Due at 11:59PM, Friday, March 7th, on Canvas. Submissions must be ***typed*** (except for questions labeled with a "📷" icon) and in PDF format. Show your work for full credit!

1. (📷) Use the ***method of recursion trees*** to solve the following recurrences:

   (a) $A(n) = 3\,A(n/3) + O(n^2)$.

   (b) $B(n) = 2\,B(n/4) + O(\sqrt{n})$.

   For each recursion tree, draw three layers (root, children, grandchildren) of the recursion tree. (If you don't want to draw, just describe the trees, e.g. "there are four children, each with label $n^3/9$,")

2. Use the ***master theorem*** to solve the following recurrences:

   (c) $C(n) = 4\,C(n/13) + \Theta(1)$

   (d) $D(n) = 9\,D(n/3) + \Theta(n^2)$

   (e) $E(n) = 19\,E(n/2) + \Theta(n^4)$

   (f) $F(n) = 2\,F(n/8) + \Theta(\sqrt[3]{n})$

   (g) $G(n) = 6\,G(n/7) + \Theta(n)$

3. Hindsight is 20/20 when it comes to investing in Pokemon cards. Suppose you're given an array $P[1, \ldots, n]$, where $P[i]$ is the price of a certain card on day i. The goal is to find the best days to buy and sell in order to maximize your profit. The return value of your algorithms should be the ***profit*** made, i.e., the difference between the selling price and the buying price.

   You may assume that you're only buying and selling one card, the price doesn't change during a given day, and that the card needs to be bought ***before*** it is sold. In other words, you are trying to maximize $P[j] - P[i]$ over all pairs of indices i and j, where $i \leq j$. For example, if the array of prices is

   $$[2, 4, 3, 5, 1],$$

   the maximum profit is 3, which is achieved when you buy on day 1 and sell on day 4. You can't make a profit of 4 because 1 comes after 5.

   For both parts below, write down an algorithm in ***pseudocode***.

   (a) Design a brute force algorithm for solving this problem. Your algorithm should be of the form:

   ```
   CardBrute(P[1,...,n]):
       // your code here
   ```

   In terms of big-O notation, how long does your algorithm take?

   (b) Use ***divide and conquer*** to design an algorithm that runs in time $O(n \log n)$ (Hint: see the maximum subarray problem from class). Your algorithm should be of the form:

   ```
   CardDnC(P[1,...,n]):
       // your code here
   ```

   Give a brief ***verbal explanation*** for why your algorithm works, and why it runs in time $O(n \log n)$. (Yes, there is a faster way of doing this problem.)

4. Recall that Karatsuba's algorithm is a divide-and-conquer algorithm for multiplying two n-digit numbers that splits each of the numbers in two parts. This problem is about a variant that splits each number into ***three*** parts. Let

$$X = 10^{2n/3} \cdot A + 10^{n/3} \cdot B + C \quad \text{and} \quad Y = 10^{2n/3} \cdot D + 10^{n/3} \cdot E + F,$$

where each of $A, \ldots, F$ are $(n/3)$-digit numbers. If we multiply X and Y together, we would get a number of the form

$$XY = w_4 \cdot 10^{4n/3} + w_3 \cdot 10^{3n/3} + w_2 \cdot 10^{2n/3} + w_1 \cdot 10^{n/3} + w_0$$

(a) By expanding out the above expressions for X and Y using the distributive property, write down what the missing numbers $w_4, \ldots, w_0$ are in terms of $A, \ldots, F$.

Like in Karatsuba's algorithm, there are ways to write each of w_3, w_2, w_1 in a more complicated way that actually results in fewer recursive calls. Consider the following values:

$$\begin{aligned} M_1 &= AD \\ M_2 &= (A + B + C)(D + E + F) \\ M_3 &= (A - B + C)(D - E + F) \\ M_4 &= (4A + 2B + C)(4D + 2E + F) \\ M_5 &= CF \end{aligned}$$

Then by calculating these values and storing the answers as temporary variables, we may reuse them several times to compute the unknown numbers $w_4, \ldots, w_0$:

$$\begin{aligned} w_4 &= M_1 \\ w_3 &= (-12M_1 - 3M_2 - M_3 + M_4 + 3M_5)/6 \\ w_2 &= (-2M_1 + M_2 + M_3 - 2M_5)/2 \\ w_1 &= (12M_1 + 6M_2 - 2M_3 - M_4 - 3M_5)/6 \\ w_0 &= M_5 \end{aligned}$$

(b) Verify that the equation for w_2 is correct by comparing it to your answer in part (a). Show your calculations! (Of course, we should also check the other equations, but one is enough for this homework)

The final algorithm computes $M_1, \ldots, M_5$ using recursive calls, and then finds $w_4, \ldots, w_0$ using the above identities.

(c) Assuming multiplying and dividing an n-digit number by a constant takes $O(n)$ time, ***write down*** a recurrence for $T(n)$, the amount of time this algorithm takes to multiply two n-digit numbers, and ***solve the recurrence*** using any method you like. In terms of big-O notation, how does this algorithm compare to Karatsuba's algorithm?