CS 161 **Problem Set 1**
Autumn 2017 **Due:** Friday, October 6, 2017 at 3pm on Gradescope.

Style guide and expectations: Please see the "Homework" part of the "Resources" section on the webpage for guidance on what we are looking for in homework solutions. We will grade according to these standards.

Exercises

Exercises should be completed **on your own.**

0. **(1 pt.)** Have you thoroughly read the course policies on the webpage?

   [We are expecting: The answer "yes."]

1. **(1 pt.)** See the IPython notebook `HW1.ipynb` for Exercise 1. Modify the code to generate a plot that convinces you that $T(x) = O(g(x))$.[1]

   [We are expecting: Your choice of c, n_0, the plot that you created after modifying the code in Exercise 1, and a short explanation of why this plot should convince a viewer that $T(x) = O(g(x))$.]

2. **(3 pt.)** See the IPython notebook `HW1.ipynb` for Exercise 2, parts A, B and C.

   (A) What is the asymptotic runtime of the function `numOnes( lst )` given in the Python notebook? Give the smallest answer that is correct. (For example, it is true that the runtime is $O(2^n)$, but you can do better).

   [We are expecting: Your answer in the form "The running time of `numOnes( lst )` on a list of size n is $O(\_\_\_)$.", and a few sentences informally justifying why this is the case.]

   (B) Modify the code in `HW1.ipynb` to generate a picture that backs up your claim from Part (A).

   [We are expecting: Your choice of c, n_0, and $g(n)$; the plot that you created after modifying the code in Exercise 2; and a short explanation of why this plot should convince a viewer that the runtime of `numOnes` is what you claimed it was.]

   (C) How much time do you think it would take to run `numOnes` on an input of size $n = 10^{15}$?

   [We are expecting: Your answer (in whichever unit of time makes the most sense) with a brief justification, that references the runtime data you generated in part (B). You don't need to do any fancy statistics, just a reasonable back-of-the-envelope calculation.]

3. **(4 pt.)** Using the definition of big-Oh, formally prove the following statements.

   (a) $2\sqrt{n} + 6 = O(\sqrt{n})$ (Note that you gave a "proof-by-picture" of this in Exercise 1).

   (b) $n^2 = \Omega(n)$

   (c) $\log_2(n) = \Theta(\ln(n))$

   (d) 4^n is **not** $O(2^n)$.

   [We are expecting: For each part, a rigorous (but short) proof, using the definition of $O(), \Omega()$, and $\Theta()$.]

[1] **Note:** There are instructions for installing Jupyter notebooks in the pre-lecture exercise for Lecture 2.

(1) $2\sqrt{n}+6 = O(\sqrt{n})$

$\exists\, c, n_0 > 0$; s.t. $\forall n > n_0$, $2\sqrt{n}+6 \le c\sqrt{n}$

so we take $c=3$. $\Rightarrow 3\sqrt{n} \ge 6+2\sqrt{n} \Rightarrow \sqrt{n} \ge 6 \Rightarrow n \ge 36$

so we take $n_0 = 36$ $\Rightarrow$ when $n > 36$, $2\sqrt{n}+6 \le 3\sqrt{n}$.

$\Rightarrow$ Thus, $2\sqrt{n}+6 = O(\sqrt{n})$

(2) $n^2 = \Omega(n)$

$\exists\, c, n_0 > 0$, s.t $\forall n > n_0$, $n^2 \ge cn$

$\Rightarrow$ Let $c=1$ $\Rightarrow n^2 \ge n \Rightarrow n \ge 1$

$\Rightarrow$ so we take $n_0 = 1$ $\Rightarrow$ when $n > 1$, $n^2 > n > 0$

$\Rightarrow$ Thus, $n^2 = \Omega(n)$

(3) $\log_2(n) = \Theta(\ln n)$

$\exists\, c_1, c_2, n_0 > 0$. s.t. $\forall n > n_0$, $c_1 \ln(n) \le \log_2 n \le c_2 \ln(n)$

$\Rightarrow$ take $c_1 = 1$, $n_0 = 1$ $\Rightarrow \log_2 n \ge \ln(n) = \frac{\log_2 n}{\log_2 e} \Leftrightarrow 1 \ge \frac{1}{\log_2 e} \Leftrightarrow e \ge 2$ (V) (Trivially. when $n > 1$, $\log_2 n > 0$)

