Name: Mauro Vargas

ID: 107212593

CSCI 3104, Algorithms **Escobedo & Jahagirdar**

Homework 1A (30 points) **Summer 2020, 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**, we cannot accept hand-written solutions. Here's a short intro to **Latex**.
- In this homework we denote the asymptomatic *Big-O* notation by $\mathcal{O}$ and *Small-O* notation is represented as o.
- We recommend using online Latex editor **Overleaf**. Copy and paste the **.tex** file located in Canvas into the overleaf editor.
- 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 Canvas 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.

Name: Mauro Vargas
ID: 107212593

CSCI 3104, Algorithms **Escobedo & Jahagirdar**
Homework 1A (30 points) **Summer 2020, CU-Boulder**

Piazza threads for hints and further discussion

Piazza Threads
Question 1
Question 2
Question 3

Recommended reading: For complete background read Chapters 1, 2 and 3. Chapter 3 will especially be helpful.

1. *(5 pts) Provide an example of* ***unique*** *functions $f(n)$, $g(n)$, $h(n)$ such that*
   $f(n) \in \mathcal{O}(g(n))$ is an asymptotic upper bound.
   $f(n) \in \mathcal{O}(h(n))$ is an asymptotically ***tight*** *upper bound.*
   and also give a brief description of the difference between asymptotically ***tight*** *upper bound and asymptotic upper bound.*

   Unique functions : $f(n) = n, g(n) = 2n, h(n) = \frac{1}{3}n$
   $f(n) \in \mathcal{O}(h(n))$ is an asymptotically **tight** upper bound: if Θ-notation bounds a function to within constant factors. We write $f(n) = \Theta\ (g(n))$ if there exist positive constants n_0, c_1, and c_2 such that at and to the right of n_0, the value of $f(n)$ always lies between $c_1\ g(n)$ and $c_2\ g(n)$ inclusive.
   $f(n) \in \mathcal{O}(g(n))$ is an asymptotic **upper bound**: if $\mathcal{O}$-notation gives an upper bound for a function to within a constant factor. We write $f(n) = \mathcal{O}g(n)$ if there are positive constants n_0 and c such that at and to the right of n_0, the value of $f(n)$ always lies on or below $c_1 g(n)$.
   Differences between asymptotically **tight upper bound** and asymptotic **upper bound**. $\Theta\ g(n) = f(n)$: there exist positive constants c_1, c_2, and n_0 such that:
   $0 \leq c_1\ g(n) \leq f(n) \leq c_2\ g(n)$ for all $n \geq n_0$ **vs** $\mathcal{O}g(n) = f(n)$: there exist positive constants c and n_0 such that: $0 \leq f(n) \leq cg(n)$ for all $n \geq n_0$

Name: Mauro Vargas
ID: 107212593

CSCI 3104, Algorithms **Escobedo & Jahagirdar**
Homework 1A (30 points) **Summer 2020, CU-Boulder**

2. *(5 pts) The small o-notation is used to represent upper bounds that are not asymptotically* ***tight****.*
State if the above statement is true or false and briefly justify your answer by comparing the small o-notation to big $\mathcal{O}$-notation .

The statement above is true. Big O-Notation is bounded above and can be a tight bound **versus** Small o-Notation is bounded above ,but can not be a tight bound.
$\mathcal{O}g(n) = \{f(n)$: there exist positive constants c and n_0 such that:
$0 \leq f(n) \leq cg(n)$ for all $n \geq n_0$ $\}$
$og(n) = \{f(n)$: for any positive constant $c > 0$, there exist a constant $n_0 > 0$ such that $0 \leq f(n) < cg(n)$ for all $n \geq n_0$ $\}$
The main difference is that in $f(n) = Og(n)$, the bound $0 \leq f(n) \leq cg(n)$ holds for some constant $c > 0$, but in $f(n) = o(g(n))$, the bound $0 \leq f(n) < cg(n)$ holds for all constants $c > 0$.

