WEEK1: ASYMPTOTIC NOTATIONS AND CORRECTNESS OF ALGORITHMS

Agenda

Compare Order of Growth

Mathematic Analysis of Algorithms

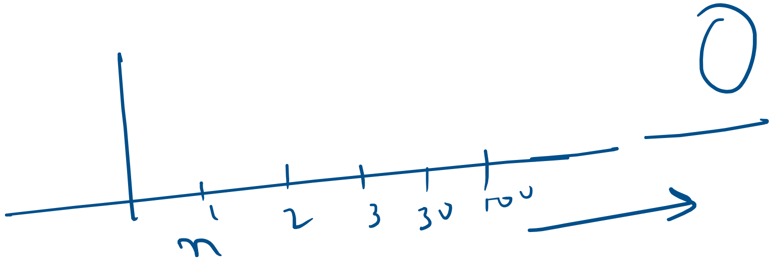
Proving Correctness

Asymptotic Analysis

Math Summation basics + other comments

Compare Order of Growth

- *Would like a walk through on how to determine the best way to prove an algorithm's asymptotic notation.*



- Two methods

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} \to \infty,\ 0,\ c$$

$f(n) = O(g(n))$, Θ, Ω

$$\lim_{n\to\infty}\left(\frac{n}{5}\right) = \frac{\infty}{5} = \infty$$

$$\lim_{n\to\infty}\left(\frac{1}{n}\right) = \frac{1}{\infty} = 0$$

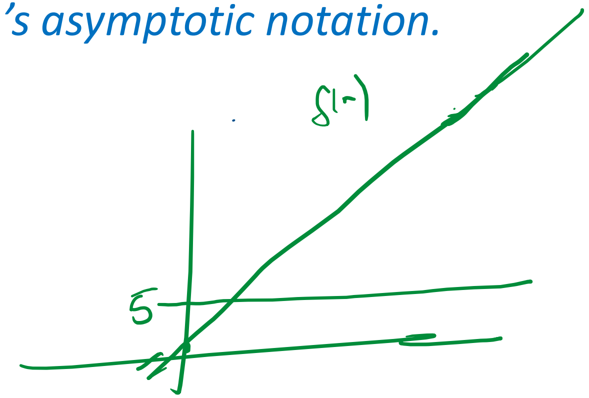
$$\lim_{n\to\infty}\left(\frac{n}{n}\right) = \lim_{n\to\infty}(1) = 1$$

$f(n) = n$

$g(n) = n$

$\frac{1}{10} = 0.1$ $\frac{1}{100} = 0.01$

$\frac{1}{10^5} = 0.00001$



Math

- Math concepts (including derivatives)

$T(n) = n^{10}$ $= 10n^9$

$T(n) = \log n$ $\frac{1}{n} \cdot \frac{1}{\log_e 2}$

$T(n) = \sqrt{n}$ $n^{1/2} = \frac{1}{2} n^{1/2 - 1} = \frac{1}{2\sqrt{n}}$

Compare Order of Growth

$f(n)= 0.01n^3$; $g(n) = 50n+10$

$f(n)= 0.01n^3$; $g(n) = 50n+10$

$f(n) = ?(g(n))$

$$\underset{n\to\infty}{Lt}\left(\frac{0.01n^3}{50n+10}\right) =$$

$$\underset{n\to\infty}{Lt}\left(\frac{0.03n^2}{50}\right)$$

$$\frac{0.03}{50}\left(\underset{n\to\infty}{Lt}(n^2)\right) = \infty$$

$\times \infty$

$(0.01n^3)' =$

$0.01 \times 3\, n^2$

$(50n)' + (10)'$

$= 50 + 0 = 50$

$(10)'$

$= 0$

A. O

B. Θ

C. Ω

D. None

$$\lim_{n\to\infty} \frac{T(n)}{g(n)} = \begin{cases} 0 & \text{implies that } T(n) \text{ has a smaller order of growth than } g(n). \quad T(n) \in O(g(n)) \\ C & \text{implies that } T(n) \text{ has the same order of growth as } g(n). \quad T(n) \in \theta(g(n)) \\ \infty & \text{implies that } T(n) \text{ has a larger order of growth than } g(n). \quad T(n) \in \boldsymbol{\Omega}(g(n)) \end{cases}$$

