CS 212 **Lab2**
Spring 2021 **Due:** See Deadline on Eduko.

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.)** 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 $T(x) = O(g(x))$.]

1. **(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.]

2. **(4 pt.)** Using the definition of big-Oh, formally prove the following statements.
   (a) $2\sqrt{n} + \sqrt{6} = O(\sqrt{n})$ (Note that you gave a "proof-by-picture" of this in Exercise 1). $2\sqrt{n} + 6 = O(\sqrt{n})$
   (b) $n^2 = \Omega(n)$
   (c) (c) $\log_2(n) = \Theta(\ln(n))$
   (d) (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 Lab1.

Problems

You may talk with your fellow CS212-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 $z_i \in \{0,\ldots,15\}$.[1]

   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.]

   [We are also expecting its Python code for base 2 and 16 multiplication]

2. **New friends.** Each of n users spends some time on a social media site. For each $i = 1, \ldots, n$, user i enters the site at time a_i and leaves at time $b_i \geq a_i$. You are interested in the question: how many distinct pairs of users are on the site at the same time? (Here, the pair (i, j) is the same as the pair (j, i)).

[1] You may be used to representing number in hex on a computer, where it doesn't use symbols 0,...,15 but rather 0,...,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.

Example: Suppose there are 5 users with the following entering and leaving times:

User	Enter Time	Leave Time
1	1	4
2	2	5
3	7	9
4	9	10
5	6	10

Then, the number of distinct pairs of users who are on the site at the same time is four: these pairs are (1, 2), (3, 4), (4, 5), (3, 5).

Note: If the Leave Time of one user is the same as the Enter Time of another, this counts as an overlap. For example, user 3's Leave Time is 9, and User 4's Enter Time is 9, and this counts as an overlap.

(a) (3 pt.) Given input (a_1, b_1), (a_2, b_2), ..., (a_n, b_n) as above, there is a straightforward algorithm that takes about n^2 time to compute the number of pairs of users who are on the site at the same time. Give this algorithm and explain why it takes time about n^2.

[We are expecting: Pseudocode for your algorithm, a clear English description of what your algorithm is doing and why it is correct, and a brief runtime analysis. You do not need to prove that your algorithm is correct.]

[We are expecting Python code]

(b) (5 pt.) Give an $O(n \log(n))$-time algorithm to do the same task and analyze its running time. (Hint: consider sorting relevant events by time).

[We are expecting: Pseudocode for your algorithm, a clear English description of what your algorithm is doing and why it is correct, and a brief runtime analysis. You do not need to prove that your algorithm is correct.]

[We are expecting Python code]

3. **(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 Tom´as 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.]

[We are also expecting its Python code]

(b) **(3 pt.)**[1] 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.]

[We are also expecting its Python code]

(c) **(1 bonus pt.)** 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 $\lfloor n/2 \rfloor$.

(*?*) *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).]

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

(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.]