Name: Qiuyang Fu

ID: 108667420

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**. Download the **.tex** file from Canvas and upload it on overleaf to edit.
- 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: Qiuyang Fu

ID: 108667420

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.*

   If $f(n) = 2n$, $g(n) = n^2$, $h(n) = n$, then we can say $f(n) \in \mathcal{O}(g(n))$ is an asymptotic upper bound and $f(n) \in \mathcal{O}(h(n))$ is an asymptotically **tight** upper bound.
   Asymptotically upper bound may or may not be asymptotically tight upper. Just as the example showed above. If asymptotically upper bound is asymptotically tight upper, it means the function $f(n)$ becomes significant relative to $g(n)$ as n approaches infinity. However as to asymptotically upper bound, function $f(n)$ may become insignificant relative to $g(n)$ as n approaches infinity.

Name: Qiuyang Fu

ID: 108667420

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 above statement is true. If a certain function satisfied small o-notation then it also satisfied the big $\mathcal{O}$-notation. But if a certain function satisfied big $\mathcal{O}$-notation, it may or may not satisfied small o-notation.
The main difference is that in $f(n) \in \mathcal{O}(g(n))$, the bound $0 \leq f(n) \leq cg(n)$ holds for some constant $c > 0$, but in $f(n) \in o(g(n))$, 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 $g(n) = n \log_e(n)$.

Valid constants $c = 1$, $n_0 = 3$.

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

Since $g(n) = 2^{n \log_2(n)} = n^n$, it is obvious that valid constants $c = 1$, $n_0 = 1$ satisfied.

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

Valid constants $c = 1$, $n_0 = 2$.

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

Valid constants $c = 1$, $n_0 = 1$.

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/

   The two solution below will all run in $O(n)$ time complexity with space complexity of $O(1)$ does not include result space.
   Solution 1:

```
class Solution {
public:
    vector<int> findDisappearedNumbers(vector<int>& nums) {
        // The solution below without extra space and in O(n) runtime,
        // the returned list does not count as extra space.
        vector<int> result;
        // Change all the elements in the array to its right places
        for(int i = 0;i < nums.size();i++) {
            int tmp = nums[i];
            int index = i + 1;
            while (tmp != index) {
                int value = tmp;
                int right_place = value - 1;
                tmp = nums[right_place];
                nums[right_place] = value;
                index = right_place + 1;
            }
        }
        // elements that does not in the right place is absent
        for(int i = 0;i < nums.size();i++) {
            if (nums[i] != i + 1) {
                result.push_back(i + 1);
            }
        }
        return result;
    }
};
```

Solution 2:

```
class Solution {
public:
    vector<int> findDisappearedNumbers(vector<int>& nums) {
        // The solution below without extra space and in O(n) runtime,
        // the returned list does not count as extra space.
        vector<int> result;
        for(int i = 0;i < nums.size();i++) {
            int index = abs(nums[i]) - 1;
            // make the value to be negative which is in the array
            if (nums[index] > 0) {
                nums[index] = -nums[index];
            }
        }
        // positive values in the array is absent
        for(int i = 0;i < nums.size();i++) {
            if (nums[i] > 0) {
                result.push_back(i + 1);
            }
        }
        return result;
    }
};
```