ECS 32B Homework 1

ECS 32B — Spring 2019

Geoffrey Mohn ID: 912568148

April 16, 2019

Problem 1

Complexity of a Code Sample

For this problem you will analyze the complexity of the following bit of instructive Python code that does not solve anything important.

```
a = 4
b = 10
for i in range(n):
        for j in range(a):
            total = total + 1
    for i in range(b) :
        total = total + 1
print(total)
```

a) **State a function T(n) for this code in terms of n. The function T(n) should give the number of statements executed by the Python interpreter as a function of the integer variable n. Explain your reasoning.**

$$T(n) = O(1) + O(1) + O(1) + O(1) + (n^2 \times O(1)) + O(1) + (n \times O(1)) + O(1)$$

Reasoning: The first 2 O(1) initializes the variables 'a' and 'b' which is ran in a constant amount of time. The 3rd and 4 O(1) sets up the for loop of i in range of n, and the for loop of j in range of a respectively which are also ran in a constant amount of time. The initialized variable 'total' (O(1)) in the nested for loops is iterated n^2 times as the for loops are executed n(for i in range(n)) × n (for j in range(a)) times. The following O(1) sets up the for loop for i in range of b. The variable 'total' is intialized and ran n times for i in range(b). The print statement takes a constant amount of time and is expressed as O(1) as well.
T(n) = 2+2+n^2+1+n+1
or T(n) = n^2 + n + 6

b) **Give the smallest worst-case (big-O) complexity for this code in terms of n that works. Formally prove it by finding c and n_0 such that T(n) $\leq$ cf(n) for n $\geq n_0$**

T(n) $= n^2 + n + 6$
T(n) $\leq$ cf(n)
$n^2 + n + 6 \leq cn^2$
$n^2 + n \leq cn^2 - 6$
n$(n+1) \leq cn^2 - 6$
$(n+1) \leq cn - \frac{6}{n}$
$n \leq cn - \frac{6}{n} - 1$ try $c = 2$, try $n_0 = 6$
$6 \leq 12 - 2$
$6 \leq 10 =$ True
Choosing c = 2 and $n_0 = 6$, we have shown that n^2 + n + 6 is O(n^2) because $n^2 + n + 6 \leq 2n^2$ for $n \geq 6$

Problem 2

Exponential Complexity

Suppose the number of operations required by a particular algorithm is exactly T(n) = 2^n and our 1.6 Ghz computer performs exactly 1.6 billino operations per second. What is the largest problem, in terms of n, that can be solved in under a second? In under a day?

T(n) = 2^n 1.6 Ghz computer performing 1,600,000,000 operations per second
1,600,000,000 = 2^n
log_2(1,600,000,000) = $log_2(2) \times n$
log_2(1,600,000,000) = n
$n \approx 30.5$
86,400 seconds in a day
log_2(1,600,000,000)$\times 86,400$
$\approx 2,641,716.7$
O(2^n) where the largest problem in terms of n, is $n \approx 30.5$ per second and $\approx 2,641,716.7$ per day

Problem 3

The Traveling Salesman Problem

Given a list of cities and the distances in between them, the task is to find the shortest possible tour that starts at a city, vists each city exactly once and returns to a starting city.A particular tour can be described as list of all cities [c1, c2, c3, ..., cn] ordered by the position in which they are visited with the assumption that you return from the last city to the start.

n = number of cities
m = n × n matrix of distances
min = ∞
the for loop checks randomly and takes all possible outcomes then checks each outcome for shortest distance, this would check $n!$ times.
suppose $n = 4 \therefore m = 16$ $\forall$ possible tours theres 4 starting locations, once one is picked there are 3 remaining destinations to go after, then 2 and after that 1.
the for loops would iterate $4 \times 3 \times 2 \times 1$ or, 4!
the (big-O) complexity would be n! or O(n!)

Problem 4

Complexity Bound Types

Formal proofs are not required here but briefly explain your reasoning.

is $log_2(n)O(n)$?
yes, $log_2(n)$ is logarithmic, whereas O(n) is linear. O(n)denotes an upper bound. O(n) is above $log_2(n)$ so O(n) shows the upper bound and can therefore be lower than O(n)

is $log_2(n)\Omega(n)$?
no, $log_2(n)$ is below $\Omega(n)$ and $\Omega(n)$ denotes a lower bound. Therefore anything below the lower bound cannot be $\Omega(n)$

is $log_2(n)\Theta(n)$?
no, Θ denotes an asymptotically tight upper and lower bound. if the function is $\Theta(n)$ then it is also O(n) and Ω(n). Ω(n) is not $log_2(n)$ because $log_2(n)$ is lower than the lower bound $\Omega(n)$.

Problem 5

Calculating Bounds

Suppose an algorithm solves a problem of size n in at most T(n) $= 2n^3+n^2+1$ **steps.**

a) **Prove that T(n) is O** (n^3)**. Show your work including values for c and** n_0
T(n) $= 2n^3 + n^2 + 1$
$2n^3 + n^2 + 1 \leq cn^3$
$2n^3 + n^2 \leq cn^3 - 1$
$n^2(2n + 1) \leq cn^3 - 1$
$2n + 1 \leq cn - \frac{1}{n^2}$
$2n \leq cn - \frac{1}{n^2} - 1$ let $n_0 = 1$ and c $= 4$
$2(1) \leq (4)(1) - \frac{1}{1} - 1$
$2 \leq 2$

b) **Prove that T(n) is** $\Theta(n^3)$ **by proving that it is also** $\Omega(n^3)$ **Show your work including values for c and** n_0
show T(n) is $\Omega(n)$
T(n) $= 2n^3 + n^2 + 1$
$2n^3 + n^2 + 1 \geq cn^3$
$2n^3 + n^2 \geq cn^3$ - 1
$n^2(2n + 1) \geq cn^3$ - 1
$(2n + 1) \geq cn - \frac{1}{n^2}$
$2n \geq cn - \frac{1}{n^2} - 1$ let $n_0 = 1$ and c $= 2$
$2(1) \geq (2)(1) - \frac{1}{1} - 1$
$2 \geq 0$
so T(n) is also $\Omega(n)$ $\therefore$ T(n) is $\Theta(n)$ since T(n) is $\Omega(n)$ and T(n) is $\Omega(n)$