Playing Against the Computer
Recursion & the Minimax Algorithm

Key to Acing Computer Science

If you understand *everything*, ace your computer science course.

Otherwise, study & program something you don't understand, then see
Key to Acing Computer Science

Winning Tic-tac-toe

Anticipate the implications of your move.

Avoid moves which could
enable your opponent to win.

Attempt moves which would
force your opponent to lose, so you win.

How to pass the buck: recursion!

Imagine this:

You have all sorts of friends who are willing to help you do your work with two caveats:

- They won't do all of your work
- They will only do the *type* of work you have to do

Example:

How would you do your homework consisting of 100 calculus problems?

Easy – do the first problem yourself and ask your friend to do the rest

How could this approach work?

What actually happens is that your friends doing most of your work have friends of their own. Everyone eventually does a small part of the work and piece it back together to do the work in its entirity.

In the last example, 100 total people would be working on the homework, *each person doing only one* problem each and *passing on the rest*.

Important: the person to do the last homework problem does it entirely without help.

The Parts of Recursion

- **Base Case – this is the simplest form of the problem which can be solved directly. In the example earlier this would be doing 1 homework problem.**
- **Recursive Step – this is when a *smaller identical problem* is solved as part of solving the larger problem. In the earlier example, this would be passing along all but one of the homework to a friend.**

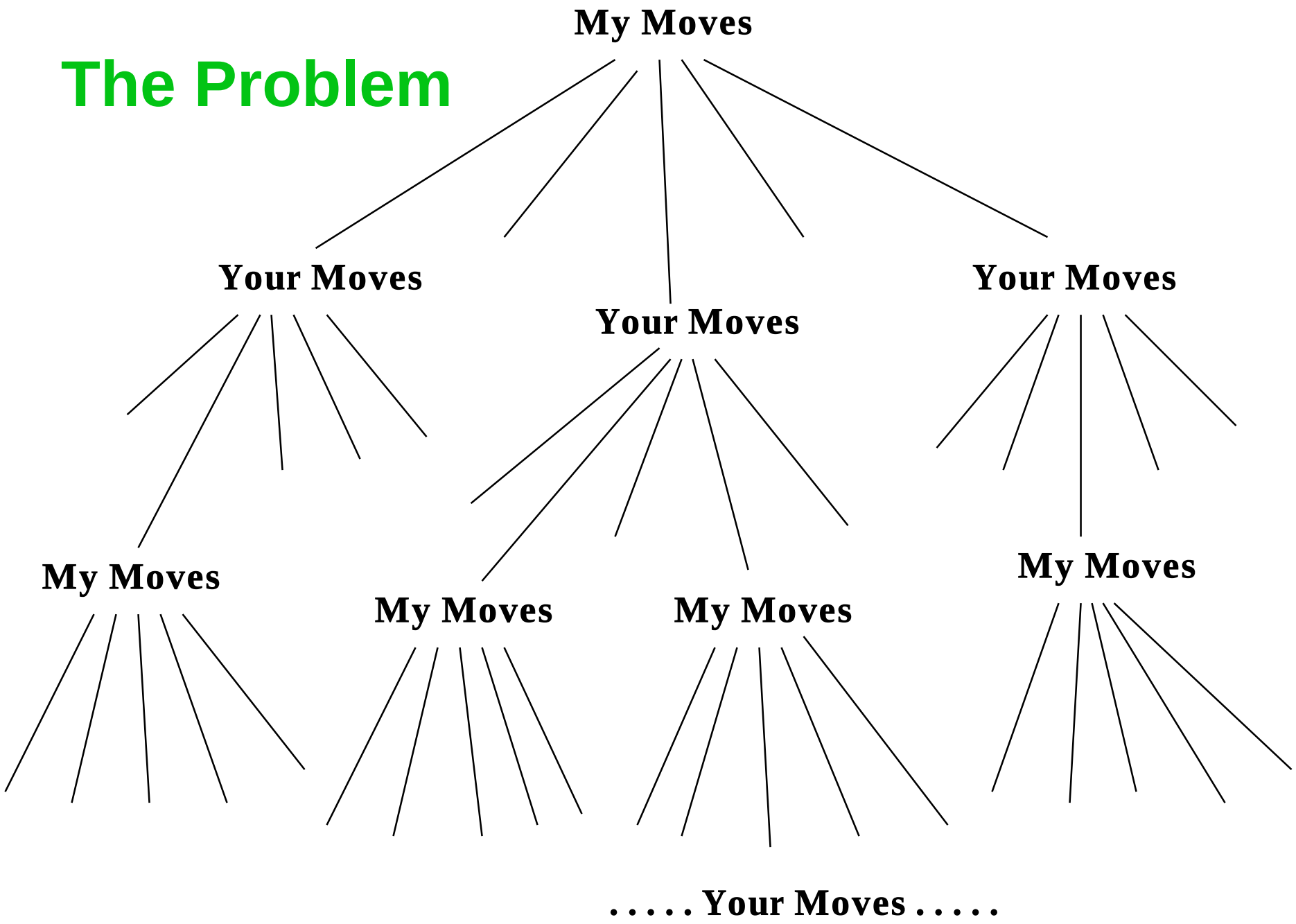
Example: Finding the Maximum

- Imagine you are given a stack of 1000 unsorted papers with grades on them and told to find the maximum grade.
- You can only get help from friends in finding the maximum and in less than the 1000 papers you were given.
- What do you ask your friend to do to help you find the maximum in the least effort on your part?

Example: Sorting

- **Now imagine you must sort those 1000 papers by grade. Define a recursive algorithm to solve this problem. To do this define two parts:**
  - **What is the simplest number of papers to sort, and how would you sort them?**
  - **How would you divide the task in to smaller sorting tasks and integrate theses solutions to sort the entire set of N papers?**

The Problem


My Moves
Your Moves
Your Moves
Your Moves
My Moves
My Moves
My Moves
My Moves
. . . . . Your Moves . . . . .

Example: the Minimax Algorithm

- In many games, one players loss is another player's gain.
- A winning strategy for this type of game is to minimize the maximum potential gain of your opponent and *assume your opponent is following the same strategy*.
- Better than brute force lookahead:
  - Consider all possible moves to the bitter end
  - Pick the move that leads to a win, if possible
  - Why not program Computer Chess that way?

Example: the Minimax Algorithm

- **Since your opponent is following the same strategy, and game moves eventually end the game, this algorithm can be implemented recursively.**

Anticipate the implications of your move.

Avoid moves which could enable your opponent to win.

Attempt moves which would force your opponent to lose, so you win.

A Winning Computer Player for Tic-tac-toe

- **What is the algorithm? Assume the input is a board and whose turn it is (X or O):**
  evaluate(board, position)
  - **What is the base case? What is the simplest board to evaluate win, loss, or cat outcome? Remember you assume both players are doing everything possible to win.**
  - **What is the recursive step?**

How can we adapt this to Chess?

- Can't go to the bitter end: Takes too long: "Eons"
- How can we see which move is better?
  - Need something like a *Board Evaluation Function*
  - What does it use to evaluate board?
    - Material: who has more of what kind of pieces
    - Position ?
    - Anything else?
  - Can then use limited look-ahead to suggest who comes out ahead
    - Will not be perfect
    - How do you avoid traps (sacrifices)?
- Use Minimax based on Board Evaluation Functions