Name: Sasha Farhat

ID: 105887541

CSCI 3104, Algorithms **Escobedo & Jahagirdar**
Homework 1B (70 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: Sasha Farhat

ID: 105887541

CSCI 3104, Algorithms **Escobedo & Jahagirdar**
Homework 1B (70 points) **Summer 2020, CU-Boulder**

Piazza threads for hints and further discussion

Piazza Threads
Question 1 Question 2

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

1. *($4 \times 10 = 40$ pts) For each part of this question, put the growth rates in order, from slowest-growing to fastest. That is, if your answer is $f_1(n), f_2(n), ..., f_k(n)$, then $f_i(n) \leq \mathcal{O}(f_{i+1}(n))$ for all i. If two adjacent ones are asymptotically the same (that is, $f_i(n) = \Theta(f_{i+1}(n))$), you must specify this as well.*

   (a) Polynomials.
   $n^{\frac{1}{2}}, n + 10, n^{\frac{1}{3}}, n^3, n^3 + n^2 + 100, 2^{100}$

   Constant's come first in the order so:
   $2^{100} < n^{\frac{1}{2}}, n + 10, n^{\frac{1}{3}}, n^3, n^3 + n^2 + 100$
   $2^{100} < \mathcal{O}(n^{1/3})$ therefore $n^{1/3}$ comes next.
   $n^{1/3} < \mathcal{O}(n^{1/2})$ or $n^{1/3} < c \cdot n^{1/2}$ therfore $n^{1/2}$ comes next.
   $n^{1/2} < \mathcal{O}(n)$ therefore n comes next.
   Now we evaluate n^3 and $n^3 + n^2 + 100$
   $n^3 \leq \mathcal{O}(n^3 + n^2 + 100)$ that is $n^3 = \Theta(n^3 + n^2 + 100)$
   Therefore the polynomials are ordered as:
   $2^{100}, n^{1/3}, n^{1/2}, n, n^3 = n^3 + n^2 + 100$

   (b) Logarithms and related functions.
   $\log_3 n^2, (\log_3 n)^3, \log_3 n, \log_5 n^2, \log_2 n, \sqrt{n}$

$\log_3 n^2 \leq \mathcal{O}((\log_3 n)^3) \rightarrow 2\log_3 n \leq c \cdot (\log_3 n)^3$
$2 \leq c \cdot (\log_3 n)^2 \rightarrow$ therefore $\log_3 n^2 \leq \mathcal{O}((\log_3 n)^3)$ holds true
$\log_3 n \leq \mathcal{O}(\log_3 n^2)$
$log_3 n \leq c \cdot 2\log_3 n \ 1 \leq 2c \cdot \rightarrow$ using definition of $\Theta - notation 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \rightarrow$ therefore $\log_3 n = \Theta(\log_3 n^2)$
$\log_5 n^2 \leq \mathcal{O}(\log_3 n^2)$
$2\log_5 n \leq c \cdot 2\log_3 n$
$\frac{2\log n}{\log 5} \leq c \cdot \frac{2\log n}{\log 3} \rightarrow \frac{1}{\log 5} \leq \frac{c}{\log 3} \rightarrow$ using definition of Θ: $\log_5 n^2 = \Theta(\log_3 n^2)$
$\log_3 n \leq \mathcal{O}(\log_2 n)$
$\frac{\log n}{\log 3} \leq c \cdot \frac{\log n}{\log 2} \rightarrow \frac{1}{\log 3} \leq c \cdot \frac{1}{\log 2} \rightarrow$ definition of $\Theta - notation$ cab be applied here such $\log_3 n = \Theta(\log_2 n)$ is true
$\log_2 n \leq \mathcal{O}(\sqrt{n})$
$\log_2 n \leq c \cdot \sqrt{n}$ by big $\mathcal{O} - notation \log_2 n \leq \mathcal{O}(\sqrt{n})$ holds true
Therefore the polynomials are ordered as:
$\log_5 n^2 = \log_3 n^2 = \log_3 n = \log_2 n, \sqrt{n}, (\log_3 n)^3$

