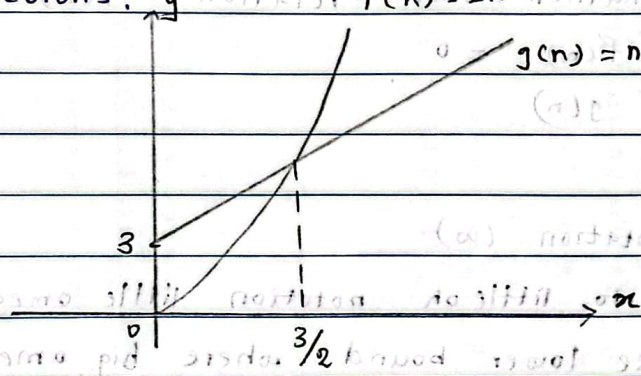
Date: No:

CS 2023 - Take Home Assignment

Question -01

Big Omega notation (Ω).

- Omega notation represents the lower bound of the running time of an algorithm. Therefore, this provides the best case time complexity for a given algorithm.
- Let g, f be 2 functions. If $f(n) = \Omega(g(n))$ we can say that there exist positive constants c and n_0 such that $c.g(n) \leq f(n)$ for all $n \geq n_0$.
- For an instant let $f(n) = 2n^2$ and $g(n) = n + 3$. Given below shows the graphical design of given 2 functions.



From the above graph we can say that when $c = 1$ and $n_0 = 3/2$, $\forall n \geq n_0 = 3/2$, $n + 3 \leq 2n^2$.

Thus, $2n^2 = \Omega(n+3)$.

So any function $f(n)$ which satisfy $c.g(n) \leq f(n)$ $\forall n \geq n_0$, where c is a positive constants, $f(n)$ can be said $\Omega(g(n))$ or $f(n)$ belongs to $\Omega(g(n))$.

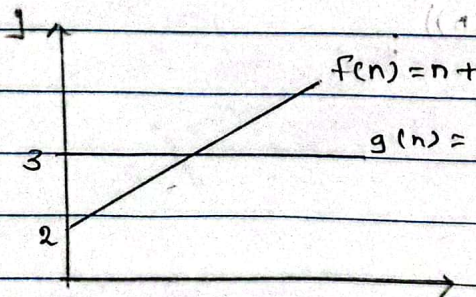
Date: No:

Little oh notation (o)

- Big Oh is used as a tight upperbound on the growth of an algorithm whereas little oh notation is used to describe an upper bound that cannot be tight, in other words loose upperbound
- Let $f(n)$ and $g(n)$ be 2 different functions $f(n) = o(g(n))$ is true when for any real constant c, there exists an integer n_0 such that for all $n \geq n_0$ $f(n) < c.g(n)$
- In the previous example if $g(n) = 2n^2$ and $f(n) = n+3$ for $c=1$ and $n_0 = 4$,
  $\forall n \geq n_0 = 4$, $n+3 < 2n^2$
  ie $f(n) < c.g(n)$
- In mathematical relation $f(n) = o(g(n))$ means
  $$\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$$

Little omega notation (ω)

- Similar to little oh notation little omega notation represents the loose lower bound where big omega represents the tight lower bound
- Let f and g be 2 functions. $f(n)$ is $\omega(g(n))$ or $f(n) \in \omega(g(n))$ is true when for any positive constant c, there exists a positive integer n_0 such that $\forall n \geq n_0$ $c.g(n) < f(n)$
- For example let $f(n) = n+2$, and $g(n) = 3$



$f(n) = n+2$ for $c=1$ and $n_0 = 4$,
$g(n) = 3$ $\forall n \geq n_0 = 4$,
$1(3) < n+2$
ie $c.g(n) < f(n)$

In mathematical relationship $\Rightarrow \lim_{n \to \infty} \frac{f(n)}{g(n)} = \infty$.

