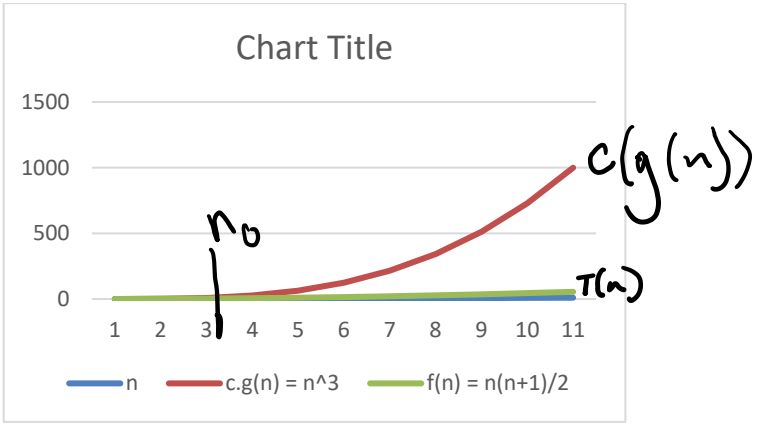
Khrystian Clark

Winter CS325

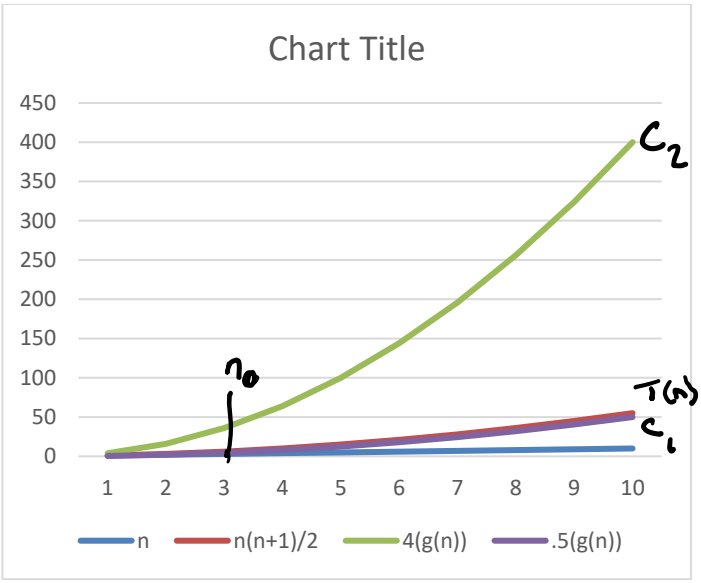
Assignment: Asymptotic Notations and Correctness of Algorithms

[You may include handwritten submission for the parts of the assignment that are difficult to type, like equations, rough graphs etc., but make sure it is legible for the graders. Regrade requests due to the illegible parts of the work will not be accepted.]

1. **Identify and compare the order of growth**: Identify if the following statements are true or false. Prove your assertion using any of the methods shown in the exploration. Draw a rough graph marking the location of c and n_0, if the statement is True. [A generic graph would do for this purpose. You **don't** have to find the values of c and n_0. On the graph you can just write c and n_0 without mentioning their values. The idea is that you know how it looks graphically.].
    a. $n(n+1)/2 \in O(n^3)$ = True
        - $T(n) \in O(g(n))$, if $T(n) \leq c(g(n))$, for all $n \geq n_0$ for positive const c
        - $g(n) = n^3$
        - $f(n) = n(n+1)/2$
        - prove: $T(n) \leq cg(n)$, for all positive c and n
            - $T(n) = f(n)$
            - If $c = 1$, then $c(g(n)) = 1(n^3)$
            - If $n = 6$, then $T(n) = 21$, $c(g(n)) = 216$
            - $21 < 216$
            - Since this holds true for $c = 1$, then by definition of Big-O the statement is true for all values of $c \geq 1$ and $n \geq 1$.

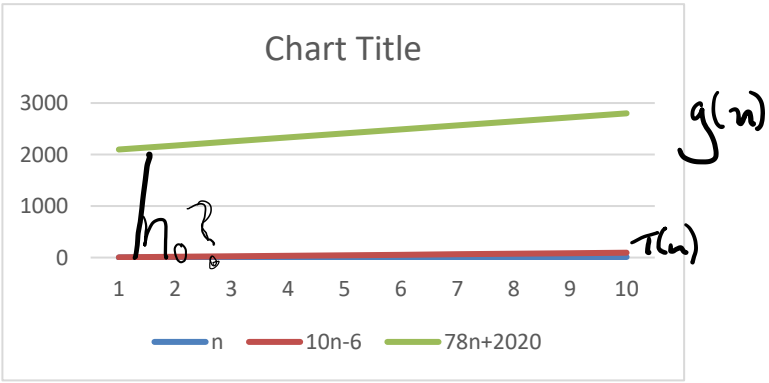


    b. $n(n+1)/2 \in \Theta(n^2)$ = True
        - $T(n) \in \Theta(g(n))$, if $c_1 g(n) \leq T(n) \leq c_2 g(n)$, $\forall\ n \leq n_0$
        - $T(n) = n(n+1)/2$
        - $g(n) = (n^2)$
        - If c1 = 1/2, c2 = 4, while $n \geq 0$, then it creates the graph below
        - This proves by definition of Big Theta, that the statement is true, for values of c1, c2 and n that are greater than 0.