$\Rightarrow$ take $c_2 = 2$, $n_0 = 1$ $\Rightarrow \log_2 n \le 2\ln(n) = 2 \cdot \frac{\log_2 n}{\log_2 e} \Leftrightarrow \log_2 e \le 2 \Leftrightarrow e \le 2^2 = 4$ (V)

$\Rightarrow$ Thus $\log_2(n) = \Theta(\ln(n))$

(4) $4^n \neq O(2^n)$

Assume, $4^n = O(2^n)$

so, $\exists\, c, n_0 > 0$, $\forall n > n_0$, $4^n \le c2^n$

$\Rightarrow$ If $4^n \le c2^n \Leftrightarrow c \ge 2^n$ $\Rightarrow$ when $n > \log_2 c$, $c < 2^n$, so $4^n > c \cdot 2^n$

$\Rightarrow$ This assume is not correct. $\Rightarrow 4^n \neq O(2^n)$

(a) √, no matter what the base is:

x can be interpreted as the form $x = x_1 \cdot b^{n/2} + x_0$, so as $y = y_1 \cdot b^{n/2} + y_0$

$\Rightarrow$ so $xy = x_1y_1 b^n + (x_1y_0 + x_0y_1) b^{n/2} + x_0y_0$

(b) √, the length can be expressed as $\log_b(x)+1$, so b ↗, the length ↙.

(c) X, though we express x in length n_b based b. the operation is $O(1)$.

but K's Algorithm takes $O(n^{1.6})$ time, but the wrong step is $n \neq O(1)$, because after the $O(1)$. n is variable.

So K's Algorithm still takes $O(n^{1.6})$ time. If we take $b = \max\{x, y\}$, K's Algorithm doesn't do anything at all !!

Problems

You may talk with your fellow CS161-ers about the problems. However:

- Try the problems on your own *before* collaborating.
- Write up your answers yourself, in your own words. You should never share your typed-up solutions with your collaborators.
- If you collaborated, list the names of the students you collaborated with at the beginning of each problem.

---

1. **(4 pt.)** In class we discussed Karatsuba's algorithm for n-digit integers written in base 10. That is, for an integer x, we wrote $x = \sum_{i=0}^{n-1} x_i 10^i$, for $x_i \in \{0, \ldots, 9\}$. But we can also consider an n-bit integer y written in base 2: $y = \sum_{i=0}^{n-1} y_i 2^i$, for $y_i \in \{0, 1\}$. Or we can think about an n-hexadecimal integer z written in base 16: $z = \sum_{i=0}^{n-1} z_i 16^i$, for $z_i \in \{0, \ldots, 15\}$.[2]

   Your friend has come up with the following argument that integer multiplication can be done in $O(1)$ time. The argument has three parts:

   (a) Whatever base we choose to write the numbers out in, Karatsuba's algorithm correctly finds the product of those numbers. For example, if we wanted to multiply the numbers 11010011 and 01011010 (which are written in binary), we could do that by recursively performing three multiplications involving the numbers $1101, 0011, 0101$, and 1010.

   (b) For a given number x, the length of x's base-b representation is decreasing as b increases. For example, the same number $x = 1024$ (base 10) can be written as
   - 1000000000 base 2 (10 bits)
   - 1024 base 10 (4 digits)
   - 400 base 16 (3 hexits)

   (c) Suppose we want to multiply two numbers x and y. Part (b) means that there's some large-enough b so that the base-b representations of x and y have length $n = O(1)$. Then we run Karatsuba's algorithm in this base (which works by part (a)), and it takes time $O(n^{1.6}) = O(1)$ because $n = O(1)$. Therefore we can multiply any two integers in time $O(1)$.

   Unfortunately (from the perspective of fast integer multiplication) your friend's argument is flawed in at least one place. Which of their steps are faulty and why?

   [We are expecting: For each of (a), (b), and (c), either assert that your friend's logic is correct, or give a brief argument about why it is wrong. You do not need to give a formal proof in either direction.]

[2] You may be used to representing number in hex on a computer, where it doesn't use symbols $0, \ldots, 15$ but rather $0, \ldots, 9$, along with the symbols A, B, C, D, E, F. In fact, this is the same thing, we just read "A" as 10, "B" as 11, and so on. So in hex, $1AF = 1 \cdot 16^2 + 10 \cdot 16^1 + 15 \cdot 16^0 = 431$ base 10.

