Name: Alex Book
ID: 108073300

Collaborated with: Michael Hasenkamp, Bum Kim, Irvin Carbajal, Varun Narayanswamy

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: Alex Book
ID: 108073300
Collaborated with: Michael Hasenkamp, Bum Kim, Irvin Carbajal, Varun Narayanswamy
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.
   $L = \lim_{n\to\infty} \frac{ln(n)}{\sqrt{n}} = \frac{\infty}{\infty}$ (indeterminate form, apply L'Hopital's Rule)
   $L = \lim_{n\to\infty} \frac{\frac{1}{n}}{\frac{1}{2\sqrt{n}}} = \lim_{n\to\infty} \frac{2}{\sqrt{n}} = 0$
   Therefore, by the limit comparison test, since $L = 0$, $\ln(n) \in \mathcal{O}(\sqrt{n})$.

Name: Alex Book
ID: 108073300
Collaborated with: Michael Hasenkamp, Bum Kim, Irvin Carbajal, Varun Narayanswamy
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.
   Apply ratio test.
   $a_n = \frac{f(n)}{g(n)} = \frac{(n-3)!}{3^{5n}}$
   $\mathrm{L} = \lim_{n\to\infty} \mid \frac{a_{n+1}}{a_n} \mid = \lim_{n\to\infty} (\mid \frac{(n-2)!}{3^{5n+5}} \mid * \mid \frac{3^{5n}}{(n-3)!} \mid) = \lim_{n\to\infty} \frac{(n-2)}{3^5} = \infty$
   Therefore, by the limit comparison test, since $L = \infty$, $f(n) \in \Omega(g(n))$.

Name: Alex Book
ID: 108073300
Collaborated with: Michael Hasenkamp, Bum Kim, Irvin Carbajal, Varun Narayanswamy
CSCI 3104, Algorithms **Profs. Hoenigman & Agrawal**
Problem Set 2b (51 points) **Fall 2019, CU-Boulder**

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.
   Note: this uses the '2nd way' as described by Shiv in Piazza post @128, as this type of Master's Theorem problem was not covered in lecture.

   $T(n) = 4T(\frac{n}{5}) + \log(n)$
   $\log(n) = \mathcal{O}(n^{\epsilon})$ where ϵ>0
   $T(n)$<$4T(\frac{n}{5}) + n^{\epsilon}$
   Choose $\epsilon = .1$ because a value of $\epsilon < \log_5(4)$ gives the tightest possible bound on T(n).
   Therefore $T(n) = 4T(\frac{n}{5}) + n^{.1}$.
   .1<$log_5(4)$, so by Master's Theorem, $T(n) \in \mathcal{O}(n^{\log_5(4)})$ (now $\mathcal{O}$ instead of Ω because n^{ϵ} is asymptotically larger than $\log(n)$).

Name: Alex Book
ID: 108073300
Collaborated with: Michael Hasenkamp, Bum Kim, Irvin Carbajal, Varun Narayanswamy
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.
   $T(n) = T(n-3) + n$
   $T(n) = T(n-6) + 2n - 3$
   $T(n) = T(n-9) + 3n - 9$
   $T(n) = T(n-12) + 4n - 18$
   $T(n) = T(n-15) + 5n - 30$

   Recognize pattern:
   $T(n) = T(n-3k) + kT(3) + kn + c_1$

   Base Case: $T(3) = c_2$ (runtime is constant for $n \leq 3$)

   Recursion stops when $n - 3k = 3$.
   $k = \frac{n-3}{3}$
   $T(n) = T(n - 3\frac{n-3}{3}) + \frac{n-3}{3}T(3) + \frac{n-3}{3}n + c_1$
   $T(n) = T(3) + \frac{n-3}{3}T(3) + \frac{n^2-3n}{3} + c_1$
   $T(n) = c_2 + \frac{n-3}{3}c_2 + \frac{n^2-3n}{3} + c_1$
   $T(n) \in \Theta(n^2)$

   Therefore, $g(n) = n^2$.

Name: Alex Book
ID: 108073300
Collaborated with: Michael Hasenkamp, Bum Kim, Irvin Carbajal, Varun Narayanswamy
CSCI 3104, Algorithms **Profs. Hoenigman & Agrawal**
Problem Set 2b (51 points) **Fall 2019, CU-Boulder**

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.
   $f(n) \in \Theta(1)$ (if statement and one simple return, so constant runtime aside from recursion)
   One sub-problem of size $2n - 2$.
   For clarity and simplicity, let n' represent the length of the the input, therefore $n' = 2n$.

   $T(n') = T(n' - 2) + \Theta(1)$
   $T(n') = T(n' - 4) + 2\Theta(1)$
   $T(n') = T(n' - 6) + 3\Theta(1)$
   $T(n') = T(n' - 6) + 4\Theta(1)$

   Recognize pattern:
   $T(n') = T(n' - 2k) + k\Theta(1)$
   Base Case: $T(0) = 1$

   Recursion stops when $n' - 2k = 0$.
   $k = \frac{n'}{2}$
   $T(n') = T(n' - 2\frac{n'}{2}) + \frac{n'}{2}\Theta(1)$
   $T(n') = T(0) + \frac{n'}{2}\Theta(1)$
   $T(n') = 1 + \frac{n'}{2}\Theta(1)$
   $T(n') \in \Theta(n') = \Theta(2n) = \Theta(n)$

Name: Alex Book
ID: 108073300
Collaborated with: Michael Hasenkamp, Bum Kim, Irvin Carbajal, Varun Narayanswamy
CSCI 3104, Algorithms **Profs. Hoenigman & Agrawal**
Problem Set 2b (51 points) **Fall 2019, CU-Boulder**

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.
   $f(n) \in \Theta(1)$ (`if` statement and simple `return` or a `min` call, so constant runtime aside from recursion)
   Three sub-problems, each of size $\frac{n}{3}$.

