Name: Luke Joyce

ID: 107355873

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: Luke Joyce
ID: 107355873

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.

$$\lim_{n\to\infty} \frac{f(n)}{g(n)}$$

$$= \lim_{n\to\infty} \frac{ln(n)}{n^{1/2}}$$

$$= \lim_{n\to\infty} \frac{1/n}{\frac{1}{2}\frac{1}{\sqrt{n}}}$$

$$= 2 \times \lim_{n\to\infty} \frac{1/n}{\frac{1}{\sqrt{n}}}$$

$$= 2 \times \lim_{n\to\infty} \frac{\sqrt{n}}{n} = 0$$

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$$

$$\therefore\ ln(n) \in \mathcal{O}(\sqrt{n})$$

Name: Luke Joyce

ID: 107355873

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

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

---

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.

$$\lim_{n\to\infty} \frac{f(n)}{g(n)}$$

$$= \lim_{n\to\infty} \frac{(n-3)!}{3^{5n}}$$

$$a_n = \frac{f(n)}{g(n)}$$

$$= \lim_{n\to\infty} \frac{a_{n+1}}{a_n}$$

$$= \lim_{n\to\infty} \frac{\frac{(n+1-3)!}{3^{5(n+1)}}}{\frac{(n-3)!}{3^{5n}}}$$

$$= \lim_{n\to\infty} \left(\frac{(n-2)!}{3^{5n+5}} \times \frac{3^{5n}}{(n-3)!}\right)$$

$$= \lim_{n\to\infty} \left(\frac{(n-2)}{3^{5n+5}} \times 3^{5n}\right)$$

$$= \lim_{n\to\infty} \frac{(n-2)}{3^5} = \infty$$

$$\therefore f(n) = (n-3)! \in \Omega(3^{5n})$$

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.
   Using Master Theorem generic form $T(n) = aT(\frac{n}{b}) + f(n)$ we can come up with a = 4, b = 5, and f(n) = log(n) $\in \mathcal{O}(n^c)$ for some c > 0.
   Master Theorem Case 1: if $f(n) = \mathcal{O}(n^{log_b a - \epsilon})$ for some constant $\epsilon > 0$, then T(n) = $\mathcal{O}(n^{log_b a})$.
   We must find ϵ such that $0 < \epsilon < log_5 4$.
   When $\epsilon = log_5 3$, log(n) $\leq (n^{log_5 4 - log_5 3}) = log(n) \leq (n^{log_5 4/3})$ so that T(n) $\in \theta(n^{log_5 4})$

Name: Luke Joyce
ID: 107355873

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

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. Let C = T(n) when n≤3. New equation T(n-3) + n + C. Unrolling: T(n) = T(n-3) + n + C
   T(n-3) = T(n-6) + n - 3 + C
   T(n) = T(n-6) + n - 3 + C + n + C = T(n-6) + 2n + 2C - 3
   T(n-6) = T(n - 9) + n - 6 + C
   T(n) = T(n - 9) + n - 6 + C + 2n + 2c - 3 = T(n - 9) + 3n + 3C - 9
   After k iterations, the function will have unrolled to T(n - 3k) + kn + kC_n
   .

   Now we find what (n-3k) needs to be in our base case: n - 3k = 3
   k = (n - 3)/3
   k = $\frac{n}{3}$ - 1.

   Substitute k back into the above function:
   T(n - 3($\frac{n}{3}$ - 1)) + ($\frac{n}{3}$ - 1)n + ($\frac{n}{3}$ - 1)C_n
   = T(3) + ($\frac{n^2}{3}$ - n) + ($\frac{n}{3}$ - 1)C_n

   The highest power in this function is n^2. Therefore we can say that g(n) = n^2 and T(n) ∈ $\theta(n^2)$

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. Finding the recurrence relation: Only 1 conditional with one sub-problem per. After each call the input (2n) is reduced down to 2n - 2 = 2(n-1).

