Name: Vera Duong

ID: 109431166

**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: Vera Duong
ID: 109431166

**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.

   **Solution:**

   (a) One advantage to worst case analysis is that you would know for certain that a specific algorithm would perform *at the very least* that well. It calculates that upper bound of the running time. In a real world scenario, if an algorithm took much longer than expected because worst case wasn't considered, it could be detrimental.
   (b) One disadvantage to worst case analysis is that it is often rare to occur, so one shouldn't give the impression that it is a standard, especially if the algorithm involves randomized data (like sorting an array) because we don't always know if the numbers will be arranged to be in the worst case.
   (c) alternative to worst case analysis would be to use average-case analysis since it calculates what will most likely occur and what could perform in a real life scenario.

Name: Vera Duong
ID: 109431166

**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) .

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

**Solution:**

To put the growth rates in order we will use asymptotic notation with limits:

- $\lim_{n\to\infty} \frac{1/n}{1} = \lim_{n\to\infty} \frac{1}{n} = 0$, therefore $\frac{1}{n}$ is o(1).
- $\lim_{n\to\infty} \frac{1}{6n + 10^{100}} \Rightarrow \lim_{n\to\infty} 1 = \mathbf{1}$ and $\lim_{n\to\infty} 6n + 10^{100} = \infty \Rightarrow \frac{1}{\infty} = 0$, therefore 1 is o$(6n + 10^{100})$
- $\lim_{n\to\infty} \frac{6n + 10^{100}}{n^2} = \lim_{n\to\infty} \left(\frac{6}{n} + \frac{10^{100}}{n^2}\right) \Rightarrow \lim_{n\to\infty} \frac{6}{n} = 0$ and $\lim_{n\to\infty} \frac{10^{100}}{n^2} = 0$, therefore $6n + 10^{100}$ is o$(n^2)$.
- $\lim_{n\to\infty} \frac{n^2}{2^n} =$ *using the ratio test* $\lim_{n\to\infty} \left(\frac{(n+1)^2}{2^{(n+1)}}\right) \frac{2^n}{n^2} = \lim_{n\to\infty} \frac{2^n(n^2 + 2n + 1)}{n^2 \cdot 2^{n+1}} = \lim_{n\to\infty} \frac{n^2 + 2n + 1}{2n^2} =$ $\lim_{n\to\infty} \frac{n^2}{2n^2} + \frac{2n}{2n^2} + \frac{1}{2n^2} = \frac{1}{2} + 0 + 0 = \frac{1}{2}$. Due to the fact that the ratio test returns a number less than one (1/2 less than 1), the sequence will go to zero. This is because if the series converges then the sequence of its terms must go to 0. Thus, $n^2$ is o$(2^n)$.
- $\lim_{n\to\infty} \frac{2^n}{3^n} = \lim_{n\to\infty} \left(\frac{2}{3}\right)^n = 0$. The base is less than one and the exponent continues to grow, the limit will be zero. We can also look at it this way: $\lim_{n\to\infty} 2^n 3^{-n} = 0$. Therefore $2^n$ is o$(3^n)$.
- $\lim_{n\to\infty} \frac{1/n}{1/\sqrt{n}} = \lim_{n\to\infty} \frac{\sqrt{n}}{n} = \lim_{n\to\infty} n^{-1/2} = \lim_{n\to\infty} \frac{1}{\sqrt{n}} = 0$, therefore $\frac{1}{n}$ is o$\left(\frac{1}{\sqrt{n}}\right)$.
- $\lim_{n\to\infty} \frac{1/\sqrt{n}}{1} = 0$, therefore $\frac{1}{\sqrt{n}}$ is o(1).
- $\lim_{n\to\infty} \frac{\log_5(n^2)}{\log_2 n} \Rightarrow$ *L'H* $\lim_{n\to\infty} \frac{2/\ln(5)x}{1/x \ln 2} = \lim_{n\to\infty} \frac{2 \ln 2}{\ln 5} = \frac{2 \ln 2}{\ln 5}$, therefore since we get a constant, $\log_5(n^2)$ is $\Theta(log_2 n)$

