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 n0, if the statement is True. [A generic graph would do for this purpose. You **don’t** have to find the values of c and n0 . On the graph you can just write c and n0 without mentioning their values. The idea is that you know how it looks graphically.].
   1. n(n+1)/2 ∈ O(n^3) **TRUE**

**To prove** big O, we want to get **f(n) ≤ cg(n):**

f(n) = n(n+1)/2

g(n) = n^3

n(n+1)/2 **≤ c** n^3 for all for all n≥ n0

(1/2n2 + 1/2n) **≤** (1/2n2 + 1/4n2) true if 1/4n2 ≥ 1/2n

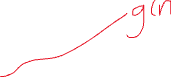
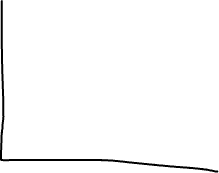
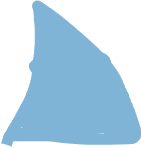
(1/2n2 + 1/2n) **≤** (1/2n2 + 1/4n2) true if 1n2 ≥ 2n

(1/2n2 + 1/2n) **≤** (1/2n2 + 1/4n2) true if n≥ 2

(1/2n2 + 1/2n) **≤** (3/4n2) for all n≥ 2

(1/2n2 + 1/2n) **≤** (3/4n3) for all n≥ 2

So we can choose c = ¾ & n≥ 2



* 1. n(n+1)/2 ∈ Θ(n^2) **TRUE**

½(n2 + n) ∈ Θ(n^2) if c1 g(n)  ≤  T(n)  ≤  c2 g(n)

f(n) = ½(n2 + n)

g(n) = n2

c1 n2  ≤  ½(n2 + n)  ≤  c2 n2

upper bound: ½(n2 + n)  ≤  c2 n2

≤  ½(n2 + n)

≤  ½n2 + ½n

½n2 + ½n ≤  ½n2 + ½n2 true if ½n2 ≥ ½n

½n2 + ½n ≤  ½n2 + ½n2 true if n2 ≥ n

½n2 + ½n ≤  ½n2 + ½n2 true if n ≥ 1

½n2 + ½n ≤  1n2 true if n ≥ 1

c = 1 n=1

Lower bound: c1 n2  ≤  ½(n2 + n)

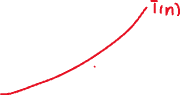
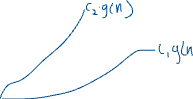
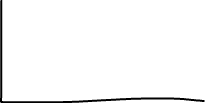
n(n+1)/2 = (n2+ n)/2 = ½ (n2+ n)

½ (n2+ n) ≥ ½ (n2) for all n ≥ 1

c1 = ½ , n = 1

Therefore, n(n+1)/2 ∈ Ω(n2).

Since n(n+1)/2 is both O(n2) and Ω(n2), it follows that n(n+1)/2 ∈ Θ(n2).



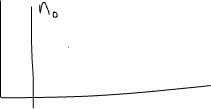
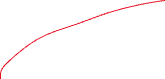
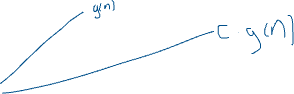
* 1. 10n-6 ∈ Ω(78n + 2020)

f(n) = 10n-6

g(n) = 78n + 2020

10n-6 ≥ 78n + 2020 For all n >= 1

Therefore, 10n-6 ∈ Ω(78n + 2020)



* 1. n! ∈ Ω (0.00001n) **False**

To prove that n! is not Ω(0.00001n), we can use the limit test again:

lim n→∞ (n!) / (0.00001n) = ∞

Since the limit is infinite, n! is not Ω(0.00001n).

1. **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: #performs (n-1)comparisons

if A[i] < minval:

minval = A[i]

if A[i] > maxval

maxval = A[i]

return maxval – \_minval

* 1. **What does this algorithm compute?**

This algorithm computes the difference between the maximum & minimum values in an array A.

* 1. **What is its basic operation (i.e. the line of code or operation that is executed maximum number of times)?**

The basic operation in this algorithm is the comparison between elements of the array to find the minimum and maximum values; A[i] < minval" and "if A[i] > maxval".

* 1. **How many times is the basic operation executed?**

The basic operation is executed **n-1 times** in the for loop, since it compares each element of the array with the current minimum and maximum values.

* 1. **What is the time complexity of this algorithm?**

The time complexity of this algorithm is O(n), since it has a single for loop that iterates n-1 times, and the operations inside the loop take constant time.

1. **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

* 1. Write the loop invariant of the reverse\_array function.
  2. Prove correctness of reverse\_array function using induction.

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.  
b. Prove correctness of reverse\_array function using induction.

Step 1: Hypothesis of loop Invariant

Arr = [7, 9, 3, 2, 1, 5]

Suppose i, j points to the location that holds value (3) which is index 2 in the array

Before iteration of the loop, i & j are: Arr = [i + 1: j-1] is reversed

For this array i & j pointed location index 1 & index 3

Arr = [7, 9, 3, 2, 1, 5]

Arr[1] = 9

Arr[3] = 2

Reverse the element at index i & j

Arr = [7, 2, 3, 9, 1, 5]

We can also say that: newArr = [i+1: j-1] = reverse(oldArr[i+1: j-1])

Where reverse ([]) = []

reverse[a0] = [a0]

reverse(a0, a1,…] )= [reverse([a1,…], a0])

Step 2: prove the loop invariant is inductive

Loop invariant: A[i+1: j-1] is reversed

Base Case: the starting of the loop

j-1 < i+1

Inductive case:

At the start of kth iteration, assume A[i+1: j-1] is reversed,

The loop body swaps A[i] and A[j], decrement I and increment j

Therefore at the start of (k+1)th iteration,

We can prove that A[i+i:j-1] is reversed

Step 3: Prove the correctness using loop invariant:

For the given array array ‘I’ points to -1 (beginning of array)

And j points to n location (end of array)

After the loop termination, i = -1 & j =n

Loop invariant tells us that A[i+1:j-1] is reversed

Therefore A[0: n-1] is reversed.

* 1. a. Write the loop invariant of the reverse\_array function.
  2. b. Prove correctness of reverse\_array function using induction.

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

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

1. 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)

1. b. Code your pseudocode into python and name your file ThreeAndFive.py