CS261L Data Structures and Algorithms (Pr)

Lab Manual (Week 3)

Instructor:

- Mr. Samyan Qayyum Wahla

Registration No.__________________

Name: _________________________

Guide Lines/Instructions:

You may talk with your fellow CS261-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.

Today's Task:

- Divide and conquer paradigm

Problem 1: Meet Multiplication

Note: All the functions go to Karatsuba.py

i Multiply two integers a and b, using partial products and carry and show the time consumed with the increased number of digits.
Multiply_integer(a,b) – a, b are integers, return type is string containing answer.

ii Repeat part(i) to take input a and b as string.
Multiply_string(a,b) – a, b are strings, return type is string containing answer.

iii Visualize the multiplication process in the following format.

```
   2 2          1 2 2
   4 5          2 4 5
   ----         ------
 1 1 0          6 1 0
 8 8 0        4 8 8 0
 ------     2 4 4 0 0
 9 9 0      ----------
 ------     2 9 8 9 0
            ----------
```

Visualize_Karatsuba(a, b) – a and b are string, do not return any thing just print the output.

iv Write the recursive code of multiplication and show the time consumed with the increasing number of digit.
Multiply_Recursive(a, b) – a and b are string, return string containing answer.

v Implement Karatsuba Algorithm.
Karatsuba_Recursive(a, b) – a and b are string, return string containing answer.

vi 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 $x_i \in \{0,...,9\}$. 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 $z_i \in \{0,...,15\}$.[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:

[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.

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]

write the following functions

- `Multiply2(x,y)—return a string array as output. x and y are also string.` Out of the function, take input from the user and prints the answer.
- `Multiply16(x,y)—return a string array as output  x and y are also string.` Out of the function, take input from the user and prints the answer.

Also validate if the user has entered the currect input w.r.t base number 2 or 16. In case of invalid input return empty string.

Problem 2: Social Friends

Note: All the functions go to Friends.py

Each of *n* users spends some time on a social media site. For each *i* = 1, ..., *n*, user *i* enters the site at time a_i and leaves at time $b_i \geq a_i$. You are interested ingo 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*)).

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) 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.(Write here)]

[We are expecting Python code]

In Friends.py, create a function

`friendSlower(Input)`

- Input represents 2D arrays such as above table can be represented as follow
  `A= [[1,4],[2,5],[7,9],[9,10],[6,10]]`
- return output as list of tuples as list [(1, 2), (3, 4), (4, 5), (3, 5)]

(b) 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]

In Friends.py, create a function

`friendsFaster(Input)`

- Input represents 2D arrays such as above table can be represented as follow
  `A= [[1,4],[2,5],[7,9],[9,10],[6,10]]`
- return output as list of tuples as list [(1, 2), (3, 4), (4, 5), (3, 5)]
- You can create helping functions in same file if required.

Problem 3: Toads

Note: All the functions go to Toads.py

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) toadto-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.

Use following class of Toad.

```
import random
class Toad:
    def __init__(self, is_trustworthy):
        self.truthful: bool = bool(int(is_trustworthy))

    def is_trustworthy(self):
        return self.truthful


    def tell_about(self,toad):
        b_trustworthy= toad.is_trustworthy()
        if(self.is_trustworthy()):
            return b_trustworthy
        else:
            r = random.random()
            if(r<0.5):
                return True
            else:
                return False
```

(a) 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]

Create functions

`getPopulation(n)-- n is number of total toads on island and returns the list of toads meeting above criteria.`

`TruthFulToadsA(population) – Population is list of toads and return list integers containing the indices of truthful toads`

(b) 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]

`TruthFulToadsB(population) – Population is list of toads and return list integers containing the indices of truthful toads`

(c) 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 $0 < m \leq$d$n/2$e, 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 b$n/2$c.

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

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

(e) 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) 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) Give a procedure to find *all* trustworthy toads using $O(n)$ toad-to-toad comparisons.
[We are expecting: An informal description of the procedure.]

What to Submit:

1. Only .py files are allowed.
2. You are required to submit the following files.
    a. karatsuba.py
    b. friends.py
    c. toads.py
    d. Lab3.docx with all the descriptive answers typed. (you may take snapshot of handwritten work to paste in the file, but text should be typed)
3. Functions names, input and output should be exactly same.
4. Zip all files, and submit on eduko