Compare Order of Growth

- $\log n$ vs n

A. O

B. Θ

C. Ω

D. None

$$\lim_{n\to\infty}\left(\frac{\log n}{n}\right) = \lim_{n\to\infty}\frac{1/n}{1} \times c$$

$$= c \cdot \lim_{n\to\infty}\left(\frac{1}{n}\right)$$

$$\left(\frac{n}{\log n}\right)$$

$$\frac{1}{1/n} \quad = \frac{1}{\infty} = \underline{\underline{0}}$$

$$\text{£}\, n) = \infty$$

$$\log n = O(n)$$

$$\lim_{n\to\infty}\frac{T(n)}{g(n)} = \begin{cases} 0 & \text{implies that } T(n) \text{ has a smaller order of growth than } g(n). \quad T(n) \in O(g(n)) \\ C & \text{implies that } T(n) \text{ has the same order of growth as } g(n). \quad T(n) \in \theta(g(n)) \\ \infty & \text{implies that } T(n) \text{ has a larger order of growth than } g(n). \quad T(n) \in \boldsymbol{\Omega}(g(n)) \end{cases}$$

Compare Order of Growth

① $\log n^2$ vs n

$\lim \frac{2\log n}{n} = \lim \frac{1}{n}\cdot 2 = 0 \Rightarrow \boxed{f(n) = O(g(n))}$

② n^2+n vs n^2

$\boxed{f(n) = \Theta(g(n))}$

③ $\sqrt{n}\log n$ vs n

$\lim \frac{\log n}{\sqrt{n}} = \lim \frac{1}{n}\sqrt{n} = \lim \frac{1}{\sqrt{n}} = 0$

$\boxed{f(n) = O(g(n))}$

A. O

B. Θ

C. Ω

D. None

$$\lim_{n\to\infty} \frac{T(n)}{g(n)} = \begin{cases} 0 & \text{implies that } T(n) \text{ has a smaller order of growth than } g(n). \quad T(n) \in O(g(n)) \\ c & \text{implies that } T(n) \text{ has the same order of growth as } g(n). \quad T(n) \in \theta(g(n)) \\ \infty & \text{implies that } T(n) \text{ has a larger order of growth than } g(n). \quad T(n) \in \boldsymbol{\Omega}(g(n)) \end{cases}$$

Compare Order of Growth

- . *I would like some more examples using Stirling's formula to identify and compare growth order.*

Use Stirling's Formula $n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$ for large values of n.

Or

Factorial expansion = n*(n-1)*(n-2)….1

Compare Order of Growth

$n!$ vs n^2

$$\lim_{n\to\infty} \frac{n!}{n^2} = \frac{\cancel{n}\boxed{(n-1)(n-2)\cdots}}{n^{\cancel{2}}} =$$

$n^2 \times n^3 = n^5$

$$\frac{n^n}{n^2} = n^n \times n^{-2} = n^{n-2}$$

$$= \lim_{n\to\infty} \frac{\sqrt{2\pi n}\left(\frac{n}{e}\right)^n}{n^2} = \frac{(\sqrt{n})(n^n)}{n^2 e^n} = \infty$$

2×10

$2 \times (100{,}000)$

$$\underset{n\to\infty}{=} (\sqrt{n})(n^{n-2})$$

$$\boxed{\sqrt{\infty}\cdot\infty^{\infty}} \Rightarrow \infty$$

$$\lim_{n\to\infty} \frac{T(n)}{g(n)} = \begin{cases} 0 & \text{implies that } T(n) \text{ has a smaller order of growth than } g(n). \quad T(n) \in O(g(n)) \\ C & \text{implies that } T(n) \text{ has the same order of growth as } g(n). \quad T(n) \in \theta(g(n)) \\ \infty & \text{implies that } T(n) \text{ has a larger order of growth than } g(n). \quad T(n) \in \Omega(g(n)) \end{cases}$$

Compare Order of Growth

- *Clarification on comparing growth order of O, Omega and Theta*
- *Some examples of working through comparing growth rate for the different notations would be very helpful, I've never seen these types of problems before and could use the extra examples!*

Proving Correctness

a couple of walk-throughs on proofs of correctness would be very helpful

- loop invariants

```
def findMax(list):
    max = list[0]
    for i from 0 to len(list)-1:
        if list[i] > max:
            max = list[i]
    return max
```

(1) EX

(2) max, i Before in the loop
After

(3) check

	0	1	2	3	4
	10	20	30	40	15
max : 10	10	20	30	40	40
i = 0	0	1	2	3	4

40

max holds maximum until pos of loop variable i

Proving Correctness

answer holds the sum of i, in the range of length of A

- loop invariants

answer will contain the total of array elements A up to A[i]

```
def sum(A):

    answer = 0

    for i in range(len(A)): # in pseudo-code for i=0,...,len(A)-1

        answer += A[i]

    return answer
```

(ans + A[k+1])

answer =

Answer will hold the sum of array members up to the position A[I]

Loop invariant: answer is always the running sum

the variable answer will hold the sum of values in the arr[0..i]

Proving correctness

- *Some review examples of mathematical induction to prove an algorithm*
- *overview of proving correctness of an algorithm*

Loop invariant Ready

Base case: Start of each iteration Loop invariant holds true

Inductive Case: Assume loop invariant holds true for k iterations. Show that it holds true after (k+1)th iteration

Termination: Show that you get your answer after the loop terminates.

