CSE 20, Fall 2020 - Homework 1

Due: Monday 10/12 at 11am PDT

Instructions

Upload a single file to Gradescope for each group. All group members' names and PIDs should be on each page of the submission. Your assignments in this class will be evaluated not only on the correctness of your answers, but on your ability to present your ideas clearly and logically. You should always explain how you arrived at your conclusions, using mathematically sound reasoning. Whether you use formal proof techniques or write a more informal argument for why something is true, your answers should always be well-supported. Your goal should be to convince the reader that your results and methods are sound.

Reading Rosen Sections 2.1, 4.1, 4.2 and 5.3

Key Concepts Set Definitions, Division Algorithm, Algorithms, Tracing Pseudocodes for Different Inputs, Base Expansion and Conversions

Problem 1 (20 points)

In this question we think about recommendation systems. A key concept about recommendation system is clustering, which refers to the process of grouping data with similar patterns together. For example, each YouTube user's watching/liking/disliking history can be represented as an n-tuple indicating their tastes of videos. Users with similar tastes can then be clustered to provide future recommendation for one another. Mathematically, clustering is based on the notation of distance between pairs of n-tuples.

Consider n = 5. Define the following functions whose inputs are ordered pairs of 5-tuples whose components come from the set {0, 1, 2}.

$$d_{max,5}((x_1, x_2, x_3, x_4, x_5), (y_1, y_2, y_3, y_4, y_5)) = \max_{1 \le i \le 5} |x_i - y_i|$$

$$d_{rmse,5}((x_1, x_2, x_3, x_4, x_5), (y_1, y_2, y_3, y_4, y_5)) = \sqrt{\frac{\sum_{i=1}^{5} (x_i - y_i)^2}{5}}$$

(a) Give a recursive definition of the max function whose input is a sequence of integers and whose output is the maximum value in the sequence. Include a description of the domain and codomain of the function, along with the basis step and the recursive step of the function definition.

(b) Repeat (a) for the RMSE function.

(c) Provide possible examples input for which the function has a preset value.

(i) Give an example input to $d_{max,5}$ for which the output of the function is 0. Is your example the only possible example? Why?

(ii) Give an example input to $d_{max,5}$ for which the output of the function is 1. Is your example the only possible example? Why?

(iii) Give an example input to $d_{rmse,5}$ for which the output of the function is 0. Is your example the only possible example? Why?

(iv) Give an example input to $d_{rmse,5}$ for which the output of the function is 2. Is your example the only possible example? Why?

Solution:

(a) As we previously posted on Piazza, there are 2 different ways to understand what this problem asks. The first interpretation is to give a definition of the normal max function that we use most frequently, i.e., $\max(x_1, ..., x_n)$ returns the maximum value among $x_1, ..., x_n$. The second interpretation is to give a definition of the max function defined in this problem, i.e., $d_{max,n}$. When grading, both interpretations are acceptable. For simplicity, we only provide the definition for the first interpretation in this solution. The solution for the second interpretation is very similar.

The domain of the max function is the collection of all finite sequences whose elements are integers and the codomain of the max function is the set of integers. To define the function, the only piece remaining is the rule, which we define recursively. The recursion is on the length of the sequence of which we are computing the max value:

Basis Step: If the input sequence has length 1, $max(a_1) = a_1$

Recursive Step: If the input sequence has length $n+1$ then

If $a_{n+1} > \max_{1 \le i \le n} a_i$, then $\max_{1 \le i \le n+1}(a_i) = a_{n+1}$

Otherwise, $\max_{1 \le i \le n+1}(a_i) = \max_{1 \le i \le n}(a_i)$

(b) For simplicity, we denote $x = (x_1, ..., x_n)$, $y = (y_1, ..., y_n)$

Basis Step: If the input sequence has length 1, $d_{rmse}(x, y) = |x_1 - y_1|$

Recursive Step: If the input sequence has length $n+1$ then

$$d_{rmse,n+1}(x,y) = \sqrt{\frac{n * d_{rmse,n}(x,y)^2 + (x_{n+1} - y_{n+1})^2}{n+1}}$$

(c)

(i) $x = y = (0, 0, 0, 0, 0)$
This is not the only possible answer. For example, $x = y = (0, 0, 0, 0, 1)$ also yields the same output.

(ii) $x = (0, 0, 0, 0, 0)$, $y = (1, 1, 1, 1, 1)$
This is not the only possible answer. For example, $x = (0, 0, 0, 0, 0)$, $y = (0, 1, 1, 1, 1)$ also yields the same output.

