CSE 102: Data Structures and Algorithms
Quiz II (TOTAL OF 20 POINTS), 28 Jan, 2020
Good luck! Please write legibly, ambiguous answers will not be given credits. It is a closed book quiz.

Name: **Roll no:**

1. What is Big O? Is it useful? If so why? [3 points]

Ans. Big O is the most common method and notation for discussing the execution time of algorithms. It is the **asymptotic execution time** of the algorithm.

If there are positive constants c and n_0, such that for all n >= n_0,
f(n) >= cg(n)
then we say that **f(n) is of order O(g(n))** [1.5 marks definition]

Yes, it is useful. Because, Big-O notation is the efficiency/performance of your work. You have to know how fast your code works when the inputs get bigger because in real life you can't know the exact number of inputs.
Furthermore, you can't compare two different algorithmic approaches without an asymptotic notation so if you want to choose the better one, you are going to compare them with big-O and see which one fits your situation. Both may be inefficient but you will know which one is better. [1.5 marks reason]
(Students may answer it differently)

2. What is the Order [Big O notation] of an algorithm that takes $2n^3+14n^2+8n+45$. Justify answer. [4 points]

Ans. The algorithm requires O(n3) time. [1 mark for complexity]

When we have a polynomial that describes the time requirements of an algorithm, we simplify it by:
– Throwing out all but the highest-order term
– Throwing out all the constants [3 marks for justification]

3. What are recurrence relations? Why are they needed? [3 points]

Ans. A recurrence relation for the sequence $\{a_n\}$ is an equation that expresses a_n in terms of one or more of the previous terms of the sequence, namely, a_0 , a_1 , …, a_{n-1} , for all integers n with n >= n_0 , where n_0 is a non-negative integer
Or a recurrence relation is like a recursively defined sequence, but without specifying any initial values (initial conditions). [1.5 marks for definition]

In case of Recursive algorithms, it is not easy to compute the number of instructions by looking at code.
➔ The number of instructions in one instance of function call depends on the number of instructions executed when recursive calls are made.
➔ Hence to analyze recursive algorithms, we require more sophisticated techniques

• Specifically, we need to solve recurrence relations [1.5 marks for importance]

4. Derive factorial complexity through solving recurrence relation? [4 points]

[1 mark for recurrence, 2 for derivation, 1 for Complexity]

Ans. The number of multiplications can be expressed as a formula-

$T(n) = T(n-1) + 1, \quad T(0) = 0$

$T(n) = T(n-1) + 1$ // $T(n-1) = T(n-2) + 1$

$= [T(n-2) + 1] + 1$

$= T(n-2) + 2$ // $T(n-2) = T(n-3) + 1$

$= T(n-3) + 3$

.. . .

$= T(n-k) + k$

To use Boundary condition let $T(n-k) = T(0)$

$k = n$

$T(n) = 0 + n = n$

$T(n) = O(n)$

5. What is the Order of fastest known Fibonacci sequence generation algorithm? What gives it its efficiency? [3 points]

Ans. Log(n). Its efficient because of matrix multiplication and use of recursion to solve the multiplication operation. [1.5 for Order , 1.5 for reason]

6. Is there a theoretical limit on the performance of sorting algorithm? Specify some ways to get around such limits, if there are any. [3 points]

Ans. **Yes. There is a nlogn hard limit.** [1.5]

Can be relaxed through parallelization and computational speed up. [1.5]