2. **(10 pt.)** On an island, there are trustworthy toads and tricky toads. The trustworthy toads always tell the truth; the tricky toads may lie or may tell the truth. The toads themselves can tell who is tricky and who is trustworthy, but an outsider can't tell the difference: they all just look like toads.

You arrive on this island, and are tasked with finding the trustworthy toads. You are allowed to pair up the toads and have them evaluate each other. For example, if Tiffany the Toad and Tomás the Toad are both Trustworthy Toads, then they will both say that the other is trustworthy. But if Tiffany the Toad is a Trustworthy Toad and Tyrannus the Toad is a Tricky Toad, then Tiffany will call Tyrannus out as tricky, but Tyrannus may say either that Tiffany is tricky or that she is trustworthy. We will refer to one of these interactions as a "toad-to-toad comparison." The outcomes of comparing toads A and B are as follows:

Toad A	Toad B	A says (about B)	B says (about A)
Trustworthy	Trustworthy	Trustworthy	Trustworthy
Trustworthy	Tricky	Tricky	Either
Tricky	Trustworthy	Either	Tricky
Tricky	Tricky	Either	Either

Suppose that there are n toads on the island, and that there are strictly more than $n/2$ trustworthy toads.

In this problem, you will develop an algorithm to find all of the trustworthy toads, that only uses $O(n)$ toad-to-toad comparisons. Before you start this problem, think about how you might do this—hopefully it's not at all obvious! Along the way, you will also practice some of the skills that we've seen in Week 1. You will design two algorithms, formally prove that one is correct using a proof by induction, and you will formally analyze the running time of a recursive algorithm.

(a) **(1 pt.)** Give a straightforward algorithm that uses $O(n^2)$ toad-to-toad comparisons and identifies all of the trustworthy toads.

[We are expecting: A description of the procedure (either in pseudocode or very clear English), with a brief explanation of what it is doing and why it works.]

(b) **(3 pt.)*** Now let's start designing an improved algorithm. The following procedure will be a building block in our algorithm—make sure you read the requirements carefully!

Suppose that n is even. Show that, using only $n/2$ toad-to-toad comparisons, you can reduce the problem to the same problem with less than half the size. That is, give a procedure that does the following:

- **Input:** A population of n toads, where n is even, so that there are strictly more than $n/2$ trustworthy toads in the population.
- **Output:** A population of m toads, for $0 < m \leq n/2$, so that there are strictly more than $m/2$ trustworthy toads in the population.
- **Constraint:** The number of toad-to-toad comparisons is no more than $n/2$.

[We are expecting: A description of this procedure (either in pseudocode or very clear English), and rigorous argument that it satisfies the Input, Output, and Constraint requirements above.]

*This is the trickiest part of the problem set! You may have to think a while.

(c) **(1 bonus pt.) [This problem is NOT REQUIRED, but you may assume it for future parts.]** Extend your argument for odd n. That is, given a procedure that does the following:

- **Input:** A population of n toads, where n is odd, so that there are strictly more than $n/2$ trustworthy toads in the population.
- **Output:** A population of m toads, for $0 < m \leq \lceil n/2 \rceil$, so that there are strictly more than $m/2$ trustworthy toads in the population.
- **Constraint:** The number of toad-to-toad comparisons is no more than $\lfloor n/2 \rfloor$.

($\star$) *For all of the following parts, you may assume that the procedures in parts (b) and (c) exist even if you have not done those parts.*

(d) **(1 pt.)** Using the procedures from parts (b) and (c), design a recursive algorithm that uses $O(n)$ toad-to-toad comparisons and finds a *single* trustworthy toad.

[We are expecting: A description of the procedure (either in pseudocode or very clear English).]

(e) **(2 pt.)** Prove formally, using induction, that your answer to part (d) is correct.

[We are expecting: A formal argument by induction. Make sure you explicitly state the inductive hypothesis, base case, inductive step, and conclusion.]

(f) **(2 pt.)** Prove that the running time of your procedure in part (d) uses $O(n)$ toad-to-toad comparisons.

[We are expecting: A formal argument. Note: do this argument "from scratch," do not use the Master Theorem.]

(g) **(1 pt.)** Give a procedure to find *all* trustworthy toads using $O(n)$ toad-to-toad comparisons.

[We are expecting: An informal description of the procedure.]

