CS280 Notes 1/20

by Christian Sagel

A lot of the issues came from people not checking Doxygen warnings! And Dr. Memory!
Most objects in use = high watermark in Software Engineering. This will tell you how much RAM your game is going to need.

When you are checking for double free, is it okay to go through the free list and check for double free?

It's safe to call delete on a null ptr.

Is it safe to just run Valgrind instead of Dr. Memory?

It's very rare for Dr. Memory to catch things that Valgrind didn't. If we mess something up the whole memory system is suspect. Make damn sure the memory manager is fully debugged before using it in the game.

On Windows 10 you need the latest version of Dr.Memory.

If you aren't struggling with linked lists, you aren't challenged enough. We are no longer doing linked lists for their own sake. Now we are doing linked lists for using it. We will be doing one with increased performance that blows the STL's one off the water.

We will analyze several loops: What does each loop, which one is fastest/slowest?
He gave us a couple minutes.
The 3 functions made an identity matrix.
The first one was written by a CS120 student?

We will count comparisons, assignments, multiplications and divisions.
We just have to read the bodies.

Test 1: 0 comparisons, N^2 assignments, N^2 multiplications, 2N^2 divisions = 4N^2
Test 2: N^2 comparisons, N^2 assignments, 0 mults, 0 divisions
Test 3: 0 comparisons, (N^2)+n assignments, 0 mults 0 divisions

Taking the big O(), we drop the constants and on paper all are N^2.
The only way to test if all your code has been executed. Coverage tools will keep track of which lines got executed and which ones didn't.
Code that doesn't get executed doesn't get tested.
Writing test cases that test for all possible cases on your application.

Main is the not the first function to get called. Main's overhead comes from allocating memory Dynamic memory allocation is one of the slowest operations.

Profiles help find bottlenecks.
Test1 was shown to be the slowest.

As a rule of thumb you never wanna print to the console when benchmarking. Just occasionally. And even better to print to a file.
What we see and evaluate on paper may not always translate properly.

Everyone is looking like what? When we profiled past 60,000 to 70,000 it went from 9s to 1m50s because it running out of memory and having to do a lot of swaps.
On a real test you would reboot the computer between tests.

Branching with if statements is expensive. Branching slows down the CPU.
Branching misprediction.

In an if statement, you can use any expression.
Switch statements only work with constants and integrals. It uses a binary search.
Micro-optimizations should be thoroughly documented.

Sometimes its impossible to test averages.
We have to think of worst cases and best cases. Average tests won't be detected.
There's a self-check table for testing our analysis of complexity.

Assignment

Normally in a linked list, there's one node per item. For this we will have multiple items per node (4,8, etc). We will keep these items of number power of 2.
We might have.
This is similar to a Btree. Btrees usually have 1000 elements per node. We will do something similar.

We have 2 primary ways to add. Insert and push_back/push_front.
If they are sorted, we will do a binary search tree to find items within a node.
O(N) 1,000,000
O(logN) 20

We don't want to dynamically allocate the array at run-time. We will be doing static arrays for the nodes.

We only have two types of exceptions. If we get a wrong subscript, we throw that. If we can't allocate, we will throw another exception.
It's a templated list.

Elements will be packed from left to right.

T has to support the default constructor (arrays have to be default constructed)
Subscript operators for L-values and R-values.
Insert will be our hardest function to write. Do it last!

What if a node is full and it needs to be inserted? If so it has to be split into two equal halves and decide where that item will go?
Historically this has been the hardest assignment for this class. Mead doesn't know why!

There are a lot of edge cases in the insert.
Always start early and do the simple stuff first.
Whatever size array fits the size of your cache will give you the best performance?
Nodes don't get merged in a Blist. When a node becomes empty we unlink it.

In a Blist every node is half-full.
When we split, we have to create a new node. A specific half of the node will go towards the new node depending on which side will the new inserted element go into.
This assignment will be due in 2 weeks.

We will go over skip lists at the end of the semester.