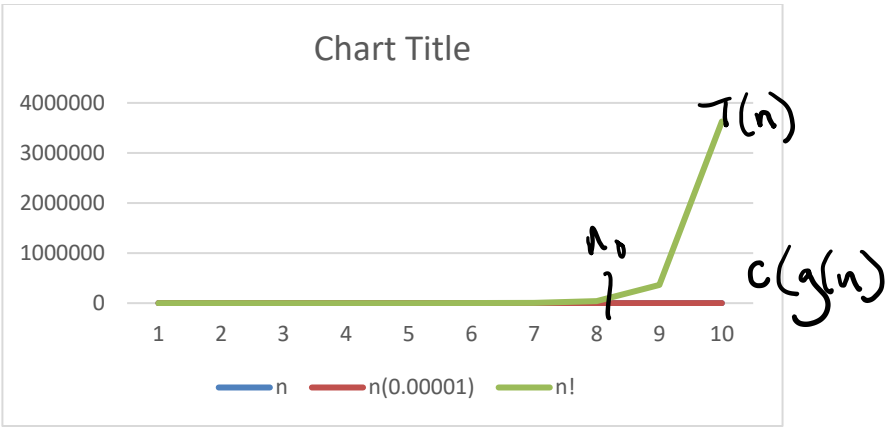
c. 10n-6 ∈ Ω(78n + 2020) = False

- T(n) ∈ Ω(g(n)), if T(n) ≥ c(g(n)) ∀ n ≥ n_0
- T(n) = 10n-6
- g(n) = 78n+2020
- c(78n+2020) ≤ 10n-6, ∀ n ≥ n_0
- There is no point while n ≥ n_0 that T(n) will equal or be greater than c(g(n)). So by definition of Big Omega, the statement is false for all c and n values ≥ 1.



d. n! ∈ Ω (0.00001n) = True

- T(n) ∈ Ω(g(n)), if T(n) ≥ c(g(n)) ∀ n ≥ n_0
- T(n) = n!
- g(n) = (0.00001n)
- For any given value of n > 0, n! is greater than n(0.00001). Therefore by definition of Big Omega, the statement is true.



2. **Read and Analyze Pseudocode:** Consider the following algorithm (In the algorithm, A[0..n-1] refers to an array of n elements i.e. A[0], A[1]… A[n-1])

```
Classified(A[0..n-1]):
    minval = A[0]
    maxval = A[0]
    for i = 1 to n-1:
        if A[i] < minval:
            minval = A[i]
        if A[i] > maxval
            maxval = A[i]
    return maxval – minval
```

   a. What does this algorithm compute?
      - This algorithm computes the maximum and minimum values in a number set or array.
      - It returns the difference between the minimum and maximum values found
   b. What is its basic operation (i.e. the line of code or operation that is executed maximum number of times)?
      - The “for” operation is executed the maximum number times because it iterates through all existing values in the array or set.
   c. How many times is the basic operation executed?
      - It is executed however many instances there are of 2n values, minus 2(for the instances 0 place in the array) because the values run through the two “if” statements. 2(n-1) = 2n-2
   d. What is the time complexity of this algorithm?
      - Since it is looped 2n-2 times, time complexity is O(n) which is linear (based on the chart from the exploration.

3. **Using mathematical induction prove below non-recursive algorithm:**

```
def reverse_array(Arr):
    n = len(Arr)
    i = (n-1)//2
    j = n//2
    while(i>= 0 and j <= (n-1)):
        temp = Arr[i]
        Arr[i] = Arr[j]
        Arr[j]  =  temp
        i = i-1
        j = j+1
```

a. Write the loop invariant of the reverse_array function.
    - At the start of the iteration of the while loop, the loop invariant is i=i–1 : j=j+1.
b. Prove correctness of reverse_array function using induction.
    - Induction: i=i+1, j=j-1. Reverse the loop variant.
    - Need to attend office hours for further understanding of this concept.
    - Initialization: At the start of the initial length of array is what is given to be sorted.
    - Maintenance: Assume the loop invariant is true for the "i"th iteration is true. i=(i-1)//2, j=i//2. i and j can never equal each other, one will be 1 while the other is 0, regardless of the number of iterations. Therefore for any value of i, the loop invariant stands true for (i+1).
    - Termination: The algorithm will terminate when the last element in the array is at the beginning or position [0].

----------(Ungraded question: you can try this question if time permits)----------

Any number greater than 8 can written in terms of three or five.

a. Write a pseudocode of algorithm that that takes a number greater than 8 and returns a tuple (x,y); where x represents number of threes and y represent number of fives make that number
   If number = 8 your pseudocode should return (1,1)
b. Code your pseudocode into python and name your file ThreeAndFive.py

Debriefing (required!): ----------

Report:

Fill the report in the Qualtrics survey, you can access the link here.

(https://canvas.oregonstate.edu/courses/1884866/assignments/8743179?module_item_id=21836732)

Note: 'Debriefing' section is intended to help us calibrate the assignments.