# CS280 Notes 1/27

*by Christian Sagel*

## Questions

The values are being inserted into the nodes in the wrong order?

If we remove the last value from a node, do we get rid of it and collapse the nodes around it? Yes!

We don't have to delete the BNode arrays since they are statically created?
To remove values from a node, do we just 0 them out? We just shift everything over.
When deleting from the end of an array, count--. Decrement the size.

Do we apply the binary search algorithm on the find method?
When performing the binary search, do we apply the algorithm on each node we find or somehow across the entire list at once? We apply the binary search to each node in turn, not on the list at once. (There's no way to apply it on a linked list)

Assignment operator should add new nodes to match the size or to clear the existing list and just remake the nodes? This has to do with efficiency, I guess. Mead leaves it up to you, as long as yo
In hash tables you end up reusing nodes.

Write your helper functions first, for everything you think you need to do. For example, for Mead's implementation of BList:

- Given a value, find what node its on
- Split nodes

One function, one task!

## Recursion Details

If we are asked to write code on a test, we will probably be asked to write a recursive function.
The optimized functions in assembly will have less instructions.
Mead usually optimizes between o2 and o3 for gcc. o2 is usually safest, o3 sometimes makes incorrect assumptions about the code.
We may gain FPS by optimizing the code wit our compiler.

Tracing function calls

Given a simplistic view of the the stack,
If the compiler finds no dependencies, it is free to rearrange the assignments.
The stack. Each function's portion of the stack is called an activation record or stack frame.
The stack grows and shrinks during program execution.
We call local variables automatic.
When doing operations on containers recursively…
The head is the iterative case, the tail is the recursive case.

Recursive reverse of array:
Swap the ends, reverse the middle.

There are some languages out there that have no iteration, but that only do recursion.
We are taught iterative process first, even though recursion is more elegant, due to iterative being more performant.

Writing a recursive function that can reverse a singly-linked list was a common interview question.
Using recursion on a SLL is generally not a good idea.

Mead was disappointed that we didn't find the typo in the Fibonacci example before him.
A recursive Fibonacci sequence function call has… a fibonacci sequence number of calls too. It ends up being an exponential algorithm, very slow. 2^n

The problem with the recursive version is that later calculations depend on the earlier ones, these were discarded
The iterative version saves and uses those previously-calculated values.
This saving of previously calculated values is termed dynamic programming.

The eight queens problem: A brute search algorithm.
The knight's tour is similar to that too.

## Abstract Data Types

Mead prefers Data Structures being taught in the 4th semester.
C makes a terrible language for abstraction due to it not being hidden.
We use C++ and OOP to protect ourselves from our own mistakes.
For Mead if the hardware doesn't work, just throw it out and get a new one.
Binary -> Assembly -> High level languages

The effort required to express the solution in different languages.
Higher-level languages came about by identifying the necessary constructs:

Mead just *doesn't* understand why there's floating point instructions in his code… that has no floating point variables!

64-bit machines tend to be slower because the pointers are bigger. Most users won't care about the increased size. 64 bit also brings more instructions.
The biggest change is the increased amount of registers.

Typically we call the drivers the client. The client doesn't want to know the underlying structure of the code.
The interface is opaque. (Handles (pointers to pointers) are considered opaque)
In OOP, you pass a message to the object.

Mixing linked lists and arrays allow us to mix the best of both.
We don't want to directly manipulate our data structures in our drivers. We want all that to be hidden behind abstraction and the interface.