(iii) $x = y = (0, 0, 0, 0, 0)$
This is not the only possible answer. For example, $x = (0, 0, 0, 0, 1)$, $y = (0, 0, 0, 0, 1)$ also yields the same output.

(iv) $x = (0, 0, 0, 0, 0)$, $y = (2, 2, 2, 2, 2)$
This is not the only possible answer. For example, $y = (0, 0, 0, 0, 0)$, $x = (2, 2, 2, 2, 2)$ also yields the same output.

Problem 2 (20 points)

DNA is made up of strands of four different bases that match up in specific ways. The bases are elements of the set B = {A, C, G, T}.

Definition The set of DNA strands S is defined (recursively) by:

Basis Step: $A \in S,\ C \in S,\ G \in S,\ T \in S$

Recursive Step: If $s \in S$ and $b \in B$, then $sb \in S$

A function $dnalen$ that computes the length of DNA strands in S is defined by:

$$rnalen : S \rightarrow Z^+$$

Basis Step: If $b \in B$ then $dnalen(b) = 1$

Recursive Step: If $s \in S$ and $b \in B$, then $dnalen(sb) = 1 + dnalen(s)$

Each of the sets below is described using set builder notation. Rewrite them using the roster method. For example, the set described in set builder notation as

$\{s \in S \mid$ the leftmost base in s is T and $dnalen(s) = 2\}$

is described using the roster method by

$$\{TA, TC, TG, TT\}$$

Please solve the 2 problems below:

(a) $\{s \in S \mid$ the leftmost base in s is the same as the rightmost base in s, the second-from-left base of s is not A and is not C, and $dnalen(s) = 4\}$

(b) $\{s \in S \mid s$ has at most one T and there are twice as many As as Gs in s, and $dnalen(s) = 4\}$

Solution:

(c) {AGAA, AGCA, AGGA, AGTA, ATAA, ATCA, ATGA, ATTA, CGAC, CGCC, CGGC, CGTC, CTAC, CTCC, CTGC, CTTC, GGAG, GGCG, GGGG, GGTG, GTAG, GTCG, GTGG, GTTG, TGAT, TGCT, TGGT, TGTT, TTAT, TTCT, TTGT, TTTT}

You can run the following python scripts to generate the above result:

```
d = ['A', 'C', 'G', 'T']
s = []
for x in d:
   for y in d:
       for z in d:
           ans = ['\n'] * 4
           ans[0] = x
           ans[1] = y
           ans[2] = z
           ans[3] = x
           if y != 'A' and y != 'C':
               s.append(''.join(ans))

print(s)
```

(d) There're 29 strands of DNA that satisfy the above requirements:
{AACG, AAGC, AAGT, AATG, ACAG, ACGA, AGAC, AGAT, AGCA, AGTA, ATAG, ATGA, CAAG, CAGA, CCCC, CCCT, CCTC, CGAA, CTCC, GAAC, GAAT, GACA, GATA, GCAA, GTAA, TAAG, TAGA, TCCC, TGAA}

You can run the following python scripts to generate the above result:

```
d = ['A', 'C', 'G', 'T']
s = []
for x in d:
   for y in d:
       for z in d:
           for w in d:
               ans = ['\n'] * 4
               ans[0] = x
               ans[1] = y
               ans[2] = z
               ans[3] = w
               if ans.count('T') <= 1 and ans.count('A') == ans.count('G') * 2:
                   s.append(''.join(ans))

print(len(s))
print(s)
```

Problem 3 (20 points)

We focus on **The Division Algorithm** (Rosen 4.1 Theorem 2, p. 239) in this problem:

Let n be an integer and d a positive integer. There are unique integers q and r, with $0 \leq r < d$, such that $n = dq + r$. In this case, d is called the divisor, n is called the dividend, q is called the quotient, and r is called the remainder. We write $q = n \, div \, d$ and $r = n \, mod \, d$.

In display industry, any color can be mathematically represented as a 3-tuple (r, g, b) where r, g, b represent the red, green and blue component respectively. Each of (r, g, b) must be from the collection $\{x \in N \mid 0 \leq x \leq 255\}$.

(a) Convert the number expressed in base 16 as $(FF)_{16}$ to binary, octal and decimal. Justify your answer in a clear way that anyone who has just taken the course can easily follow. What is the color (x, x, x) where x = $(80)_{16}$? You can use a web color tool (include the URL of the tool you use as part of your assignment write-up) .

