/postconditions.
   (c) Be sure to handle these changes yourself.
10. The code does not need to be much more complex than the following.

```
algorithm $Alg(a,b,c)$
⟨pre−cond⟩: Here $a$ is a tuple, $b$ an integer, and $c$ a binary tree.
⟨post−cond⟩: Outputs $x$, $y$, and $z$ which are useful objects.

begin
    if( $\langle a,b,c\rangle$ is a sufficiently small instance) return( $\langle [0], 0, emptytree\rangle$ )
    $\langle a_{sub1}, b_{sub1}, c_{sub1}\rangle$ = a part of $\langle a,b,c\rangle$
    $\langle x_{sub1}, y_{sub1}, z_{sub1}\rangle = Alg(\langle a_{sub1}, b_{sub1}, c_{sub1}\rangle)$
    $\langle a_{sub2}, b_{sub2}, c_{sub2}\rangle$ = a different part of $\langle a,b,c\rangle$
    $\langle x_{sub2}, y_{sub2}, z_{sub2}\rangle = Alg(\langle a_{sub2}, b_{sub2}, c_{sub2}\rangle)$
    $\langle x,y,z\rangle$ = combine $\langle x_{sub1}, y_{sub1}, z_{sub1}\rangle$ and $\langle x_{sub2}, y_{sub2}, z_{sub2}\rangle$
    return( $\langle x,y,z\rangle$ )
end algorithm
```

11. Give and solve the recurrence relation for the Running Time of this algorithm.

Formal Proof: The following are the formal steps similar to those for an iterative algorithm needed to prove a recursive program correct.

***Precond* $\Rightarrow$ *Precond*:** Prove that if your instance meets the preconditions, then each of your subinstances also meet the preconditions.

Smaller or Basecase: Prove that if your instance is *sufficiently small* according to some measure of your choosing, then it is a *base case* in which case you do not recurse. Otherwise, the subinstances that you give to you friends are (according to this same measure) *smaller* than your instance. Also your instance has a finite size.

***Postcond* $\Rightarrow$ *Postcond*:** Prove that if your friend's solutions meet the postconditions for their subinstances, then your solution meets the postcondition for your instance.

Prove that your solution for the base case meets the postconditions.

Running Time: There are two methods.

Recurrence Relations: Let a denote the number of friends that you have. Let b be such that each friend is given an instance of size $\frac{n}{b}$, where n is the size of your instances. Let c be such that the time that you personally spend is $\mathcal{O}(n^c)$. Solve the recurrence relation $T(n) = aT(\frac{n}{b}) + \mathcal{O}(n^c)$. If $\frac{\log a}{\log b} > c$, then the running time is dominated by the bases cases and is $T(n) = \mathcal{O}(n^{\frac{\log a}{\log b}})$. If $\frac{\log a}{\log b} < c$, then the time is dominated by the work in the top stack frame and is $T(n) = \mathcal{O}(n^c)$. If $\frac{\log a}{\log b} = c$, then all the levels require about the same amount of time and the the total time is $\mathcal{O}(n^c \log n)$.

Tree of Stack Frames: Compute and add up the work done in each stack in the tree of stack frames.

Binary Trees: A common example is recursion on binary trees.

Subinstances: If your input is a binary tree, then surely you will have two friends, one to which you give your left subtree and one to which you give your right subtree.

Wall Around Subtrees: To save yourself some time, you must trust your two friends. Imagine building a big wall around your two subtrees. The only information the friends have about the root or about the other subtree is what you pass them in their subinstances. The only information that you have is the root and what your two friends pass back to you in their solutions. From this information, and this alone, you have to be able to solve the given problem for the entire tree. In addition to your friends solving the given problem for the subtrees, what additional information might you want them to tell you?

Cases: The base case should be the empty tree. The general code needs to consider the case in which you have a big left and a big right subtree. Only if absolutely necessary should you be having an extra case for the tree consisting of a single node or when one of the subtrees is empty. Avoid cases with if statements in general. For example use $max(a, b, c)$ instead of separate cases for which is bigger.

Running Time: It is hard to use recurrence relations to compute the running time, because we do not know the sizes of the left or the right subtrees. Instead, we compute and add up the work done in each stack in the tree of stack frames. When the input consists of a tree and there is a friend for each subtree of the root, then the tree of stack frames of the computation mirrors the nodes in the tree. (Except there is also a stack frame for each empty tree hanging off the leaves of the input tree. This does not more than double the number of nodes of the tree.) If each stack frame does a constant amount of work, then total work is $\Theta(n)$ given a tree with n nodes.

Many Children: If each node may have many children, then given a tree, a stack frame simply loops over the children of the root. No explicit base case is needed because if the root has no children then its stackframe will not recurse. When computing the total running time, note that the stack frame corresponding to a node in the tree no longer does a constant amount of work but an amount proportional to the number of children it has. Lets count this work by saying that it is proportional to the number of parent to child edges it has. Hence, the total time is proportional to the total number of parent to child edges. Each such edge can equally be counted in the other direction. Hence, the total time is proportional to the total number of child to parent edges. Clearly, this is equal to the number of nodes n in the tree because each node has at most one parent.