(c) Logarithms in exponents.
$n^{\log_3 n}, n^{\frac{1}{\log_3 n}}, n^{\log_4 n}, 1, n$

$n^{\log_4 n} \leq \mathcal{O}(n^{\log_3 n})$
$n^{\frac{\log n}{\log 4}} \leq c \cdot n^{\frac{\log n}{\log 3}} \rightarrow$ (took the $\log n$ root of the inequalities) $n^{\frac{1}{\log 4}} \leq c \cdot n^{\frac{1}{\log 3}} \rightarrow$ by big $\mathcal{O}$: $n^{\log_4 n} \leq \mathcal{O}(n^{\log_3 n})$
$1 \leq \mathcal{O}(n^{\frac{1}{\log_3 n}})$ first we simplify $n^{\frac{1}{\log_3 n}}$
$y = n^{\frac{1}{\log_3 n}} \rightarrow \log y = \frac{1}{\log_3 n} \log n \rightarrow \log y = \frac{\log 3}{\log n} \log n \rightarrow \log y = \log 3 \rightarrow y = 3$
$1 \leq \mathcal{O}(3)$, since$f(n)g(n)$ are constants $1 = \Theta(n^{\frac{1}{\log_3 n}})$
$n \leq \mathcal{O}(n^{\log_4 n})$
$n \leq c \cdot n^{\log_4 n} \rightarrow n_0 = 16 \rightarrow 16 \leq c \cdot 16^2 \rightarrow n \leq \mathcal{O}(n^{log_4 n})$ is true.
$n^{log_4 n} \leq \mathcal{O}(n^{log_3 n})$
$n^{\frac{\log n}{\log 4}} \leq c \cdot n^{\frac{\log n}{\log 3}} \rightarrow$ (log n root of the eniqualities) $n^{\frac{1}{\log 4}} \leq c \cdot n^{\frac{1}{\log 3}}$
using definition of big Θ: $0 \leq c_1 n^{log_3 n} \leq n^{log_4 n} \leq c_2 \cdot n^{log_3 n}$ holds true, therefore, $n^{log_4 n} = \Theta(n^{log_3 n})$ Therefore the polynomials are ordered as:
$1 = n^{\frac{1}{\log_3 n}}, n, n^{\log_4 n} = n^{\log_3 n}$

.

(d) Exponentials. (hint: Recall Stirlings approximation, which says that $n! \sim (\frac{n}{e})^n \sqrt{2\pi n}$ i.e. $\lim_{n\to\infty} \frac{n!}{(\frac{n}{e})^n \sqrt{2\pi n}} = 1$)

Name: Sasha Farhat

ID: 105887541

CSCI 3104, Algorithms **Escobedo & Jahagirdar**
Homework 1B (70 points) **Summer 2020, CU-Boulder**

$n!, n^5, 2^{2n}, 2^{2n+7}, 5^{n \log_5(n)}$

$n! \leq \mathcal{O}(n^5)$ using sterling's approximation
$(\frac{n}{e})\sqrt{2\pi n} \leq c \cdot n^5 \rightarrow \frac{n^{\frac{3}{2}}\sqrt{2\pi}}{e} \leq c \cdot n^5 \rightarrow \frac{\sqrt{2\pi}}{e} \leq c \cdot n^{\frac{7}{2}}$
therefore $n! \leq \mathcal{O}(n^5)$ is true.
$n^5 \leq \mathcal{O}(5^{n \log_5 n}) \rightarrow 5^{n \log_5 n} = 5^n$
$n^5 \leq c \cdot 5^n \rightarrow$ as we can see $n^5 \leq \mathcal{O}(5^{n \log_5 n}$ holds true,
$5^{n \log_5 n} \leq \mathcal{O}(2^{2n})$
$5^n \leq c \cdot 2^{2n}$ using the definition of big $\Theta \rightarrow 0 \leq c_1 2^{2n} \leq 5^n \leq c_2 2^{2n}$
therfore $5^{n \log_5 n} = \Theta(2^{2n})$
$2^{2n} \leq \mathcal{O}2^{2n+7}$
$2^{2n} \leq c \cdot 2^7 \cdot 2^{2n} \rightarrow$ using definition of big Θ: $0 \leq c_1 2^{2n+7} \leq 2^{2n} \leq c_2 2^{2n+7}$
therefore $2^{2n} = \Theta 2n + 7$
Therefore the polynomials are ordered as:
$n!, n^5, 5^{n \log_5(n)} = 2^{2n+7} = 2^{2n}$

