CS325 Spring 2021
04/02/21
Matthew Wong
Assignment: Asymptotic Notations and Correctness of Algorithms

1. Identify asymptomatic notation of a functio:
   a. n(n+1)/2 ∈ O(n^3)

      $n \leq n^2,\ for\ all\ n \geq 1$ (obvious)

      $0.5n^2 + 0.5n \leq 0.5n^2 + 0.5n^2,\ for\ all\ n \geq 1$multiply both sides and add the n^2

      $0.5n^2 + 0.5n \leq 1n^2,\ for\ all\ n \geq 1$

      So c = 1 n0 = 1

      Substitute back in:

      $n(n+1)/2 = c(n^3)$

      $1(1+1)/2 = 1(1^3)$

      TRUE proves $n(n+1)/2 \in O(n^3)$

   b. $n(n+1)/2 \in \odot(n^2)$

      $f(n) = n(n+1)\ and\ g(n) = n^2$

      => $0.5n^2 + 0.5n \leq c1(n^2)\ while\ n > k$

      Choose a value for c1 = 5

      => $0.5n^2 + 0.5n \leq 5(n^2)\ while\ n > k$

      Choose a value for k = 5 we test n = 10

      => $0.5(10^2) + 0.5(10) \leq 5(10^2)$

      $=> 55 < 500\ True\ part\ 1$

      Part 2:

      $fn \geq c2 * g(n)\ while\ n > k$

      $=> 0.5n^2 + 0.5n \geq c2 * n^2\ while\ n > k$

      Choose a value for c1 = 0.5

      $=> 0.5n^2 + 0.5n \geq 0.5n^2\ while\ n > k$

      Choose value for k = 5 we test n = 10

      => $0.5(10^2) + 0.5(10) \geq 0.5(10^2)\ TRUE\ part\ 2$

      TRUE proves $n(n+1)/2 \in \odot(n^2)$

   c. $n(n+1)/2 \in \odot(n^3)$

      $0.5n^2 + 0.5n \leq c1 * n^3$

      Choose value for c1 = 5

      $=> 0.5n^2 + 0.5n \geq 5n^3$while n>k

      Choose value for k =10 test n = 100

$0.5(100)^2 + 0.5(100) \leq 5(100^3)$
5000≤5,000,000 True part 1

Part 2:
$0.5n^2 + 0.5n \geq \odot c2(n^3)$
Choose value for c2 = 5
=> $0.5(n^2) + 0.5n \geq 5(n^3)\ while\ n > k$
Choose value for k =10 test n = 100
=> $0.5(100)^2 + 0.5(100) \geq 5(100)^3$
=>5000+500 ≥ 5,000,000 FALSE part 2
FALSE proves $n(n+1)/2 \in \odot(n^3)$

d. n(n+1)/2∈Ω(n)
Pick values for c = 2 k = 5
=>$0.5n^2 + 0.5n \geq 2n$whenever n≥5
=> $0.5(5)^2 + 0.5(5) \geq 2(5)$
=> $5 + 2.5 \geq 10\ FALSE$
Ok lets try something else for n since it can be greater than or equal
=> $0.5(10)^2 + 0.5(10) \geq 20\ whenever\ n \geq 5$
=> 55 ≥ 20 TRUE

True proves n(n+1)/2∈Ω(n)

2. Compare order of growth:
   a. $f(n) = 10n - 6,\ g(n) = 12345678n + 2020$
      i. $f(n) = \odot g(n)$[theta]
      ii. $f(n) = Og(n)\ [bigO]$
      iii. $10n - 6 \leq c * (12345678n + 2020)$
      => c≥1, n ≥1
      iv. $f(n) = \Omega g(n)$ [big omega]
      v. $10n - 6 \geq c * (12345678n + 2020)$
      =>c≥$1/12345678$ whenever n≥1
      answer:$f(n) = \odot g(n)$
   b. $f(n) = n!,\ g(n) = 0.00001n$
      i. $f(n) = n! = n(n-1)(n-2)(n-3)\ldots 2 * 1$
      ii. $f(n) = n^2,\ g(n) = 0.00001n$
      iii. $f(n) > g(n)$
      iv. $n^2(n!)\ is\ asymptotically > 0.00001n$
      v. $f(n) = \Omega g(n)\ [big - omega]$
      vi. $f(n) \geq c * g(n)$
      vii. $n^n \geq c * (0.00001n)$

