Name: Mauro Vargas Jr
ID: 107212593

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: Mauro Vargas Jr
ID: 107212593

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 × 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.
   $n^{\frac{1}{2}}, n + 10, n^{\frac{1}{3}}, n^3, n^3 + n^2 + 100, 2^{100}$

   Note: $\mathrm{f}_1(n) = \mathcal{O}(1)$; $\mathrm{f}_1(n) = 2^{100}$, $f_2(n) = n^{\frac{1}{3}}$, $f_3(n) = n^{\frac{1}{2}}$, $f_4(n) = n + 10$, $f_5(n) = n^3$, $f_6(n) = n^3 + n^2 + 100$
   We know that $f_1 = 2^{100}$ is a constant and does not grow, therefore it's an element to any $f_i(n) = function$. We also know that polynomials with higher degrees grow faster. $f_2(n)$ grows slower then $f_3(n)$, because it has a smaller degree $n^{\frac{1}{3}} \leq n^{\frac{1}{2}}$. Next we have $f_4 = n + 10$ and n^3, again we take the highest degree, so $n + 10 \in n^3$. Last we have $n^3 = \Theta n^3 + n^2 + 100$ various asymptotic notations are closely related to the definition of a limit. $\lim n \to \infty \frac{n^3}{n^3+n^2+100} = 1$, thus L'Hospital's rule tells us that $f_5(n)$ grows at the same rate as $f_6(n)$ and $f_6(n)$ also holds a upper and lower bound to $f_5(n)$.
   Answer: $f_1(n) \in \mathcal{O} \in \mathcal{O}(f_3(n)) \in \mathcal{O}(f_4(n) \in \mathcal{O}(f_5(n) = \Theta(f_6(n))$

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

$f_1(n) = \log_3(n)$, $f_2(n) = \log_5(n^2)$, $f_3(n) = log_2(n)$, $f_4(n) = log_3(n^2)$,
$f_5(n) = \sqrt{n}$, $f_6(n) = \log_3(n))^3$
Using the $\mathcal{O}$ definition and log proprieties we can determine which functions are elements of one another. $\mathcal{O}g(n) = \{f(n)$: there exist positive constants c and n_0 such that: $0 \leq f_1(n) \leq cf_2(n)$, for all $n \geq n_0$
$0 \leq \log_3(n) \leq clog_5(n^2)$, for all $n \geq n_0$
$0 \leq \frac{log(n)}{log(3)} \leq \frac{c2\log(n)}{\log(5)}$,Use log properties $\log(x)^c = c\log(x)$ and $\log_c(x)) = \frac{\log(n)}{\log(x)}$
$0 \leq \frac{1}{log(3)} \leq \frac{2}{\log(5)}$, cancel out $\log(n)$ from both numerators and $c = 1$,thus we can see that the statement holds true and we can do this for the rest of the $f_i(n)$ functions.
Answer: $f_1(n) \in \mathcal{O}(f_2(n)) \in \mathcal{O}(f_3(n)) \in \mathcal{O}(f_4(n)) \in \mathcal{O}(f_5(n)) \in \mathcal{O}(f_6(n))$

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

$f_1(n) = 1$, $f_2(n) = n^{\frac{1}{\log_3(n)}}$, $f_3(n) = n$, $f_4(n) = n^{\log_4(n)}$, $f_5(n) = n^{\log_3 n}$
$f_1(n) = 1$ and $f_2(n) = n^{\frac{1}{\log_3(n)}}$ are both constants when taking the limit to infinity. $f_2(n)$ turns into a constant when n = 3, we get $f_2(n) = 3^1 = 3$, Using the L Hospital method we will get a constant when taking the
$\lim n \to \infty(\frac{1}{n^{\frac{1}{\log_3 n}}})= \frac{1}{3}$ telling us that $f_1(n) \in \Theta(f_2(n))$. $f_2(n)$ is an element of $f_3(n)$ because it has the higher degree $n^{\frac{1}{\log 4}} \leq n^1$. $f_3(n) \in \mathcal{O}(f_4(n)))$, because of the higher degree. Finally $f_4(n) = \Theta(f_5(n))$ because the base does not change the function it self $f_4(n)$ and $f_5(n)$ are the same function and grow at the same rate, therefore they elements of one another.
Answer:
$f_1(n) \in \Theta(f_2(n)) \in \mathcal{O}(f_3(n)) \in \mathcal{O}(f_4(n))= \Theta(\mathrm{f}_5(n))$

.