(a) Trustworthy (toads):

```
for each toad T in the set of toads:
    cnt = 0
    for each toad T' other than T:
        compare (T', T)
        If T' says T is trustworthy:
            cnt += 1
    If cnt ≥ n/2:
        return T
```

Situation 1: If T is trustworthy, and we remove T.

The # of the rest trustworthy toads $\geq n/2$

The # of the rest tricky toads $\leq n-1-n/2 = n/2-1$

so # trustworthy > # tricky $\Rightarrow$ we can verify T is trustworthy.

Situation 2: If T is tricky, and we also remove T.

The # of the rest trustworthy toads $> n/2$

The # of the rest tricky toads $< n-1-n/2 = n/2-1$

so # trustworthy > # tricky $\Rightarrow$ # trustworthy win the vote again that T is tricky.

(d) Trustworthy (toads):

```
If n = 1:
    return the only toad.
If n is even:
    return Trustworthy (PartB(toads)).
If n is odd:
    return Trustworthy (PartC(toads))
```

(e) **Inductive Hypothesis:**

For all $m \leq n$, if toads contains m toads with more than m/2 trustworthy ones, our function will return a single trustworthy toad in toads.

Base Case:

when n=1, then m=1, then this single toad must be trustworthy.

Inductive Step: ??? see (b) & (c)

Conclusion:

By the induction, we conclude that the inductive is true.

(b) reduceByHalfForEvenN(toads):

Divide the n toads into n/2 pairs, $\{T_0, T_1\}, \{T_2, T_3\}, \cdots, \{T_{(n-2)}, T_{(n-1)}\}$

Ask each pair to evaluate each other. compare part

Let S be the set of pairs where both toads said "trustworthy"

Initialize subToads = { }

for $\{T_i, T_{(i+1)}\}$ in S:

    add T_i to subToads

return subToads.

This algorithm would be true.

Because the # of elements $\leq n/2$ and ≥ 1 (There are $> n/2$ trustworthy toads).

Suppose there are m pairs, $S = \{P_1, P_2, \cdots, P_m\}$ so that each toad in the pair said that their partner was trustworthy. And there were $n/2 - m$ pairs, $T = \{Q_1, \cdots, Q_{n/2-m}\}$ that the toads said different things or both their partner was tricky.

If there were $\leq m/2$ (Trust, Trust) pairs in S.

trustworthy $\leq 2 \cdot$ # (Trust, Trust) pairs in S + $|T| \leq 2 \cdot m/2 + n/2 - m) = n/2$ contradicted with $> n/2$

Thus, there are $> m/2$ (Trust, Trust) pairs in S. This completes the argument.

(c) reduceByHalfForOddN(toads):

oddToadOut = toads[0]

evenToads = all the toads except oddToadOut.

subToads = reduceByHalfForEvenN(evenToads)

If len(subToads) is even:

    add oddToadOut to subToats.

return subToads.

Analyze:

Case 1: subToods's length is even:

(a) # trustworthy − # tricky ≥ 2 (# trust > # tricky)

(b) # trustworthy = # tricky. ??? The oddToadOut must be trustworthy or there won't have been a strict majority to begin with.

(c) # trustworthy < # tricky can't happen.

Case 2: subToods's length is odd:

(a) # trustworthy > # tricky

(b) # trustworthy $\leq$ # tricky can't happen.

(f) First problem size is $n_0 = n$

second problem size is $n_1 \le \lceil n_0/2 \rceil \le n/2 + 1$

Third problem size is $n_2 \le \lceil n_1/2 \rceil \le n_1/2 + 1$

$\vdots$

$(i+1)$th problem size is $n_i \le \lceil n_{i-1}/2 \rceil \le n_{i-1}/2 + 1$

$\Rightarrow n_i - 2 \le \frac{n_{i-1}-2}{2} \le \frac{n_{i-2}-2}{2^2} \le \frac{n_0-2}{2^i} \Rightarrow n_i \le \frac{n_0}{2^i} + 2 - \frac{1}{2^{i-1}} \le \frac{n}{2^i} + 1$

$\Rightarrow \sum \text{compare} \le \sum_0^k n_i/2 \le \frac{1}{2} \sum_{i=0}^k (n/2^i + 1) \le n/2 \sum_{i=0}^k \frac{1}{2^i} + k/2 \le n + \frac{\log n}{2} = O(n)$

(g) First, use $O(n)$ comparisons to find a single trustworthy toad.

Then, use this trustworthy to compare the remaining toads are tricky or not.