(*continue on next page*)

- $\lim_{n\to\infty} \frac{1}{\log_5(n^2)} = \frac{1}{\infty} = 0$, therefore 1 is $o(\log_5(n^2))$
- $\lim_{n\to\infty} \frac{6n + 10^{100}}{\log_5(n^2)} \Rightarrow$ *L'H* $\lim_{n\to\infty} \frac{6}{2/\ln(5)n} = \lim_{n\to\infty} \frac{6\ln(5)n}{2} = \lim_{n\to\infty} 3\ln(5)n = \infty$, therefore $6n + 10^{100}$ is $\omega(\log_5(n^2))$

**Final order from slowest to fastest:**

$\mathbf{\frac{1}{n}} < \mathbf{\frac{1}{\sqrt{n}}} < \mathbf{1} < \log_{\mathbf{5}}(\mathbf{n^2}) = \log_{\mathbf{2}}(\mathbf{n}) < \mathbf{6n + 10^{100}} < \mathbf{n^2} < \mathbf{2^n} < \mathbf{3^n}$

Name: Vera Duong
ID: 109431166

**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                                        |    c1    | 1
3    for i = 1 to n-1:                              |    c2    | n
4      for j = i+1 to n:                            |    c3    | 1 + ∑_{x=1}^{x=n} x
5        if A[i] > A[j]:                            |    c4    | ∑_{x=1}^{x=n} x
6          ans += 1                                 |    c5    | ∑_{x=1}^{x=n} x
7
8    return ans                                     |    c6    | 1
```

   **proof / work:**
   Each line has the cost of a constant. Line 2 runs once since it's a declaration. Line 3 runs n times because of the n-1 and we need to account for its termination. Line 4 runs $1 + \sum_{x=1}^{x=n} x$ times because the $j^{th}$ index relies on what the value of $i$ is, so the number of times we pass through the inner loop is different. Lines 5 and 6 have the same number of times to run since we are looking at the worst case, so it will always be called (without any termination steps).

$$
\begin{aligned}
T(n) &= c_1 + c_2 n + c_3(1 + \sum_{x=1}^{x=n} x) + c_4(\sum_{x=1}^{x=n} x) + c_5(\sum_{x=1}^{x=n} x) + c_6 \\
&= c_2 n + \left(\sum_{x=1}^{x=n} x\right)(c_3 + c_4 + c_5) + c_7 \\
&= c_2 n + \left(\sum_{x=1}^{x=n} x\right)(c_8) + c_7 \\
&= c_2 n + \left(\frac{n(n+1)}{2}\right)(c_8) + c_7 \\
&= c_8 \frac{n^2}{2} + c_9 \frac{3n}{2} + c_7 \Rightarrow \text{same format as: } an^2 + bn + c
\end{aligned}
$$

   We will then use the limit comparison test where we choose $f(n) = c_8 \frac{n^2}{2} + c_9 \frac{3n}{2} + c_7$ and $g(n) = n^2$
   $\lim_{n\to\infty} \frac{c_8 \frac{n^2}{2} + c_9 \frac{3n}{2} + c_7}{n^2} \Rightarrow L'H \lim_{n\to\infty} \frac{c_8 n + c_9 \frac{3}{2}}{2n} \Rightarrow L'H \lim_{n\to\infty} \frac{c_8}{2} = \frac{c_8}{2}$

   Since the limit is a constant number, we know that $f(n)$ is $\Theta(g(n))$. Thus, this algorithm will have a worst case running time of $\Theta(n^2)$

(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 $\Theta(ab)$.

**Solution:**