(b) Suppose you were told that the colors that will work best for your web app are (r, g, b) where $r \, mod \, 16 = 6$ and $g \, div \, 16 = 7$ and $b \, mod \, 16 = 8$. Give three distinct examples of such colors. For each example, specify its red, green, and blue components both in decimal and in hexadecimal. Justify your answer in a clear way that anyone who has just taken the course can easily follow.

Solution:

(a) $(FF)_{16} = (11111111)_2 = (377)_8 = (255)_{10}$

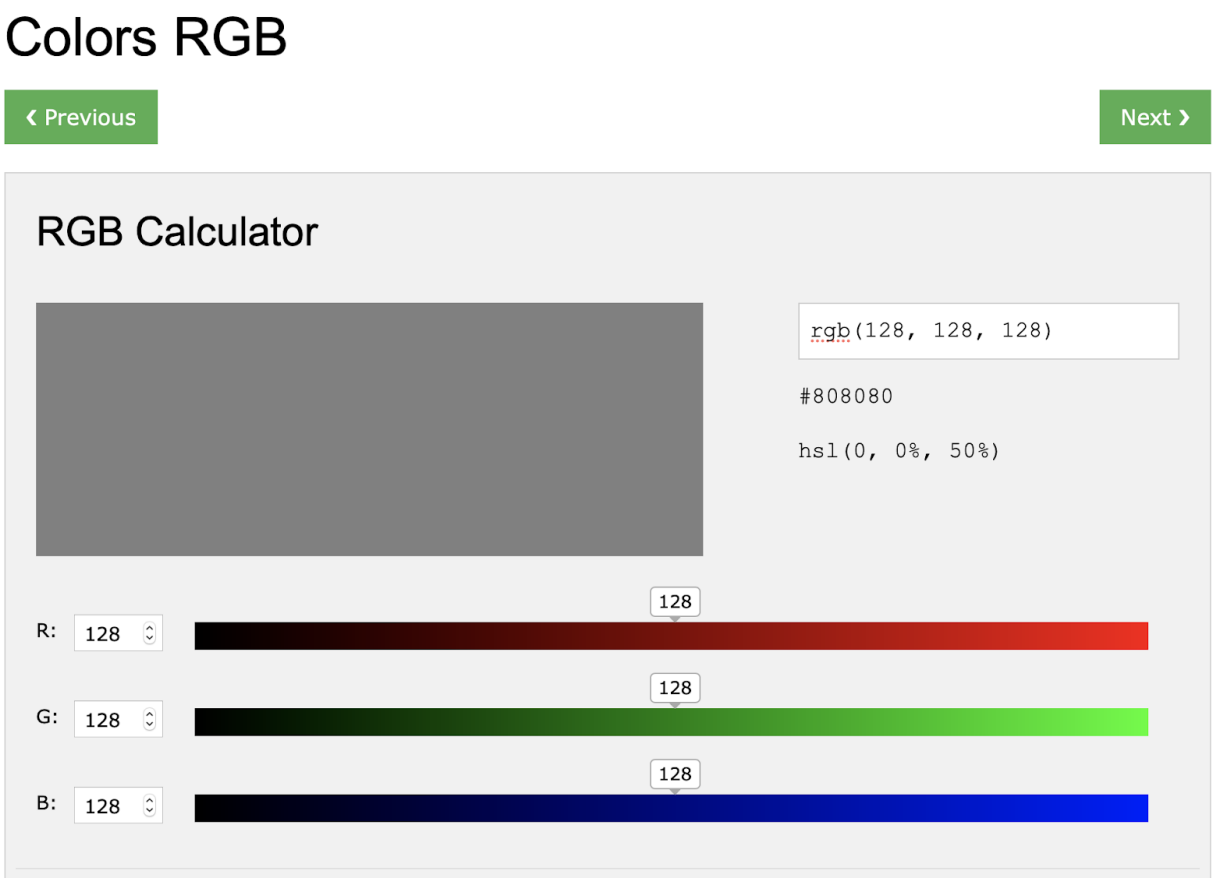
Justification:

i. $(FF)_{16} = 15 \times 16^1 + 15 \times 16^0 = 255$

ii. $(11111111)_2 = \sum_{i=0}^{7} 1 \times 2^i = 255$

iii. $(377)_8 = 3 \times 8^2 + 7 \times 8^1 + 7 \times 8^0 = 255$

The color is gray. You can easily look this color up at this website:
https://www.w3schools.com/colors/colors_rgb.asp
A screenshot is attached below:



(b) We provide 3 distinct examples for (r,g,b) below:

i. $(6, 112, 8)_{10} = (6, 70, 8)_{16}$

ii. $(86, 120, 88)_{10} = (56, 78, 58)_{16}$

iii. $(134, 120, 136)_{10} = (86, 78, 88)_{16}$

The justification takes the same form as that one used in problem 3(a).