answer$f(n) = \Omega$**g(n)**

c. $f(n) = n^{1.34}, g(n) = log(n)$

i. $f(n) = n^{1.34}, g(n) = log(n)$

ii. $n^{1.34}(poly) = asymptotically > logn(logarithm)$

=>$n^{1.34} > logn$

=>$n^{1.34} = \Omega log(n)$

=>$n^{1,34}$=c*logn

Answer: $f(n) = \Omega g(n)$

d. $f(n) = n + n^4, g(n) = n^3 + log\, n$

i. $f(n) = n + n^4, g(n) = n^3 + log(n)$

ii. $f(n) = n + n^4 = n^4\,(upper)$

iii. $g(n) = n^3 + log(n) = n^3(upper)$

=>f(n) > g(n)

$n^4 is\ asymptotically > n^3$

=>$n + n^4 \geq c(n^3 + log(n))$

Answer: $f(n) = \Omega g(n)$

3. Identify asymptomatic notation of an operation
   a. Array
      i. $\odot(log(n))$
      ii. N is the size of array since the array is sorted we have direct access to via the index. More efficient searches can be applied like binary. Searches can happen in log(n) since we are doing binary search
   b. Linked list
      i. On
      ii. Linked list unlike arrays need to parse each one and check every single node
4. Read and Analyse Pseudocode:
   a. What does this algorithm compute:
      i. Difference between the biggest and smallest elements within array
   b. What is its basic operation:
      i. Our loop using for executes max times
   c. How many times is the basic operation executed?
      i. N-1 times
   d. What is the time complexity of this algorithm?
      i. O(n) linear
5. Using mathematical induction prove below non-recursive algorithm:
   a. In: N(>=8)
      Out: (x,y) where 3x+5y = n
      Function(n){
      If N == 8

return(1,1)
If N ==9
Return (3,0)
If n == 10
Return (0,2)
(x1,y1) < Function(n-3)
Return (x1+1, y1)
}

b. Base case: when n= 8, 9, & 10
8= 3 +5, 9 = 3+3+3, 10 = 5+5
=> base case can be performed with 3, 5
Induction:
If for some k>=10,8,9,10…,k can all be written in sums of 3, 5
Need to prove that k+1 can be written in just 3, 5s
Now k+1-3 = k-2 ≥ 8 (this is for k≥10)
That means k+1-3 can be rewritten with just sums of 3s and 5s
Lets substitute 3 and 5s with a & b in k-2
=> k-2 = 3a+5b
K+1 = k-2+3 =3a+5b+3=3(a+1)+5b
=> k+1 can be rewritten as a+1 and 3s and b 5s
Answer: any number ≥ 8 can be written in forms sums of 3s, and 5s

c. Since recursion is applied calculating the time complexity is as follows:
i. t(n) = t(n-3)+1
ii. t(n) = t(n-6) +1 +1
iii. t(n-9)+1+1+1
=...
iv. t(n-(n-8)) + 1 +1 +1 +...+1(n-8)/3
v. 1+1+1+...+1
vi. On
vii. => **t(n) = On**

d. See file ThreeAndFive.py

6. Visualizing time complexity for best case, worst case and average case:
See code in evluateinsertionsort.py

E. see the sheets attached with graph info

F. Observation is that with random order we had the longest times with ascending being middle and descending having the shortest amount of time.

Debriefing:

- Spent 20 hours on the homework
- I would rate it as difficult. I needed a lot more time understanding python syntax having only coded in C or HTML beforehand.

- I feel like i understand the material 60%