Check

0

Exit

0 . . . k

$\to$ i = 0

k

answer!

k+1th

k+1th = ✓

$\to$ n

Proving correctness

- *Some review examples of mathematical induction to prove an algorithm*
- *overview of proving correctness of an algorithm*

Loop invariant: `answer` is always the rolling sum of the numbers in the array up to the current index.

Initialization (base case): `answer` is initialized at 0 and, when the loop is initialized at i=0, it is added to the current value of `answer` and thus, the loop has the current index at 0 and `answer` is the sum of 0 + A[i] and is the rolling sum of the numbers in the array up to the current index.

Maintenance (inductive case): Let `answer` be the rolling sum of all the numbers in the array up to the i^th iteration. When the loop runs for i^th + 1 iteration, it will add the values of `answer` to A[i^th + 1] and `answer` will hold the rolling sum for all numbers in the array up to the current index and our maintenance step holds.

Termination: i is incremented for every loop and end at an value of i that is equal to len(A) - 1 (the last index position in the array) and does not continue and `answer` is returned containing the sum of all number in the array up to the current index which is the solution to the function we were given.

Proving Correctness

- *in the lecture about loop invariance it says, "When the execution terminates, the invariant gives us a useful property that helps show that the algorithm is correct." Exactly what property shows the algorithm is correct? why is it for induction proofs we can use the previous values up to k to determine that the next value also behaves in the same way. e.g. in the insertion sort example, since the subarray is sorted up to j, then j+1 will also become sorted. maybe that's not the right way to think about induction?*

Source: https://www.youtube.com/watch?v=fFpk4BBEDuY

Proving Correctness

Please provide more examples on identifying loop invariants and proving correctness of algorithms

```
def binary_search(arr, x):
    low = 0
    high = len(arr) - 1
    mid = 0

    while low <= high:

        mid = (high + low) // 2

        # If x is greater, ignore left half
        if arr[mid] < x:
            low = mid + 1

        # If x is smaller, ignore right half
        elif arr[mid] > x:
            high = mid - 1

        # means x is present at mid
        else:
            return mid
```

0	1	2	3	4	5	6
-7	11	13	32	41	42	43

$\boxed{x = 13}$

Code Source: https://www.geeksforgeeks.org/python-program-for-binary-search/

Proving Correctness

0	1	2	3	4	5	6
-7	11	13	32	41	42	43

$\boxed{x=13}$

```
def binary_search(arr, x):
    low = 0
    high = len(arr) - 1
    mid = 0

    while low <= high:

        mid = (high + low) // 2

        # If x is greater, ignore left half
        if arr[mid] < x:
            low = mid + 1

        # If x is smaller, ignore right half
        elif arr[mid] > x:
            high = mid - 1

        # means x is present at mid
        else:
            return mid
```

Loop Invariant: After execution of each iteration low and high range contain ‘x’

Proving correctness

- *Some review examples of mathematical induction to prove an algorithm*
- *overview of proving correctness of an algorithm*

After execution of each iteration low and high range contain ‘x’

Loop invariant Ready

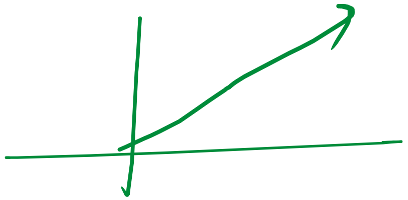
Base case: Start of each iteration Loop invariant holds true

Inductive Case: Assume loop invariant holds true for k iterations. Show that it holds true after (k+1)th iteration

Termination: Show that you get your answer after the loop terminates.

Analysis of Algo

n



- whats the difference between running time and order of growth

- Could we have more practice examples on how to analyze the time complexity of algorithms?

