Name: YOUR NAME HERE

ID: YOUR STUDENT ID HERE

CSCI 3104, Algorithms **Due February 12, 2021**

Problem Set 4 (50 points) **Spring 2021, CU-Boulder**

Advice 1: For every problem in this class, you must justify your answer: show how you arrived at it and why it is correct. If there are assumptions you need to make along the way, state those clearly.

Advice 2: Verbal reasoning is typically insufficient for full credit. Instead, write a logical argument, in the style of a mathematical proof.

Instructions for submitting your solution:

- The solutions **should be typed** and we cannot accept hand-written solutions. Here's a short intro to Latex.
- You should submit your work through **Gradescope** only.
- The easiest way to access Gradescope is through our Canvas page. There is a Gradescope button in the left menu.
- Gradescope will only accept **.pdf** files.
- It is vital that you match each problem part with your work. Skip to 1:40 to just see the matching info.

Recall that a function f expressed in terms that depend on f itself is a recurrence relation. "Solving" such a recurrence relation means expressing f without terms that depend on f.

1. Solve the following recurrence relations using the **unrolling method** (also called plug-in or substitution method), and find tight bounds on their asymptotic growth rates. Remember to show your work so that the graders can verify that you used the **unrolling method**. Assume that all function input sizes are non-negative integers. You may also assume that integer rounding of any fraction of a problem size won't affect asymptotic behavior.

   (a) $U_a(n) = \begin{cases} 2U_a(n-1) - 1 & \text{when } n \geq 1, \\ 2 & \text{when } n = 0. \end{cases}$

   (b) $U_b(n) = \begin{cases} 3U_b(n/4) + n/2 & \text{when } n > 3, \\ 0 & \text{when } n = 3. \end{cases}$

   Solution:

2. Consider this recurrence:

$$T(n) = \begin{cases} 4T(n/3) + 2n & \text{when } n > 1, \\ 1 & \text{when } n = 1. \end{cases}$$

   (a) How many levels will the recurrence tree have?

   (b) What is the cost at the level below the root?

   (c) What is the cost at the ℓ'th level below the root?

   (d) Is the cost constant for each level?

   (e) Find the total cost for all levels. *Hint: You may need to use a summation. The Geometric Sum formula may be helpful.*

   (f) If $T(n)$ is $\Theta(g(n))$, find $g(n)$.

   Solution:

3. Showing your work for relevant comparisons, for the following recurrence relations apply the **master method** to identify whether original problems or subproblems dominate, or whether they are comparable. Then write down a Θ bound.

   (a) $M_a(n) = \begin{cases} 2M_a(n/3.14) + n\log(n) & \text{when } n > 0.001, \\ 1337 & \text{otherwise.} \end{cases}$

   (b) $M_b(n) = \begin{cases} 6M_b(n/2) + n^{7/3}\log(n) & \text{when } n > 2^{273}, \\ 6734 & \text{otherwise.} \end{cases}$

   (c) $M_c(n) = \begin{cases} 9M_c(n/3) + n^3\log(n) & \text{when } n > 8/3, \\ 86 & \text{otherwise.} \end{cases}$

   Solution:

4. This is a coding problem. You will implement a version of Quicksort.

   - **You must submit a Python 3 source code file with a `quicksort` and a `partitionInPlace` function as specified below.** You will not receive credit if we cannot call your functions.
   - The `quicksort` function should take as input an array (numpy array), and for large enough arrays pick a pivot value, call your partition function based on that pivot value, and then recursively call quicksort on resulting partitions that are strictly smaller in size than the input array in order to sort the input.
     - Additionally, your quicksort should transition from recursive calls to "manual" sorting (via `if` statements or equivalent) when the arrays become small enough.
   - The `partitionInPlace` function should take as input an array (numpy array) and pivot value, partition the array (*in at most linear amount of work and constant amount of space*), and return an index such that (after returning) no further swaps need to occur between elements below and elements above the index in order for the array to be sorted.
   - You are provided with a scaffold python file that you may use, which contains some suggested function behavior and loop invariants, as well as a simple testing driver. You may alter anything within or ignore it altogether **so long as you maintain the function prototypes specified above**.
     - In particular, the suggestions are meant to allow the pivot value to not be in the array, which is NOT a requirement for Quicksort.

   Solution: