304 lecture notes

up to September 7, 2023

contents

preface

These are Hayward's course notes for the first few lectures of CMPUT 304 Algorithms II. (After that, I expect you to take your own notes.)

Keep in mind:

- analysis starts with precise definitions
- for each algorithm, ask:
  - can we prove it correct?
  - can we give its worstcase runtime?
  - can we do better?

1
lecture summaries

Here's what we've covered so far (and some of what we might cover next). For each lecture, a recording will be posted on eclass in section *reference materials*.

1.1 lecture 1

- syllabus (course outline)
- collatz conjecture

1.2 lecture 2

- algorithm: word origin?
- linear function?
  - $f(n) = an + b$
  - more generally, any $f(n) \in \Theta(n)$

```
 O( n ) is the set of all functions g(n) such that

  there exists a constant c > 0 and a threshold

  integer n_0, such that for all integers t > n_0,

  g(t) <= c * t

Omega(n) is the set of all functions g(n) such that

  there exists a constant c > 0 and a threshold

  integer n_0, such that for all integers t > n_0,

  g(t) >= c * t

give me an example of a function in Omega( n ) ?

   g(n) = n

   g(n) = 3n + (lg n)^73  + 1239871234

   g(n) = 1

   g(n) = lg n + lg (lg n)) + (lg n)^2

-----
O( f(n) ) is the set of all functions g(n) such that

  there exists a constant c > 0 and a threshold

  integer n_0, such that for all integers n > n_0,

  g(n) <= c * f(n)
```

- Prove/disprove: for an input of two integers whose max has exactly n bits, (so size of input is $\Theta(n)$), WC runtime school add'n alg. in $O(n)$.
- P/d: WC runtime school add'n alg. in $\Omega(n)$.
- P/d: WC runtime school add'n alg. in $\Theta(n)$.
- P/d: WC runtime of any add'n alg. in $\Omega(n)$.

[source: google 2023/09/07] Middle English algorism, via Old French from medieval Latin algorismus. Arabic source, al-Kwarizmi, 'the man of Kwarizm' (now Khiva), name given to 9th-century mathematician Abu Ja'far Muhammad ibn Musa, author of widely translated works on algebra and arithmetic.

worstcase runtime of addition?

- as a function of input size n, what is worstcase runtime $f(n)$?
- (this is called *asymptotic analysis*)
- input? postive integers x, y
- input size? amount of memory needed to represent x, y
- typical integer representation: binary
- how many bits needed to represent x, y?
- $1 + \lceil \lg(x) \rceil \ + \ 1 + \lceil \lg(y) \rceil$
- what is runtime of usual addition algorithm to add x, y?
- usual addition algorithm? you saw this in elementary school (you saw decimal, we use binary)

```
carry     1 1 1     1 1 1     1 1 1
x           1 0 1 0 1 1 1 0 0 0 1
y     + 1 0 0 1 1 0 1 1 1 0 1 1 1
        -------------------------
sum       0 0 0 0 1 1 1 0 1 0 0 0
```

- time to add x, y with school algorithm is $O(\lg(x) + \lg(y))$
- for a function $f(n)$, define $O(f(n))$
- answer: $O(f(n))$ is the set of all functions $t(n)$ for which there exists a positive constant c and some threshold integer n_0, so that for all integers $m > n_0$, $t(m) \leq cf(m)$
- $O(\lg(x) + \lg(y)) = O(\max(\lg(x), \lg(y)) = O(\lg(\max(x, y)))$
- conclusion: for $x < y$, school addition algorithm has runtime $O(\lg(y))$
- so to add two numbers each at most t, runtime is $O(\lg(t) + \lg(t)) = O(2\lg(t)) = O(\lg(t))$

- can we say that addition is in $O(\lg(n))$, where $\lg(n)$ is the number of bits needed to represent the two input integers?
- yes

  so, can we do better?
- no: addition takes $\Theta(t)$ time, where t is the number of bits needed to represent the two input integers
- what is $\Theta(f(n))$?
- recall $\Omega(f(n))$ is ...
- $\Theta(f(n))$ is the intersection of $\Omega(f(n))$ and $O(f(n))$
- to say that a problem (addition) takes $\Theta(f(n))$ time, I mean that the worstcase runtime of any algorithm for that problem is in $\Omega(f(n))$, and that there is some algorithm whose worstcase runtime is in $O(f(n))$
- why is addition in $\Omega(\lg(n))$ time?
- to prove that any addition algorithm takes at least some constant time $\lg(n)$ time, argue by contradiction: any algorithm that looks at every input bit takes a $\Omega(\lg(n))$ time
- but if you don't look at every bit, then you can have two instances, where we change the value of the bit that your algorithm does not look at, and those two instances will have different sums, but your algorithm will have to give the same answer for both instances, so your algorithm will get at least one of those two instances wrong

- old-school square root algorithm example: see webnotes (warmup)