Mathematical Analysis of Algorithms

- how to identify basic operations

```
IsUnique(A[0..n − 1]):
    for i =0 to n − 2
        for j = i + 1 to n − 1
            if A[i]= A[j]
                return false

    return true
```

```
foo2(n):
    sum = 0
    for i in range(n):
        for j in range(n):
            if( i == j):
                for k in range(n*n):
                    sum = i + j + k
```

for
if ( )
if ( )

O θ Mathematical Analysis of Algorithms

- *when should you categorize an algorithm with Θ and when should you categorize an algorithm with big O?*
- *Can you discuss in greater detail how we can take a growth order from a program algorithm and determine if it represents an O, Omega, or Theta notation. I have been confused by how what I would think is an O(n) is expressed to be a Theta(n) growth order.*
- *What is big theta for? When determining the run time of an algorithm mathimatically such as in Mod 1 Mathematical Analysis - is the result always big theta? In the book these proofs were also big theta. I am more comfortable with big O, so I'm trying to understand when I would want to know big theta.*

```
IsUnique(A[0..n − 1]):
  for i =0 to n − 2
      for j = i + 1 to n − 1
          if A[i]= A[j]
              return false

  return true
```

(1, 1, 2, 10, 20)

$i \neq j$

n^2 [1, 2, 3, 4]

n^2

worst case ≠ *average case* ≠ *best case*

n^2 avg const

$= O(n^2)$

O

```
def findMax(list):
    max = list[0]
    for i from 0 to len(list)-1:
        if list[i] > max:
            max = list[i]
    return max
```

worst case *average case* *best case*

$\Theta(n)$

Mathematical Analysis of Algorithms

- *I would love to see a couple more walk-throughs on analysis of non-recursive algorithms (especially the simplification of the inner summations seemed a bit tricky).*
- *Further practice with mathematical analysis of algo*

Find time complexity of the following foo function.

```
foo(n):
    j = 0
    while( j < n ):
        j = j + 2
        z = z + 1
```

A. $O(n)$

B. $O(n^3)$

C. $O(n^3 \log n)$

D. None (I'm not solving this)

j: 0, 0+2, 0+2+2 . . . . 0+2+2-. k times

count: $\underbrace{1, 1, 1 \ldots 1}_{k\ 1\text{'s}}$

Time complexity $= \theta(k)$

$k = ?$

$$0 + \underbrace{2 + 2 + 2 \ldots}_{k \text{ times}} = n$$

$$2(\underbrace{1+1+1+\ldots 1}_{k \text{ times}}) = n$$

$$2k = n \Rightarrow \boxed{n \approx k}$$

Mathematical Analysis of Algorithms

Find time complexity of the following foo function.

```
foo2(n):
    sum = 0
    for i in range(n):
        for j in range(n):
            if( i == j):
                for k in range(n*n):
                    sum = i + j + k
```

i: 0 to n

j: 0 to n

[k: 0 to n^2] - only if i=j

A. $O(n^4)$

B. $O(n^3)$

C. $O(n^3 \log n)$

D. None (I'm not solving this)

How many times is i=j ?

'n' times

Mathematical Analysis of Algorithms

1.
```
def foo(Array[], rand, pos):
    val = Array[pos]/rand
    return val
```
— c

2.
```
def foo(Array[], len):
    num = [1,2,3,4,...10]

    #two nested for loops.. n^2
```
n^2

3.
```
def foo():
    num = [1,2,3,4,...10]

    #two nested for loops.. n^2

    #one for loop ..n
```
— c

$\cancel{c} + n^2 + \cancel{n}$

$O(n^2)$

Mathematical Analysis of Algorithms

- *calculating running time formulas on different types of pseudocode and (b) help identifying the common patterns such as when you have some of these scenarios: -*
    - *for loop only, all other code is expression*
    - *for loop > if block > expression*
    - *- for loop > if block > for loop > if block > expression*
    - *- if block > for loop > for loop :*
    - *^ doesn't have to be those, but those are some specific examples that come to mind that I would benefit from (a) repetition an (b) hearing the though process as each step is counted*

for ( )

for ( )

if ( )

for ( )

Asymptotic Analysis

- *can you go over proving the lower bounds for Θ bounds for a function 1/2n(n-1) is in Θ(n^2)? calculating the lower bounds was kind of confusing.*

Proving Lower Bounds

$$\tfrac{1}{2}n\,(n-1) \;=\; \tfrac{1}{2}n^2 \;-\; \tfrac{1}{2}n$$

$$\Longrightarrow \tfrac{1}{2}n\,(n-1) \;\geq\; \tfrac{1}{2}n^2 \;-\; \left(\tfrac{1}{2}n\right)\left(\tfrac{1}{2}n\right)$$

$$\Longrightarrow \tfrac{1}{2}n\,(n-1) \;\geq\; \tfrac{1}{4}n^2 \quad \textit{for all } n \geq 2$$

Wondering, how did we get $n \geq 2$?