   $T(n) = 3T(\frac{n}{3}) + \Theta(1)$

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

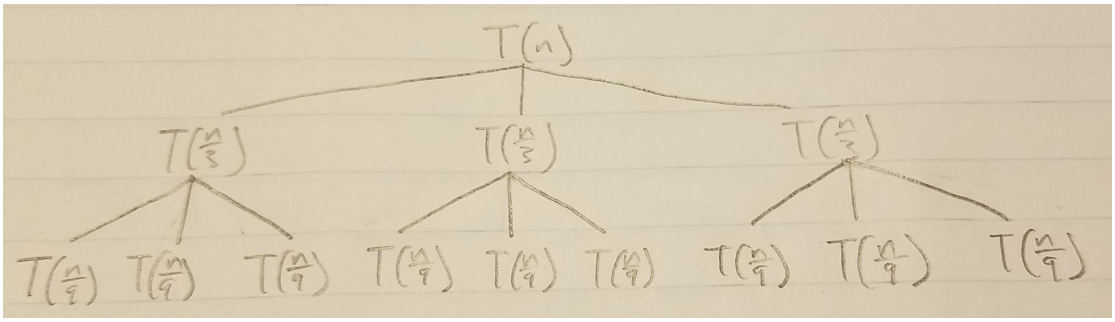
Solution.
$T(n) = 3T(\frac{n}{3}) + \Theta(1)$
$a = 3, b = 3, c = 0$ (because $n^0 = 1 \in \Theta(1)$)
$\log_b(a) = \log_3(3) = 1 > 0 = c$
Therefore by Master's Method, $T(n) \in \Theta(n^{\log_3(3)}) = \Theta(n)$.

Name: Alex Book
ID: 108073300

Collaborated with: Michael Hasenkamp, Bum Kim, Irvin Carbajal, Varun Narayanswamy

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

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



(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.

Note: Apologies, I couldn't figure out how to move the picture around the page, but anytime I reference a tree/node/root is referencing the above image.

Taking level $i = 0$ as being the root, the number of nodes per level can be represented by 3^i. The total number of levels can be represented by $\log_3(n) + 1$ (the log is floored, but as discussed in office hours, flooring is not needed in this example of runtime analysis). The cost per node can be represented by a constant c, as the complexity of each call of FindMin is $\Theta(1)$.

Therefore, the total cost of the function can be represented by the following:
$T(n) = \sum_{n=0}^{\log_3(n)} (c * 3^i)$

$T(n) = \sum_{n=0}^{\log_3(n)} (c * 3^i) = c\frac{1-3^{\log_3(n)+1}}{1-3} = c\frac{1-3n}{-2}$ (closed form of summation of geometric series)
$T(n) \in \Theta(n)$

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

Solution.

The summation from part c represents total cost (cost per node $*$ total number of nodes). Take out the constant cost c per call of FindMin, and the summation is equal to the total number of nodes (total number of calls of FindMin). Since every call of FindMin has one return, such a summation would give a tight bound on the total number of `return` calls this algorithm makes.

Let $R(n)$ represent the number of `return` calls this algorithm makes.
$R(n) = \sum_{n=0}^{\log_3(n)} 3^i$

$R(n) = \sum_{n=0}^{\log_3(n)} 3^i = \frac{1-3^{\log_3(n)+1}}{1-3} = \frac{1-3n}{-2}$ (closed form of summation of geometric series)
$R(n) \in \Theta(n)$

Name: Alex Book
ID: 108073300
Collaborated with: Michael Hasenkamp, Bum Kim, Irvin Carbajal, Varun Narayanswamy
CSCI 3104, Algorithms **Profs. Hoenigman & Agrawal**
Problem Set 2b (51 points) **Fall 2019, CU-Boulder**

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.
   For clarity and simplicity, let n=`size`.
   $f(n) \in \Theta(n)$ (if statement and one simple return, for loop of size $n-1$, so $\Theta(n)$ runtime aside from recursion)
   One sub-problem of size $n-1$.

   $T(n) = T(n-1) + \Theta(n)$
   $T(n) = T(n-2) + 2\Theta(n)$
   $T(n) = T(n-3) + 3\Theta(n)$

   Recognize pattern:
   $T(n) = T(n-k) + k\Theta(n)$
   Base Case: $T(1) = 1$

   Recursion stops when $n - k = 1$.
   $k = n - 1$
   $T(n) = T(n-(n-1)) + (n-1)\Theta(n)$
   $T(n) = T(1) + (n-1)\Theta(n)$
   $T(n) = 1 + (n-1)\Theta(n)$
   $T(n) \in \Theta(n^2)$