2. *($3 \times 10 = 30$ pts) For each of the following algorithms, analyze the worst-case running time. You should give your answer in $\mathcal{O}$ 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.*
Justify your answer (show your work). This likely means discussing the number of atomic operations in each line, and how many times it runs.

(a)

```
1 count = 0
2 for(i = 1; i < n; i = i + 1)
3 {
4     for(j = i; j < n; j = j + 1)
5     {
6         count = count+1
7     }
8 }
```

evaliuating the run time/cost of each line we get:
$c_1 + c_2(n) + c_3(n) + c_4(n^2)$
The outer-loop increments i+1 and thus runs $n-1$ times and similarily the inner loop increments j+1 and runs $n-1$ times.
Becuase the count function is within the two loops it run's n^2 the constants "c" has a time complexity of $\mathcal{O}(1)$
Here we can see that the time complexity of the loops are equal to the # of times the innermost loops is executed:
therefore in terms of big $\mathcal{O} - notation$ the following algorithm has a worst-case running time of $\mathcal{O}(n^2)$

(b)

```
1 count = 0
2 for(i = 1; i <= n; i = i * 2)
3 {
4     for(j = 0; j < n; j = j + 2)
5     {
6         count = count+1
7     }
8 }
```

evaluating the runtime of each line:
$c_1 + c_2(\log n) + c_3(n) + c_4(n \log(n))$
Evaluating the outer-loop increments $i \cdot 2$ and thus runs $\log n$ times (because i is multiplied by two every time we get the equation $i = 2^k$. Thus $2^k = n$. Solving for k we get $\log_2 n$). The inner loop increments $j + 2$ and thus has a runs $n - 2$ times.
The count function is nested within both for loops and thus has a time complexity of $n \cdot \log(n)$
as we can see from the evaulated lines, the time complexity is of the following alogirthm is $\mathcal{O}(n \log n)$

(c) Here A is a list of integers of size atleast 2 and sqrt(n) returns the square root value of its argument. You can assume that the upper bound of calculating square root takes big $\mathcal{O}(k)$ time. Provide an upper bound in terms of n and k.

```
1  count = 0
2  for(i = 0; i < n; i = i + 1)
3  {
4      for(j = i+1; j < n; j = j + 1)
5      {
6          if(A[i]>sqrt(A[j]))
7          {
8              count = count+1
9          }
10
11     }
12 }
```

Name: Sasha Farhat

ID: 105887541

CSCI 3104, Algorithms **Escobedo & Jahagirdar**
Homework 1B (70 points) **Summer 2020, CU-Boulder**

Because the inner and outer loop are dependent of eachother we canno compte their complexity directly.
First let's evaluate the variable i when $i = 0$,we se tgat the sqrt function executed n-1 times. Making a table of for the # of times n runs, we get:
$i = 1 \rightarrow k(n-1) | i = 2 \rightarrow k(n-2) | i = 3 \rightarrow k(n-3)$ ... $k(0)$
Thus the number of times that the algorithm executes is $\sum_{i=1}^{n} c \cdot n(n+1)$ the runtime of the algorithm will thus be $\mathcal{O}(k \cdot n^2)$

3. **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/product-of-array-except-self/

   ```
   Replace this text with your source code inside of the .tex document
   ```