3. *($4 \times 5 = 20$ pts) For each of the following pairs of functions $f(n)$ and $g(n)$, we have that $f(n) \in \mathcal{O}(g(n))$. Find valid constants c and n_0 in accordance with the definition of big $\mathcal{O}$-notation. For the sake of this assignment, both c and n_0 should be strictly less than 20. You do* ***not*** *need to formally prove that $f(n) \in \mathcal{O}(g(n))$. For example if it is known that $g(n)$ grows faster than $f(n)$, you need not state or formally prove. (that is, no induction proof or use of limits is needed).*

(a) $f(n) = n$ and $g(n) = n \log_e(n)$.

$\mathcal{O}(g(n)) = \{f(n)$: there exist positive constants c and n_0 such that:
$0 \leq f(n) \leq cg(n)$ for all $n \geq n_0$ $\}$

$c = 1$ and $n_0 = e$ hold true for Big O-Notation
$0 \leq n \leq (c)\ n \log_e(n)$ for all $n \geq n_0$
$0 \leq 1 \leq (c)\ \log_e(n)$, $n_0 = e$
$0 \leq 1 \leq c\ (1)$

(b) $f(n) = n!$ and $g(n) = 2^{n \log_2(n)}$.

$\mathcal{O}g(n) = \{f(n)$: there exist positive constants c and n_0 such that:
$0 \leq f(n) \leq cg(n)$ for all $n \geq n_0\}$
$0 \leq n! \leq c * 2^{n \log_2(n)}$ for all $n \geq n_0$
$0 \leq n! \leq c*\left(\ 2^{\log_2(n)} \ \right)^n$
$0 \leq n! \leq c*\left(n\right)^n$, $n_0 = 1$
$1 \leq$ c
c = 1, $n_0 = 1$

(c) $f(n) = 3^n$ and $g(n) = (2n)!$

$\mathcal{O}g(n) = \{f(n)$: there exist positive constants c and n_0 such that:
$0 \leq f(n) \leq cg(n)$ for all $n \geq n_0\}$
$0 \leq 3^n \leq c * (2n)!$ for all $n \geq 1.394$
c = 1, $n_0 = 1.394$

(d) $f(n) = n \log_{10}(n)$ and $g(n) = n \log_2(n)$

$\mathcal{O}g(n) = \{f(n)$: there exist positive constants c and n_0 such that:
$0 \leq f(n) \leq cg(n)$ for all $n \geq n_0\}$
$0 \leq n \log_{10}(n) \leq cn \log_2(n)$ for all $n \geq n_0$, divide everything by n
$0 \leq \log_{10}(n) \leq c * \log_2(n)$
$0 \leq \frac{\log(n)}{\log(10)} \leq c*\frac{\log(n)}{\log(2)}$
$0 \leq \frac{1}{\log(10)} \leq \frac{c}{\log(2)}$
$0 \leq \frac{\log(2)}{\log(10)} \leq c$, $c = \frac{\log(2)}{\log(10)}$ and $n_0 = 1$

4. **Extra Credit (5% of total homework grade)** For this extra credit question, please refer the leetcode link provided below or click here. Multiple solutions exist to this question ranging from brute force to the most optimal one. Points will be provided based on Time and Space Complexities relative to that of the most optimal solution.

   Please provide your solution with proper comments which carries points as well.

   https://leetcode.com/problems/find-all-numbers-disappeared-in-an-array/

```
//This problem could be solved in many ways,
//but having a O(n) is what makes this problem hard.
//First For Loop
//subtract one from value and save value in place holder
//whatever value we get, we want to treat this as a new index,
//this allows us to go through the vector without making a nested loop,
//Negate value in our new index,
//this show us that we have visited this element before.
//Second For Loop
////if non negative numbers then push back
//insert index value plus one to make the
//missing numbers from 1 <= nums[i]<= n appear
class Solution {
public:
    vector<int> findDisappearedNumbers(vector<int>& nums) {
        for(int i=0; i< nums.size(); i++){
            int placeholder = abs(nums[i]) - 1;
            if(nums[placeholder] > 0){
                nums[placeholder] = -nums[placeholder];
            }
        }
        vector<int> ans; // create new vector
        for (int i = 0; i < nums.size(); i++) {
            if (nums[i] > 0){
                ans.push_back(i+1);
            }
        }
        return ans;
    }
};
```