(d) Exponentials. (hint: Recall Stirling's 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: Mauro Vargas Jr
ID: 107212593

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

$f_1(n) = 2^{2n}, f_2(n) = 2^{2n+7}, f_3(n) = n^5, f_4(n) = n!, f_5(n) = 5^{n\log_5(n)}$
$f_1(n) = 2^{2n}$ and $f_2(n) = 2^{2n+7}$ are the two most slowing growing functions because of their constant base. Although $f_2(n)$ has an additional constant to its exponent it does effect the over all time complexity meaning that the two functions grow at the same rate hence $f_1(n)$=$\Theta(f_2(n))$. We get $f_2(n) \in \mathcal{O} f_3(n)$ because $f_3(n) = n^5$ has a **n** for a base, so it goes eventually grows faster then any constant base exponent. Note: $n!$ eventually grows faster then any exponent function except when its n^n. $f_5(n) = 5^{n\log_5(n)}$ is equal to n^n.
$f_5(n) = 5^{n\log_5(n)}$
$f_5(n) = 5^{\log_5(n)} * 5^n$
$f_5(n) = n * 5^n = 5 * n^n$, since constants don't effect the over all time complexity we get n^n proving why $(f_4(n)) \in \mathcal{O}(\mathrm{f}_5(n))$
Answer:
$f_1(n)$=$\Theta(f_2(n)) \in \mathcal{O}(f_3(n)) \in \mathcal{O}(f_4(n)) \in \mathcal{O}(\mathrm{f}_5(n))$

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)
```
count = 0
for(i = 1; i < n; i = i + 1)
{
    for(j = i; j < n; j = j + 1)
    {
        count = count+1
    }
}
```

$- - - - - - - - - - - - - - - - >$**Cost** **Time**
$count = 0 - - - - - - - - - - - > c_1$———$1$
$for(i = 1; i < n; i = i + 1) - - > c_2$———$n$
$for(j = i; j < n; j = j + 1) - - > c_3$———$n$
$count = count + 1 - - - - - - - - > c_4$———$1$

$T(n) = c_1 + c_4 + (c_2 * n)(c_3 * n) = \mathcal{O}(n^2)$

(b)

```
count = 0
for(i = 1; i <= n; i = i * 2)
{
    for(j = 0; j < n; j = j + 2)
    {
        count = count+1
    }
}
```

$- - - - - - - - - - - - - - - - >$**Cost Time**
$count = 0 - - - - - - - - - - - > c_1$———1
$for(i = 1; i <= n; i = i * 2) - - > c_2$———$(n-1)$
$for(j = 0; j < n; j = j + 2) - - > c_3$———$(n)$
$count = count + 1 - - - - - - - > c_4$———1
$T(n) = c_1 + c2 * (n-1) + c_3 * n + c_4 = \mathcal{O}(n^2)$

(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.

```
count = 0
for(i = 0; i < n; i = i + 1)
{
    for(j = i+1; j < n; j = j + 1)
    {
        if(A[i]>sqrt(A[j]))
        {
            count = count+1
        }

    }
}
```

$- - - - - - - - - - - - - - - - ->$**Cost Time**
$count = 0 - - - - - - - - - - - - > c_1$——1
$for(i = 0; i < n; i = i + 1) - - -- > c_2$——$(n)$
$for(j = i + 1; j < n; j = j + 1)- > c_3$——$(n + 1)$
$if(A[i] > sqrt(A[j])) - - - - - - - > c_4$——1
$count = count + 1. - - - - - - - -- > c_5$——1
$T(n) = c_1 + c2 * (n) + c_3 * (n + 1) + c_4 + c_5 = \mathcal{O}(n^2)$

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/

```
class Solution {
public:
    vector<int> productExceptSelf(vector<int>& nums) {
  //When computing the product we need to skip the current postion[i] and compute t
  //We can accomplish this allusion by creating three vectors
  //First we need to break this down, by tying to get the numbers on the left and r
  //After after making seperate vector we can multiple them togather into one vecto
  //First three create a Right, Left and answer vector
  int n =  nums.size();
  std::vector<int> v_R(n);
  std::vector<int> v_L(n);
  std::vector<int> v_A(n);
  std::cout << " size v_R: " << v_R.size() << '\n';
```

```
    std::cout << " size v_L: " << v_L.size() << '\n';
    std::cout << " size v_A: " << v_A.size() << '\n';
    v_L.front() = 1 ;//add a one to the front of the vector
    v_R.back() = 1;// and  to back
    // this allows us to multiple elements into our vector
    for (int i=1; i<n; i++)// we want to skip the current postion so we start at i =
      v_L[i] = nums[i - 1] * v_L[i - 1];//we want to multiple our first element in nu
    for (int j = n - 2; j >= 0; j--) // Do the same for the right side only, do not m
      v_R[j] = nums[j + 1] * v_R[j + 1];// we use j + 1 because we want to multiple t
    for (int k = 0; k < n; k++)// know we can multiple every postion aganist our left
      v_A[k] = v_L[k] * v_R[k];// multipling current postion will give us the actual
    return v_A;
      }

  };
```