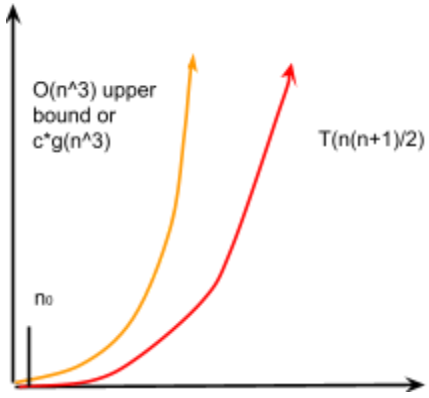
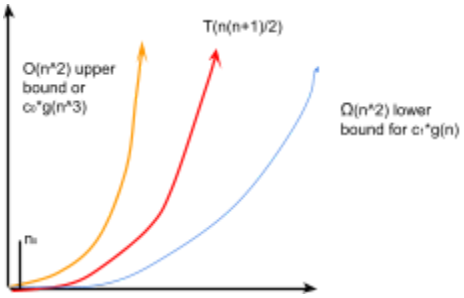
Sami Noor Syed
ONID: 934330738
CS 325 Analysis of Algorithms
Assignment #1
Due date: 6/27/22

1) **Identify and compare the order of growth**
   a) n(n+1)/2 ∈ O^3 True
      i) The algorithm with a time complexity n(n+1)/2 ∈ O(n^2) as shown below
         (1) n(n+1)/2 = (n^2+n)/2 [multiplying n into the parenthesis]
         (2) (n^2+n)/2 <= (n^2 + n) for all n >=0[multiply the RHS by 2]
         (3) (n^2 + n)<= n^2 + n^2 for all n>= 0[multiply last term of RHS by n]
         (4) n^2 + n^2 = 2*n^2 [combine like terms]
         (5) Therefore n(n+1)/2 <= 2*n^2 <= c*n^2 for some greater >= c and n
      ii) Since O(n^2) ∈ O(n^3) for all n >=1 as "O" denotes an upper growth bound and O(n^3) has by definition a larger growth rate than O(n^2), we can say definitively that n(n+1)/2 ∈ O^3.
      iii)


   b) n(n+1)/2∈Θ(n^2) True
      i) Using the same math from before to prove n(n+1)/2∈Θ(n^2):
         (1) n(n+1)/2 = (n^2+n)/2 [multiplying n into the parenthesis]
         (2) (n^2+n)/2 <= (n^2 + n) for all n >=0[multiply the RHS by 2]
         (3) (n^2 + n)<= n^2 + n^2 for all n>= 0[multiply last term of RHS by n]
         (4) n^2 + n^2 = 2*n^2 [combine like terms]
         (5) Therefore n(n+1)/2 <= 2*n^2 <= c*n^2 for some values >= c_0
         (6) It can also be said n(n+1)/2 >= c*n^2 for some values <= c_1

ii) 

c) 10n-6 ∈Ω(78n + 2020) False

i) (78n + 2020) has a higher starting n!∈Ω(0.00001n) value we can see that using L'hospitals rule that it has a much higher growth rate than 10n-6:

(1) Lim n→∞T(n)/g(n)=limn→∞T′(n)/g′(n)

(2) Lim n→∞ ((78n + 2020)/(10n-6)) = Limn→∞ ((78)/(10)) = 7.8

(3) Since the limit as n approaches ∞ = some constant, we should say that 10n-6 ∈Θ(78n + 2020)

ii) Since this statement is false, there is no need to draw a graph as per the directions stated in the Hw

d) n!∈Ω(0.00001n) True

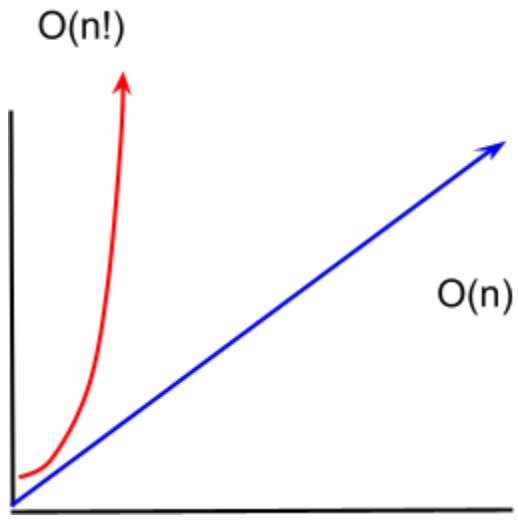
i) Using sterlings formula in conjunction with L'hospitals rule to prove the above:

ii) Lim n→∞T(n)/g(n) = Lim n→∞(n!)/(.00001n)

iii) ≈ Lim n→∞ ((2πn)^.5 *(n/e)^n)/(.00001n) [since n→∞, we can assum n is very large and apply Sterling's formula]

iv) I got stuck on being able to take the derivative of sterlings formula… so I will say that empirically n! Grows much faster than .00001n and and that Lim n→∞(n!)/(.00001n) = ∞

e)



i) There is no value of n where n>0 For which n>n!, so in this graph there is no n_0

2) Read and Analyze Pseudocode:

a) This algorithm computes the difference of the maximum and minimum values contained within the array from A[0] to A[n-1]

b) There are two lines of code that are performed the maximum number of times:

i) if A[i] < minval:

ii) if A[i] > maxval:

c) Each of these operations is performed **n-1** times:

i) From 1 to n-1 = n-1 operations

d) The time complexity of this algorithm is **O(n)**

3) Using mathematical induction:

a) ***Loop invariant****:*

i) The elements between (exclusive) the interval of i and j will be in the reverse order of the original array

b) ***Initialization*** 2 possible cases for the first loop*:*

i) If the Array has an even number of elements, then j = i+1 and there will be no elements between (exclusive) both i and j. Therefore, those elements are not in any order and are therefore the loop invariant is considered vacuously true.

ii) If the array has an odd number of elements, then j = i. Therefore, there cannot be any elements between A[j] and A[i] and the loop invariant is vacuously true.

c) ***Maintenance***: on the (i+1) iteration:

i) The items that are at the index values of i and j are an equal distance from the center of the array and are swapped, causing their order to be reversed.

ii) As i decrements for each iteration, j increments for each iteration. This causes the next iteration to swap the values that are located one position farther away from the center than the previous values, thus maintaining the loop invariant as i and j represent index values farther and father from the center of the array.

d) Termination:

i) The Algorithm will terminate as soon as both i and j reach either end of the array. For i, this occurs because i starts in the middle of the array and decrements at every iteration of the loop, causing it to eventually = 0 (the very start of the array). For j this occurs because j starts in the middle and increments at every iteration of the loop, causing it to eventually = n-1. Given how both i and j are initially calculated, this should happen simultaneously.