Problem 4 (20 points)

Consider the following pseudocode:

1. **procedure** *f1* (m : real number; n : positive integer)
2. $total := 0$
3. $a := m$
4. $b := n$
5. **while** $b > 1$
6. **if** (b **mod** $2 = 1$) **then** $total := total + a$
7. $a := 2 \cdot a$
8. $b := b$ **div** 2
9. **return** $total + a$

(a) Trace through an example of this pseudocode when $m = 312$ and $n = 32$. It might be helpful to create a table showing the value of each variable after each step. What does this algorithm do?

(b) Repeat part(a) for $m = 32$ and $n = 512$. What is the difference between this and (a)? Given 2 integers x and y, can you come up with a general principle for choosing which number to pick as m and which number to pick as n?

(c) Convert $m = 312$ and $n = 32$ to binary.

(d) Multiply the binary representation of m and n using the standard long-form multiplication technique that you use for decimal multiplication.

(e) Convert your answer from (d) to decimal and compare it with your answer from (a). Intuitively, how do you think this algorithm corresponds to the long multiplication algorithm?

Solution:

a)

Step	m	n	$total$	a	b	Condition
1	312	32	0	312	32	$32 > 1$: **True** $32 \ mod \ 2 = 1$: **False**
2			0	624	16	$16 > 1$: **True** $16 \ mod \ 2 = 1$: **False**
3			0	1248	8	$8 > 1$: **True** $8 \ mod \ 2 = 1$: **False**
4			0	2496	4	$4 > 1$: **True** $4 \ mod \ 2 = 1$: **False**
5			0	4992	2	$2 > 1$: **True** $2 \ mod \ 2 = 1$: **False**
6			0	9984	1	$1 > 1$: **False** **return** $total + a = 0 + 9984 = 9984$

At every step, this algorithm divides the value of *b* by 2 and multiplies the value of *a* by 2, until *b* becomes equal to 1: therefore, at every step, the product of *a* and *b* remains constant, and equal to the product of the initial inputs m and n. At the end, since $b = 1$,

the product $m.n$ is equal to the value of a. Therefore, this is a multiplication algorithm for m and n!
This algorithm works on the principle of binary multiplication: Essentially, what we are doing here is representing n as a binary number:

$$n = 32 = 1.2^5$$

Since n is 2^5, we multiply a by 2 5 times!
This technique would work even when n is not a power of 2! This is where we check for $b \; mod \; 2$. Let's say we had $n = 6 = 2^2 + 2^1$
Then: we would have $a(2^2 + 2^1)$. At the first step, we would divide $(2^2 + 2^1) \; by \; 2$: this yields $2^1 + 1$, leaving no remainder: therefore we do not update total and simply double a. However, when we next divide $b = 2^1 + 1$ by 2, we will get a remainder of 1: this is because now if we want to multiply $a(2^1 + 1) = 2^1 a \; + \; a$: therefore we add a to the total and continue with our algorithm.

b)

Step	m	n	$total$	a	b	Condition
1	32	512	0	32	512	$512 > 1$: **True** $512 \; mod \; 2 \; = 1$: **False**
2			0	64	256	$256 > 1$: **True** $256 \; mod \; 2 \; = 1$: **False**
3			0	128	128	$128 > 1$: **True** $128 \; mod \; 2 = 1$: **False**
4			0	256	64	$64 > 1$: **True** $64 \; mod \; 2 = 1$: **False**
5			0	512	32	$32 > 1$: **True** $32 \; mod \; 2 = 1$: **False**
6			0	1024	16	$16 > 1$: **True** $16 \; mod \; 2 = 1$: **False**
7			0	2048	8	$8 > 1$: **True** $8 \; mod \; 2 = 1$: **False**
8			0	4096	4	$4 > 1$: **True** $4 \; mod \; 2 = 1$: **False**
9			0	8192	2	$2 > 1$: **True** $2 \; mod \; 2 = 1$: **False**
10			0	16384	1	$1 > 1$: **False** **return** $total + a = 0 + 16384 = 16384$

We observe that this calculation required more computation steps, although the final result was still the same. This is because we were dividing the larger number by 2 and

attempting to reach 1: therefore, given 2 integers x and y, we should set, m to be the larger of the 2 and n to be the smaller of the 2!

