CS280 Notes 1/13

by Christian Sagel

Questions

“When using new to allocate memory, you must wrap it in a try/catch block. Do not use std::nothrow. If you don't know what that is, then you're probably not going to use it.”
How is this done?

“When do we sign the bytes? I am confused on whether it is done when debug is turned on or off”

“When setting the free list, where in the memory block should the free list be pointing at? The start of the block or… the data bytes?”
The free list should be pointing at the address of the logical block.‘

“What do we do when we catch an std::bad_alloc after new?”
We throw an OAException!

“How to call memset?” memset takes hexadecimal values.

Unexpected exceptions?

Assignment

For each object in the page list, we check whether its on a free list. If it is being used, we call that function with the address of the block and the size of the block.
Since the memory manager has no idea what it is used for, By allowing the driver to dump it out, you can get more information about it.

When do you do the validate, you check the pad bytes on the left and right. If either of those fail, you pass the block back to validate callback.

Constant and linear-time algorithms.
We have to dynamically allocate memory for the label.
How to do boundaries in constant time. Can we assume the pages start at 0. (In practice we can’t).

Given an address on a block: Figure out how big the block is. THe address has to be divisible.
Subtract it by another address and you get the value.
Then you mod that number by the size of the block.

Introduction to Algorithm Analysis

Complexity, constant-time vs linear time.
The fundamental question: WHat does it mean for it to be better?
Faster? Less memory? Easier to implement?

First to get it to be made correct, then efficient.
Every data structure and its algorithms are suited for different tasks. One size does not fit all.
Vectors are cache-friendly, hence the fastest algorithm usually.

Linear search.
If the array sorted, we would do a binary search. After every comparison, the size of the data has been cut in half.
Most number of iterations: log base 2 of n

Linear search vs binary search.
Binary search is a logarithmic-time algorithm. One of our best algorithms.
Sorting is a very expensive operation.

By doing simple analysis of the algorithm you can tell right away when the algorithm will be useful or not. Looking at the big picture, the algorithms are independent of the language, data type.
Micro-optimizations have no bearing when compared to the differences in scale of the algorithms. Doesn't matter how much you optimize a bad algorithm.

binary search with log base 2 = 10, 1000 with linear, n squared is 1 million

Given uniform distribution, it would take size/2 to..
Can't use binary search with no random access iterator.
You gain very little from having a sorted linked list.

Informally, we want to figure out the number of steps required to perform the computation.
The goal is to write a formula (equation) for the computation in terms of the size of the problem.
With logarithmic algorithms, when we double the data it only takes one more comparison. They do have to be sorted, though.
Some parts of the formula are constant (Constants of proportionality)
The growth rate is the part of the formula that varies with the size of the problem.
The dominant term is all that matters to us. n

The notation is O( ), called the "Big Oh", "growth rate", "order of", "proportion"
The performance of the algorithm as the data set grows.

Algorithms that grow at the same rate are considered the same kind of algorithms.

There's 2^n algorithms.
Elegant code, but a stupid amount of comparisons. But naive!

Lower bound will tell you the fastest the algorithm will go. This is useful to find out when the algorithm can never be more optimized.
The size of the data.
We will be doing a lot of recursion in the course.

In general, the algorithm we use is not based on the size of the data?
As the size of the data grows, we can discard the constant time.
n^2 algorithm, throw away the other terms

Try to memorize:
2^10 = 1024
2^20 = 1 million
2^30 = 1 billion

Andy giveth and Bill taketh away

These are why hash tables are sooo good?
O(k): Constant
O(log N): Logarithmic
O(N): Linear
O(N lg N): N log N. A lot of sorting algorithms use this. For every item in the data set, we are going to touch it 'log N' times.
O(N^2): Quadratic
O(N^3): Cubic
O(N^k): Polynomial
O(2^N) Exponential

The STL has been switching to using hash tables.

f(n) = dominant term + lesser terms
O(k) = O(1) = O(k) // constant
We are usually looking for the tight bound. Usually called big θ.