Take line $\tfrac{1}{2}n\,(n-1) \;\geq\; \tfrac{1}{2}n^2 \;-\; \left(\tfrac{1}{2}n\right)\left(\tfrac{1}{2}n\right)$

$$\Longrightarrow \tfrac{1}{2}n^2 \;-\; \tfrac{1}{2}n \;\geq\; \tfrac{1}{2}n^2 \;-\; \left(\tfrac{1}{2}n\right)\left(\tfrac{1}{2}n\right)$$

For this to work, following needs to be true:

$$\tfrac{1}{2}n \cdot \tfrac{1}{2}n \;\geq\; \tfrac{1}{2}n$$

Which gives, $\tfrac{1}{2}n \;\geq 1$

$$\Longrightarrow \; n \;\geq 2$$

Hence we have $\tfrac{1}{4}n^2 \;\leq \tfrac{1}{2}n\,(n-1) \;\leq \tfrac{1}{2}n^2 \quad \textit{for all } n \geq 2$

$\cancel{n^2}$

$$\tfrac{1}{2}n(n-1) \geq c \cdot n^2$$

$$\tfrac{1}{2}n(n-1) = \tfrac{1}{2}n^2 - \tfrac{1}{2}n$$

$$\tfrac{1}{2}n$$

$$\geq \tfrac{1}{2}n^2 - \tfrac{1}{4}n^2$$

$$\geq \tfrac{1}{4}n^2$$

n_0

$$\tfrac{1}{4}n^2 \geq \tfrac{1}{2}n$$

$$n \geq 2$$

Compare Order of Growth

- finding c and n0 for proving 0, omega, theta.; Also, a deep look at maintaining inequalities while solving equations.; Show to solve questions similar to question one on the homework specifically c and d like questions.; Proving bounds for big O, theta, and omega.

For lower bound:

- remove a positive term from f(n)

$$3n^2 + 5n \geq 3n^2$$

- multiply a negative lower order term in f(n)

$$5n^2 - 2n \geq 5n^2 - 2n \times n = 3n^2$$

- split a higher order term if needed

$$2n^3 + 1 = n^3 + n^3 + 1 \geq n^3$$

For upper bound:

- remove a negative term in f(n)

$$5n^2 - 3n \leq 5n^2$$

- multiply a positive lower order term in f(n)

$$3n^2 + n \leq 3n^2 + n \times n = 4n^2$$

Let us prove $5n + 100 \in O(n^2)$.

Take higher order terms that we want to prove

1. $n \leq n^2$ $\times 5$

$5n \leq 5n^2$ $+100$

$5n + 100 \leq 5n^2 + 100 \times n^2$

$\leq 5n^2 + 100n^2$

$\leq 105 \times n^2$ C

Math

- *Summation simplifications*

$$\sum_{j=1}^{n} j = 1+2+3+\cdots n =$$

$$\sum_{j=1}^{n} 1 = \underbrace{\overset{j=1}{1} + \overset{j=2}{1} + \overset{j=3}{1} + \cdots + \overset{j=n}{1}} = n$$

$$\sum_{j=1}^{n} i = \overset{j=1}{i} + \overset{j=2}{i} + \overset{j=3}{i} + \cdots \overset{j=n}{i} = i[1+1+\cdots 1] = in$$

$$= i\sum_{j=1}^{n} 1$$

Math

- *nested Summations*
- Math concepts (including derivatives and summation)

$$T_{worst}(n) = \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1$$

$$\sum_{j=i+1}^{n-1} 1 = \underbrace{1 + 1 + \cdots + 1}_{}$$

$j=i+1 \quad j=i+2 \quad j=n-1$

$j=3,\ j=4,\ 5 = 1+1+1 = 3$

$(5-3+1) =$

$= (n-1-(i+1)+1)$

$= n-1-i-\cancel{1}+\cancel{1})$

$= (n-1-i)$

$$\sum_{i=0}^{n-2} (n-1-i) = \sum_{i=0}^{n-2} n \quad \sum (n) + \sum (-i)$$

$$T(n) = \sum_{i=0}^{n} \sum_{j=0}^{m} 1$$

$j = 0 \quad 1 \quad 2 \quad \ldots \quad m$

$1 \quad 1 \quad 1 \quad \quad 1 \quad = (m+1) \approx m$

$$= \sum_{i=0}^{n} (m+1) = (m+1) \sum 1 = (m+1)(n+1)$$

$\approx n$

$O(mn$