c) $m = 312 = 2^8 + 2^5 + 2^4 + 2^3 = (100111000)_2$
$n = 32 = 2^5 = (100000)_2$

d) The multiplication is:

$$
\begin{array}{r}
100111000 \\
\times \quad 100000 \\
\hline
000000 \\
000000x \\
000000xx \\
000000xxx \\
000000xxxx \\
+\ 100111000xxxxx \\
\hline
10011100000000
\end{array}
$$

Note: x just represents 0.
With the usual algorithm for long multiplication, we multiply $(100111000)_2$ by each bit of $(100000)_2$ in turn, and then add the shifted results. Since the product of 0 with any number is 0, we only get one nonzero term in the sum, and it is shifted six times. Thus, the product is $(10011100000000)_2$.

e) $(10011100000000)_2 = 2^{13} + 2^{10} + 2^9 + 2^8 = (9984)_{10}$

The bits in the binary expansion can be computed by repeatedly “taking mod" of the number being represented. If we consider the product of *m* and *n*, we notice that in (d), the product is described by adding together shifted versions of m for each *1* in the binary expansion of *n*: shifting a number to the left in binary is equivalent to multiplying by 2 in binary. This corresponds to the fact that when we divide *n* by *2* repeatedly in part (a), we add in the current value of *m* to the output exactly when *n* is odd, *i.e.* when *n mod 2* is *1* and so the current bit in the binary expansion is *1*.

Problem 5 (20 points)

In class, we have dealt with several examples of recursive definitions of sets. These will form a basis of several algorithms that you will use throughout this class and in the future. Recursive definitions involve 2 steps: the Basis Step and Recursive step.

(a) Let's consider a recursive definition for the set S_4 of positive integer multiples of 4.

Basis Step: If $x \in$ _____, then $x \in S_4$

Recursive Step: If $x \in S_4$ and $y \in S_4$, then _____ $\in S_4$

(b) Let's say that you now want to make a set Z_4 of all (positive and negative) multiples of 4. Modify the recursive definition given above that will allow you to do this.

(c) Similar to how we define a recursive set, we can also define a recursive algorithm for computing certain functions.
Let's try and construct a recursive algorithm to calculate the factorial of a number. The factorial (defined as the product of all integers from 1 to n) of a number of n is defined as:

$$n! = n \cdot (n-1) \cdot (n-2) \ldots 3 \cdot 2 \cdot 1$$

1. **procedure** $factorial$ (n : non-negative integer)
2. **if** _____ then **return** 1
3. **else return** _____

Solution:

(a) Basis Step: If $x \in$ **{4}**, then $x \in S_4$

Recursive Step: If $x \in S_4$ and $y \in S_4$, then **x+y** $\in S_4$

If you consider $0 \in S_4$, then you may have $x \in$ **{0,4}.** For this question, we restricted to only positive multiples of 4.

(b) Basis Step: If $x \in$ **{4}**, then $x \in Z_4$

Recursive Step: If $x \in Z_4$ and $y \in Z_4$, then **x-y** $\in Z_4$

OR

(c) Basis Step: If $x \in$ **{-4,4}**, then $x \in Z_4$

Recursive Step: If $x \in Z_4$ and $y \in Z_4$, then **x+y** $\in Z_4$

(d) procedure $factorial$ (n : non-negative integer)
if **n=0** then return 1
else return **n.factorial(n-1)**

Problem 6 (Bonus: 10 points)

The base-14 expansion of a number n has d digits. Find an upper bound on the number of bits in the binary expansion of n.

Solution:
Let $n = (a_{d-1}a_{d-2}....a_0)_{14}$, and let $n_0 = (a_{d-1}a_{d-2}....a_0)_{16}$ i.e. the same digits but interpreted as a hexadecimal number. Note that $n \leq n_0$ (because for every non-negative p, $15p \leq 16p$).
From Rosen p.249 we know that n_1 has at most $4d$ bits in its binary expansion. Since $n \leq n_0$, $4d$ is also an upper bound on the number of bits for n.

This is however, not the tightest possible upper bound. Remember, the largest number you can represent in base-*14* using *d* digits is $14^d - 1$. This is actually true for all bases- the largest number you can represent using *d* digits in base-*2* is $2^d - 1$, while the smallest number you can represent using *d* digits is 2^{d-1}. This means that the number of digits you need to represent a number between $2^d - 1$ and 2^{d-1} is *d*: note that this is essentially: $ceil(log_2(number) + 1)$

Therefore, the tightest possible upper bound to represent *n* is the ceiling of $log_2(14^d - 1)$ (the smallest integer larger than $log_2(14^d - 1)$).