```
                                                                  |Cost| # times run
1  function(A,B):
2    c = zeros([n,n])                                             | c1 | 1
3      for i = 1 to n:                                            | c2 | n+1
4        for j = 1 to n:                                          | c3 | n(n+1)
5           for k = 1 to m:                                       | c4 | n^2(m+1)
6               c[i][j] = c[i][j] + A[i][k] * B[k][j]             | c5 | n^2m
7    return c                                                     | c6 | 1
```

**proof / work:**
Each line has the cost of a constant, except for line 2 where $c_1 = n^2$. This is because calling the function itself is one run, however the function has a cost of $n^2$ because zeros([a,b]) has runtime of $\Theta(ab)$.

$$\begin{aligned} T(n,m) &= c_1 + c_2(n+1) + c_3 n(n+1) + c_4 n^2(m+1) + c_5 n^2 m + c_6 \\ &= n^2 + c_2 n + c2 + c_3 n^2 + c_3 + c_4 n^2 m + c + 4n^2 + c_5 n^2 m + c_6 \\ &= n^2(1 + c_3 + c_4) + c_2 n + n^2 m(c_4 + c_5) + c_3 + c_6 \\ &= an^2 m + bn^2 + cn + d \end{aligned}$$

to show the complexity in big theta, we first need to find an upper bound:

$T(n,m) = an^2m+bn^2+cn+d \leq an^2m+bn^2m+cn^2m+dn^2m = n^2m(a+b+c+d)$ when n and m are $\geq 1$, so $T(n,m)$ is in $\mathcal{O}(n^2m)$ for $k = 1$ and $C = a+b+c+d$ where we know that $a, b, c, d$ are positive.

Next we need to show the lower bound:

$T(n,m) = an^2m + bn^2 + cn + d \geq an^2m$ for $n, m \geq 1$. This is because we know that $a, b, c, d$ are positive. So, $T(n,m)$ is in $\Omega(n^2m)$ for $k = 1$ and $C = a$.

Thus this algorithm has a worst case runtime complexity of $\Theta(n^2m)$ since T(n,m) is both $\mathcal{O}(n^2m)$ and $\Omega(n^2m)$.

Name: Vera Duong
ID: 109431166

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

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)
}}
```

**Solution**:

A general form of recurrence relations is as follows:

$$T(n) = \begin{cases} \Theta(1) & \text{if } n \leq c \\ aT(\frac{n}{b}) + D(n) + C(n) & \text{if } n > c \end{cases}$$

- T(n) is the running time for a problem of size n
- D(n) is the time to divide the problem into sub problems
- C(n) is the time to combine solutions to get the final solution
- a is the number of divisions
- $\frac{n}{b}$ is the size of the sub-problem

**(a)**

$$T(n) = \begin{cases} \Theta(1) & \text{if } n \leq 1 \\ 5T(\frac{n}{2}) + \Theta(n) & \text{if } n > 1 \end{cases}$$

The number of sub-problems is 5 which takes the spot of a. The size of the sub-problems is $\frac{n}{2}$ since the sub-problems are half the size, so it takes the spot of $\frac{n}{b}$. The time for dividing (D(n)) is $c_0 n$ which is linear and the time for combining (C(n)) is linear so it's also $c_1 n$. For a problem of size 1, the algorithm takes constant time.

**(b)**

$$T(n) = \begin{cases} \Theta(1) & \text{if } n \leq 1 \\ 2T(n-1) + \Theta(1) & \text{if } n > 1 \end{cases}$$

The number of sub-problems is 2 which takes the spot of a. The size of the sub-problems is $n-1$, so it takes the spot of $\frac{n}{b}$. The time for dividing (D(n)) is $c_0$ which is constant and the time for combining (C(n)) is constant so it's also $c_1$. For a problem of size 1, the algorithm takes constant time.

**(c)**

$$T(n) = \begin{cases} \Theta(1) & \text{if } n \leq 1 \\ 4T(\frac{n}{7}) + \Theta(1) & \text{if } n > 1 \end{cases}$$

The number of sub-problems is 4 which takes the spot of a. The size of the sub-problems is $\frac{n}{7}$ since the sub-problems are called with hello(n/7), so it takes the spot of $\frac{n}{b}$. The time for dividing (D(n)) is $c_0$ which is constant and the time for combining (C(n)) is constant so it's also $c_1$. For a problem of size 1, the algorithm takes constant time.