The Ethics of Extreme Performance Tuning (excerpt)

Andrew W. Appel

Programming challenge

Implement a ***correct*** and ***fast*** integer cube-root function.

Correct: On any input (not just the "test harness"), it must have behavior indistinguishable from this reference implementation:

```
#include <math.h>
#include "root.h"
int quickroot(int i) {
  return (int)cbrt((double) i);
}
```

Fast: When connected to the "test harness" driver, the program should run as fast as possible.

This challenge was designed by Guy J. Jacobson '81
in 1995 when he was teaching COS 333 at Princeton University

Fast integer cube roots

root.h

```
int quickroot(int);
```

slowroot.c

```
#include <math.h>
#include "root.h"

int quickroot(int i) {
  return (int)cbrt((double) i);
}
```

Floating-point cube root from math.h

testharness.c

```
#include <stdlib.h>
#include "root.h"

main (int argc, char *argv[]) {
 int i, j;
 srandom (atoi (argv[1]));
 for (i = 0; i < 10000000; i++)
    j = quickroot (random());
 exit (0);
}
```

Performance measurement

(On a 1995 computer, much slower than today's)

testharness.o + slowroot.o: 20 seconds

testharness.o + noroot.o: 2 seconds

noroot.c

```
#include <math.h>
#include "root.h"

int quickroot(int i) {
  return 0;
}
```

Note: noroot.c is really fast, but is not ***correct,*** that is, **fails** "on any input, it must have behavior indistinguishable from this reference implementation"

Challenge:

root.h

```
int quickroot(int);
```

fastroot.c

```
#include "root.h"
int quickroot(int i) {
   .
   .   /* something really fast */
   .
}
```

How to do it

```
return (int)cbrt((double) i);
```

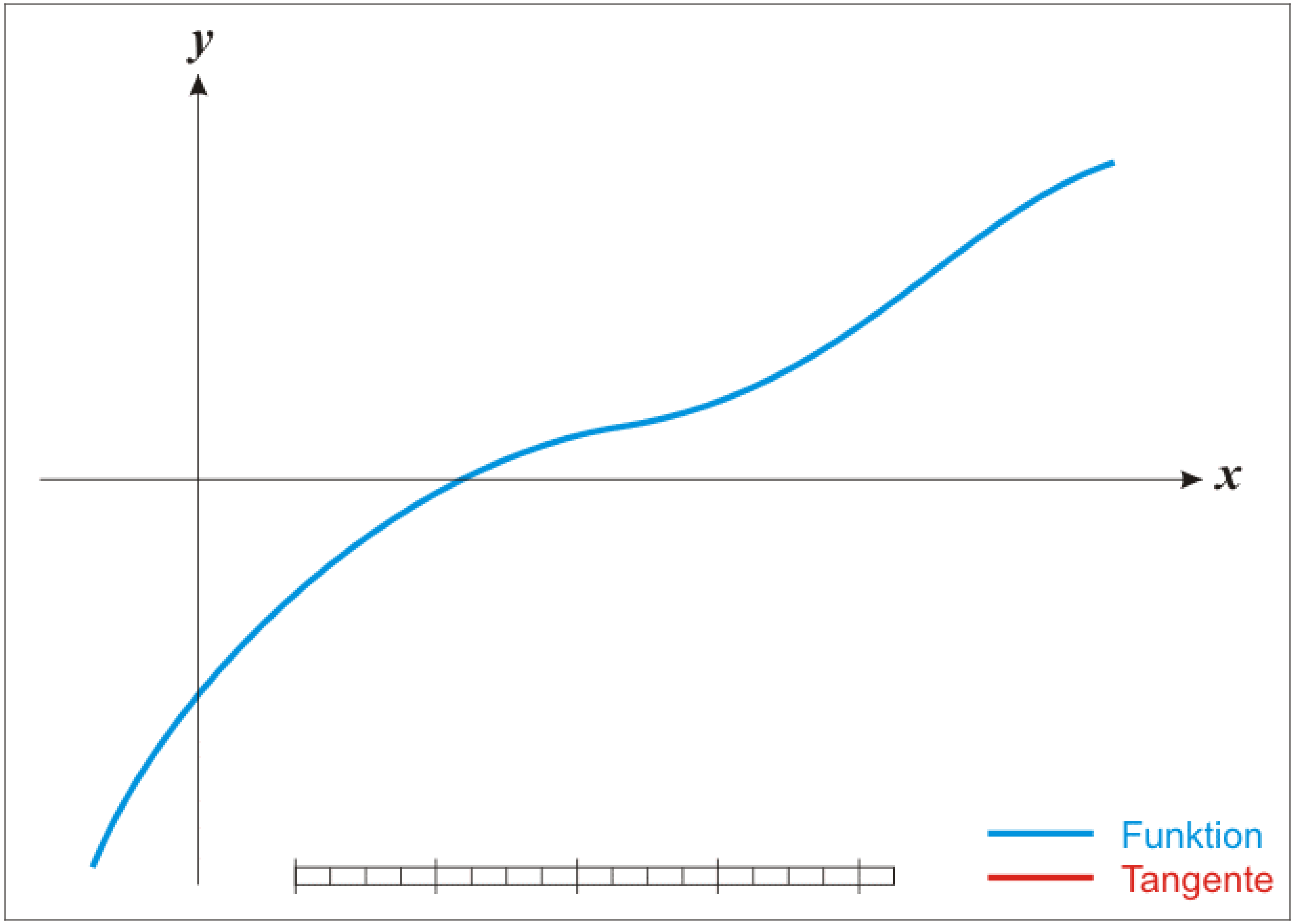
How can ya beat the highly tuned `cbrt` function from the math library?

I dunno, use Newton's method?

But doesn't the `cbrt` function already use Newton's method?

Um ...

Newton's method



To see this animated:
https://commons.wikimedia.org/wiki/File:NewtonIteration_Ani.gif

How to do it

```
return (int)cbrt((double) i);
```

How can ya beat the highly tuned `cbrt` function from the math library?

I dunno, use Newton's method?

But doesn't the `cbrt` function already use Newton's method?

Um ...

Wait, I got it!
`cbrt` calculates 64-bit precision, but we need only 32-bit precision, so Newton's method needs fewer iterations

Before-lecture cogitation

Think about how you would solve this problem.