Name: Michael Rogers

ID: 105667404

CSCI 3104, Algorithms **Profs. Hoenigman & Agrawal**

Problem Set 2b (51 points) **Fall 2019, CU-Boulder**

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.
- If you don't have an account on it, sign up for one using your CU email. You should have gotten an email to sign up. If your name based CU email doesn't work, try the identikey@colorado.edu version.
- Gradescope will only accept **.pdf** files (except for code files that should be submitted separately on Gradescope if a problem set has them) and **try to fit your work in the box provided**.
- You cannot submit a pdf which has less pages than what we provided you as Gradescope won't allow it.
- Verbal reasoning is typically insufficient for full credit. Instead, write a logical argument, in the style of a mathematical proof.
- 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.
- You may work with other students. However, **all solutions must be written independently and in your own words.** Referencing solutions of any sort is strictly prohibited. You must explicitly cite any sources, as well as any collaborators.

Name: Michael Rogers
ID: 105667404

CSCI 3104, Algorithms **Profs. Hoenigman & Agrawal**
Problem Set 2b (51 points) **Fall 2019, CU-Boulder**

1. *(4 pts) Using L'Hopital's Rule, show that* $\ln(n) \in \mathcal{O}(\sqrt{n})$.

   Solution. Using the Limit Comparison Test:

   $\lim_{n\to\infty} \frac{ln(n)}{\sqrt{n}} =^{\text{L.H}} \lim_{n\to\infty} \frac{\frac{1}{n}}{\frac{1}{2\sqrt{n}}} = \lim_{n\to\infty} \frac{2\sqrt{n}}{n} =^{\text{L.H}} 2 \lim_{n\to\infty} \frac{1}{2\sqrt{n}} = 0$

   $\therefore$ By the LCT, since $\lim_{n\to\infty} \frac{ln(n)}{\sqrt{n}} = 0$ then we can say $ln(n) \in \mathcal{O}(\sqrt{n})$

Name: Michael Rogers
ID: 105667404

2. *(6 pts) Let* $f(n) = (n-3)!$ *and* $g(n) = 3^{5n}$*. Determine which of the following relations* ***best*** *applies:* $f(n) \in \mathcal{O}(g(n))$*,* $f(n) \in \Omega(g(n))$*, or* $f(n) \in \Theta(g(n))$*. Clearly justify your answer. You may wish to refer to Michael's Calculus Review document on Canvas.*

   Solution.

Name: Michael Rogers
ID: 105667404

3. *(4 pts) Let* $T(n) = 4T(n/5) + \log(n)$*, where* $T(n)$ *is constant when* $n \leq 2$*.* ***Using the Master Theorem****, determine tight asymptotic bounds for* $T(n)$*. That is, use the Master Theorem to find a function* $g(n)$ *such that* $T(n) \in \Theta(g(n))$*. Clearly show all your work.*

   Solution.

4. *(6 pts) Let $T(n) = T(n-3) + T(3) + n$, where $T(n)$ is constant when $n \leq 3$.* ***Using unrolling****, determine tight asymptotic bounds for $T(n)$. That is, find a function $g(n)$ such that $T(n) \in \Theta(g(n))$. Clearly show all your work.*

   Solution.

5. *(8 pts) Consider the following algorithm, which takes as input a string of nested parentheses and returns the number of layers in which the parentheses are nested. So for example, "" has* 0 *nested parentheses, while* ((())) *is nested* 3 *layers deep. In contrast,* ()() *is* ***not*** *valid input. You may assume the algorithm receives only valid input. For the sake of simplicity, the string will be represented as an array of characters.*

   Find a recurrence for the worst-case runtime complexity of this algorithm. Then ***solve*** *your recurrence and get a tight bound on the worst-case runtime complexity.*

```
CountParens(A[0, ..., 2n-1]):
    if A.length == 0:
        return 0
    return 1 + CountParens(A[1, ..., 2n-2])
```

   Solution.

6. *(16 pts) For the given algorithm to find* ***min****, solve the following.*

   You may assume the existence of a ***min*** *function taking* $\mathcal{O}(1)$ *time, which accepts at most three arguments and returns the smallest of the three.*

```
FindMin(A[0, ..., n-1]):
     if A.length == 1:
         return A[0]
     return min( FindMin(A[0, ..., floor(n/3)],
                 FindMin(A[floor(n/3) + 1, ..., floor(2n/3)],
                 FindMin(A[floor(2n/3) + 1, ..., n-1])
             )
```

   (a) (3pts) Find a recurrence for the worst-case runtime complexity of this algorithm.
   Solution.

(b) (3 pts) Solve your recurrence **using the Master's Method** and get a tight bound on the worst-case runtime complexity.

Solution.

(c) (6 pts) Solve your recurrence **using the recurrence tree method** and get a tight bound on the worst-case runtime complexity. (It's ok to put an image of your hand drawn tree but label it neatly.)

Solution.

(d) (4 pts) Give a tight bound (Θ bound) on the number of `return` calls this algorithm makes. Justify your answer.

Solution.

7. *(7 pts) Consider the following algorithm that sorts an array.*

   Express and provide the worst-case runtime complexity of this algorithm as a function of n, where n represents the size of the array. Provide a tight bound on the worst-case runtime complexity.

```
buffSort(A, size):
    if size <= 1:
        return

    buffSort(A, size-1)

    foo = Arr[size-1]

    for(index = size-2; index >= 0 AND A[index] > foo; index--)
        A[index+1] = A[index]

    A[index+1] = foo
```

   Solution.