Name: YOUR NAME HERE

ID: YOUR STUDENT ID HERE

CSCI 3104, Algorithms **Due February 5, 2021**

Problem Set 3 (50 points) **Spring 2021, 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** and we cannot accept hand-written solutions. Here's a short intro to Latex.
- You should submit your work through **Gradescope** only.
- The easiest way to access Gradescope is through our Canvas page. There is a Gradescope button in the left menu.
- Gradescope will only accept **.pdf** files.
- It is vital that you match each problem part with your work. Skip to 1:40 to just see the matching info.

Name: YOUR NAME HERE
ID: YOUR STUDENT ID HERE

CSCI 3104, Algorithms **Due February 5, 2021**
Problem Set 3 (50 points) **Spring 2021, CU-Boulder**

1. Name (a) one advantage, (b) one disadvantage, and (c) one alternative to worst-case analysis. For (a) and (b) you should use full sentences.

Name: YOUR NAME HERE
ID: YOUR STUDENT ID HERE

CSCI 3104, Algorithms **Due February 5, 2021**
Problem Set 3 (50 points) **Spring 2021, CU-Boulder**

2. Put the growth rates in order, from slowest-growing to fastest. That is, if your answer is $f_1(n), f_2(n), \ldots, f_k(n)$, then $f_i(n) \leq O(f_{i+1}(n))$ for all i. If two adjacent ones have the same order of growth (that is, $f_i(n) = \Theta(f_{i+1}(n))$), you must specify this as well. Justify your answer (show your work).

   - You may assume transitivity: if $f(n) \leq O(g(n))$ and $g(n) \leq O(h(n))$, then $f(n) \leq O(h(n))$, and similarly for little-oh, etc. Note that the goal is to order the growth rates, so transitivity is very helpful. We encourage you to make use of transitivity rather than comparing all possible pairs of functions, as using transitivity will make your life easier.
   - You may also use the limit test (see Michael's Calculus Notes on Canvas, referred to as the Limit Comparison Test). However, you **MUST** show all limit computations at the same level of detail as in Calculus I-II.
   - You may **NOT** use heuristic arguments, such as comparing degrees of polynomials or identifying the "high order term" in the function.
   - If it is the case that $g(n) = c \cdot f(n)$ for some constant c, you may conclude that $f(n) = \Theta(g(n))$ without using Calculus tools. You must clearly identify the constant c (with any supporting work necessary to identify the constant- such as exponent or logarithm rules) and include a sentence to justify your reasoning.

   (a) .

$$2^n, \qquad n^2, \qquad \frac{1}{n}, \qquad 1, \qquad 6n + 10^{100}, \qquad \frac{1}{\sqrt{n}}, \qquad \log_5(n^2), \qquad \log_2(n), \qquad 3^n.$$

Name: YOUR NAME HERE
ID: YOUR STUDENT ID HERE

CSCI 3104, Algorithms **Due February 5, 2021**
Problem Set 3 (50 points) **Spring 2021, CU-Boulder**

3. Analyze the worst-case running time for each of the following algorithms. You should give your answer in Big-Theta notation. You *do not* need to give an input which achieves your worst-case bound, but you should try to give as tight a bound as possible.

   - In the column to the right of the code, indicate the cost of executing each line once.
   - In the next column (all the way to the right), indicate the number of times each line is executed.
   - Below the code, justify your answers (show your work), and compute the total runtime in terms of Big-Theta notation. You may assume that $T(n)$ is the Big-Theta of the high-order term. You need not use Calculus techniques. However, you **must** show the calculations to determine the closed-form expression for a summation (represented with the $\sum$ symbol). You cannot simply say: $\sum_{i=1}^{n} i \in \Theta(n^2)$, for instance.
   - In both columns, you don't have to put the *exact* values. For example, putting "c" for constant is fine. We will not be picky about off-by-one errors (i.e., the difference between n and $n-1$); however, we will be picky about off-by-two errors (i.e., the difference between n and $n-2$).

   (a) Consider the following algorithm.

```
                                                | Cost     | # times run
1  f(A[1, ..., n]):
2    ans = 0                                    |          |
3    for i = 1 to n-1:                          |          |
4      for j = i+1 to n:                        |          |
5        if A[i] > A[j]:                        |          |
6           ans += 1                            |          |
7
8    return ans                                 |          |
```

(b) Write an algorithm to multiply two matrices A with dimension $n \times m$ and B with dimension $m \times n$. Write your algorithm in pseudocode in Latex, similar in style to the given pseudocode in $3(a)$. Analyze the worst-case runtime of your algorithm as above. You may use the subroutine `zeros([a,b])` to create an array of zeros with dimension $a \times b$. `zeros([a,b])` has runtime $\Theta(ab)$.

4. For the following algorithms, write down the recurrence relation associated with the runtime of the algorithm. **You do not** have to solve the recurrence relation.

   (a) Algorithm A solves problems by dividing them into five sub-problems of half the size, recursively solving each sub-problem, and then combining the solutions in linear time. For a problem of size 1, the algorithm takes constant time.

   (b) Algorithm B solves problems of size n by recursively solving two sub-problems of size $n-1$ and then combining the solutions in constant time. For a problem of size 1, the algorithm takes constant time.

   (c)
```
def hello(n) {
     if (n > 1) {
        print( 'hello' 'hello' 'hello' )
        hello(n/7)
        hello(n/7)
        hello(n/7)
        hello(n/7)
}}
```