   WE can make a recurrence relation based off that: T(2n) = T(2(n-1)) + 1. + 1 is for the base case (n = 0), and T(2(n-1)) for n > 1. Now we unroll this function: T(2n) = T(2(n-1)) + 1 T(2n) = T(2(n-2)) + 1 + 1 = T(2n - 4) + 2 After k iterations, T(2n) = T(2(n - k)) + k. Find the base case for k: n - k = 0, k = n. T(2(n-n)) + n = 1 + n. There fore T(n) $\in \theta(n)$

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 == 0:
        return infinity
     else if A.length == 1:
         return A[0]
     else if A.length == 2:
        return min(A[0], A[1])
     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])
               )
```

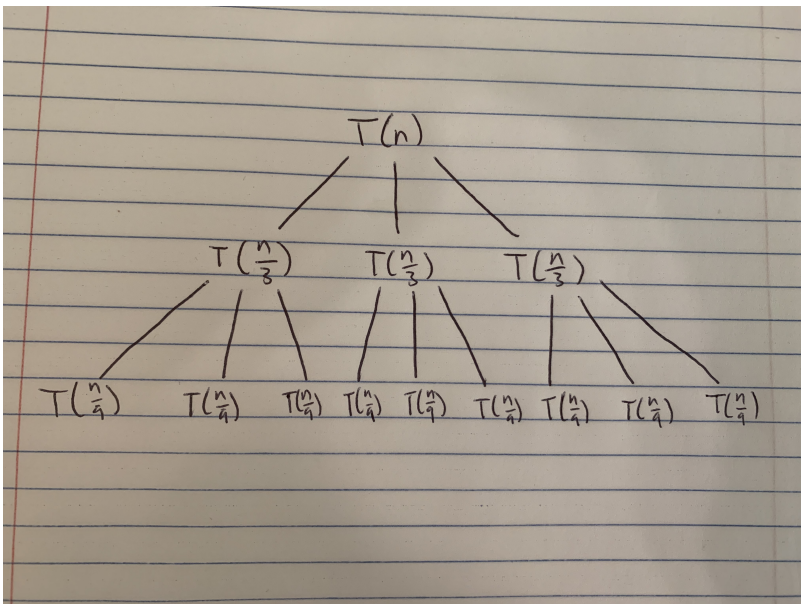
   (a) (3pts) Find a recurrence for the worst-case runtime complexity of this algorithm.

   Solution. 3 different conditionals in this one. Each is bounded by $\theta(1)$ because min has a runtime of O(1). 3 Sub-problems, and the size of each is 3, therefore the recurrence relation should be: T(n) = $3T(\frac{n}{3} + \theta(1))$

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

Solution. Using Master Method for the above recurrence relation we find that a = 3, b = 3, f(n) = $\theta(1)$
f(n) = O($n^{log_b a-\epsilon}$) for some $\epsilon > 0$ Since $log_3 3$=1, and ϵ must be less than 1, we'll call $\epsilon = 1/2$. $\therefore$ T(n) = $\theta(n)$

(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.
The cost of each level is as follows:
Level 0: 1C
Level 1: 3C
Level 2: 9C
First row (T(n)) is row 0, so the number of levels would be $log_3(n) + 1$, and the number of elements per level is equal to 3^i
$\sum_{i=0}^{log_3(n)+1} C(3^i)$ is the summation based off the tree, and C is the cost constant.
Based on that formula we can come up with $\frac{1-3^{log_3 n-1}}{1-3} = \frac{1-3n}{-2}$ so T(n) $\in \theta(n)$

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

Solution.
$\sum_{i=0}^{log_3(n)+1} C(3^i) = (C)\frac{1-3^{log_3 n-1}}{1-3} = (C)\frac{1-3n}{-2}$

Since we found in part (c) that each FindMin() call has only one return, C is equal to 1 in the above equation. So the number of return calls has a tight bound of $\theta(n)$

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. The worst case runtime for the loops iterations is n + C. Only one subproblem whose size is n - 1, since the problem is making the array shorter by one index.
   We can unroll now: T(n) = T(n-1) + n + C.
   T(n-1) = T(n-1) + n - 1 + C
   T(n) = T(n-2) + 2n + C
   After k iterations T(n) = T(n - k) + kn + C. Base case for k: n-k=0, therefore n=k.
   PLug that back in to get T(n) = T(n-n) + n(n)+C = n^2.$Therefore we get T(n) \in \theta(n^2)$.