# malloc()

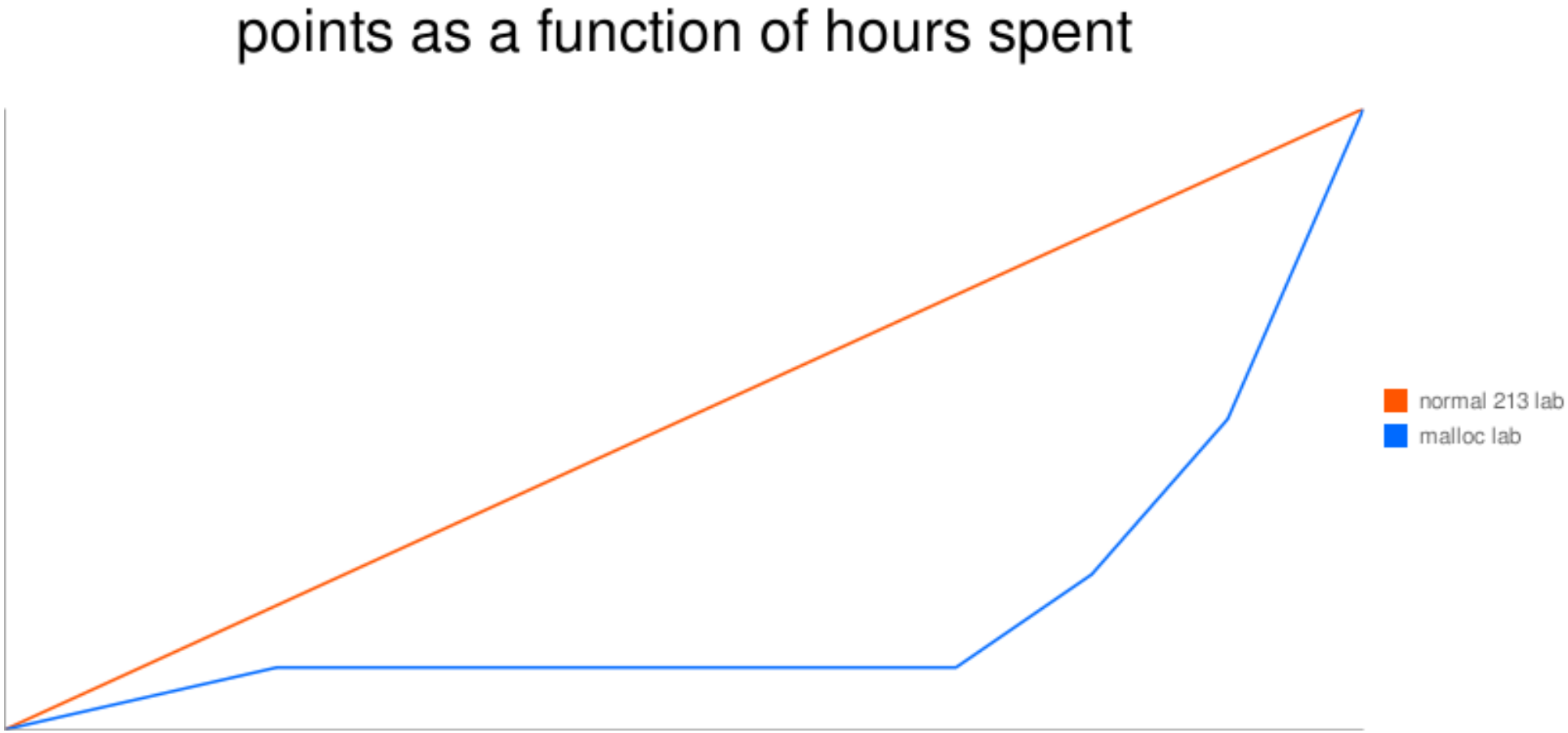
(serious business)

# A Series of Dire Warnings

- malloc() is hard
- it's weighted heavily in your final grade
- you can't start at the last minute
- a few standard solutions, but lots of design room
- challenging to debug
- easy to get a zero
- hard to get help from TAs
- easy to discover that your solution will never work after you've spent 20 hours on it
- easy to break your working solution
- children will mock you in the street

# malloc() is hard

- famously so
- probably the hardest programming assignment in your undergrad career (so far)
- "can't do malloc" => "will never pass OS"
- if your interviewer is a CMU alum, they will ask about this lab
- "subtle bugs"
- can be a massive (50 hours+) effort




can't do malloc() at the last minute and get a C...
you'll probably get a zero!

# no partial credit

- regardless of what you might have heard
- or heard from your friends from previous semesters
- allocator from book doesn't pass the bar: no credit
- broken allocator doesn't pass the bar: also no credit

if you don't complete at least an explicit allocator implementation, you will get a zero!

# subtle bugs (two kinds)

**"nothing works!"** can be caused by one bad line

**"it almost always works!"** can ALSO be caused by one bad line

Debugging is twice as hard as writing the code
in the first place. Therefore, if you write
the code as cleverly as possible, you are,
by definition, not smart enough to debug it.
-- Brian W. Kernighan

# please use version control

- you will be sad if you dont :(
- "my allocator worked just five minutes ago!"
- just be careful enough to copy working solutions to another directory somewhere
- or do the symlink thing from the handout (malloc.c ----> mm-mine.c)
- or just man up and use svn/cvs/etc

# 90/10 rule

"The first 90 percent of the code will take half the time, and the last 10 percent of the code will take the other half of the time."

- if your allocator is 90 percent working, you're only half done!
- it happens all the time that a crippling malloc design flaw is discovered hours before the due date
- leave yourself time to start over

# ta's are very unhelpful

- as a rule
- also, we don't like debugging code any more than you do
- probably less so, actually
- don't expect to hand us your laptop and wander away
- we don't want to look at your code
- we can discuss your design
- we can help troubleshoot VERY specific problems ("allocations of size blah corrupt my heap after coalescing the previous block at this line number..." is specific, "it segfaults" is not)
- two ta's are skipping town the week malloc is due
- come see us now, not later
- set up one-on-one appointments by email if you'd like

# how to beat malloc

# plan of attack

- read malloc section in the textbook
- look at the textbook allocator
- copy it out and make it work for 64-bit machines
- understand how EVERY line works (especially the macros)
- look at k&r allocator for another (slightly different) example
- save this allocator away
- congratulations! you have zero points (much too slow)

<---------**if you aren't at least this far, you are doomed**------->

- convert to explicit allocator
- save this allocator away (you should have points now!)
- start getting fancy! (segmented lists, other data structures)
- start getting obsessive! (optimize for the traces, etc)
- congratulations! you know more than when you started

# words of wisdom

- don't use printf() for debugging
- DO use gdb
- DO write the heap checker as you write your code (these should be free points)
- DO start early and often
- don't feel bad about throwing away broken solutions (starting over will clear your head)
- if you're stuck, come to office hours this week instead of next week
- use macros to maximize code readability
- write other tests besides the
- really really, please don't put this off to the end