Question 03

Θ (Theta)	O (Big Oh)	o (Little oh)	Ω (Big omega)	ω (little omega)
$\Theta(g(n)) =$ $\{f(n)$: $c_1, c_2 > 0$, $n_0 > 0$, $n_0 \in \mathbb{N}$ $0 \le c_1 g(n) \le f(n) \le c_2 g(n)$ $\forall n \ge n_0\}$	$O(g(n)) =$ $\{f(n)$: $c, n_0 > 0$ $n_0 \in \mathbb{N}$ $0 \le f(n) \le c \cdot g(n)$ $\forall n \ge n_0\}$	$o(g(n)) =$ $\{f(n)$: $c, n_0 > 0$ $n_0 \in \mathbb{N}$ $0 \le f(n) < c \cdot g(n)$ $\forall n \ge n_0\}$	$\Omega(g(n)) =$ $\{f(n)$: $c, n_0 > 0$ $n_0 \in \mathbb{N}$ $0 \le c \cdot g(n) \le f(n)$ $\forall n \ge n_0\}$	$\omega(g(n)) =$ $\{f(n)$: $c, n_0 > 0$ $n_0 \in \mathbb{N}$ $0 \le c \cdot g(n) < f(n)$ $\forall n \ge n_0\}$
Asymptotic tight bound	Asymptotic upper bound (It may or may not be tight)	Asymptotic loose upper bound	Asymptotic lower bound (It may / may not be tight)	Asymptotic loose ~~upper~~ lower bound
Represents exact complexity. (Because this is asymptotically tight).	May represent exact complexity used for worst case time complexity.	This doesn't give the exact time complexity but provides a rough estimation of worst case complexity but not asymptotically tight	Used for best case complexity This may represent the exact complexity.	Rough estimation of best case complexity but not asymptotically tight
	$\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$.			$\lim_{n \to \infty} \frac{f(n)}{g(n)} = \infty$

Aaditya

Date: No:

Relations between Θ, Ω, O

- For any 2 functions $g(n)$ and $f(n)$,

$f(n) = \Theta(g(n))$ iff $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$

That is,

$\Theta(g(n)) = O(g(n)) \cap \Omega(g(n))$

Question 03.

(01) Version I

Worst Case

```
for j = A.length to 2 do                          Cost    Time
    swapped = false                               c1      n
    for i = 2 to j do                             c2      n-1
        ~~swapped = false~~                       c3      n(n+1)/2 - 1
        if (A[i-1] > A[i]) then                   c4      n(n-1)/2
            temp = A[i]                           c5      n(n-1)/2
            A[i] = A[i-1]                         c6      n(n-1)/2
            A[i-1] = temp                         c7      n(n-1)/2
            swapped = true                        c8      n(n-1)/2
    if (!swapped) then                            c9      n-1
        break;                                    c10     0
n = newLimit
```

Worst Case - Reverse sorted array

Let n = A.length

$$T(n) = c_1 n + c_2(n-1) + c_3\left(\frac{n^2+n}{2} - 1\right) + c_4\left(\frac{n^2-n}{2}\right) + (c_5 + c_6 + c_7 + c_8)\left(\frac{n^2-n}{2}\right) + c_9(n-1) + c_{10} \cdot 0$$

$$= \left(c_3 + c_4 + c_5 + c_6 + c_7 + c_8\right)\frac{n^2}{2} + \left(c_1 + c_2 + c_3 - c_4 - c_5 - c_6 - c_7 - c_8\right)\frac{n}{2} - c_2 - c_9 - c_3$$

$$= c_{11} n^2 + c_{12} n + c_{13}$$

$$= \Theta(n^2)$$

Araliya

Date..... No:.....

Version 2

Worst case

	Cost	Time
n = A.length		
do	c_1	1
swapped = false	c_2	n
for i = 2 to n do	c_3	n
if (A[i-1] > A[i]) then	c_4	$\frac{n(n+1)}{2} - 1$
temp = A[i]	c_5	$\frac{n(n-1)}{2}$
A[i] = A[i-1]	c_6	$\frac{n(n-1)}{2}$
A[i-1] = temp	c_7	$\frac{n(n-1)}{2}$
swapped = true	c_8	$\frac{n(n-1)}{2}$
newLimit = i-1	c_9	$\frac{n(n-1)}{2}$
n = newLimit	c_{10}	$\frac{n(n-1)}{2}$
while swapped	c_{11}	n
	c_{12}	n

Worst case reverse sorted array

$$T(n) = c_1 + nc_2 + nc_3 + \left(\frac{n^2+n}{2} - 1\right)c_4 + \frac{(n^2-n)}{2}c_5 + c_6\left(\frac{n^2-n}{2}\right) + (c_7 + c_8 + c_9 + c_{10})\frac{n^2-n}{2} + (c_{11} + c_{12})n$$

$$= c_{13}n^2 + c_{14}n + c_{15}$$

$$= \Theta(n^2)$$

Date: No:

(02) In the above given algorithms, there is no difference in both worst case time complexities. Both have $\Theta(n^2)$ time complexity.

(03) Yes.

Instead of calculating every steps we can calculate using the loops

First we just consider the number of times each loop executes in the worst case. ~~loop is~~ The interactions are multiplied if the loop is within a loop ~~and~~ or added if the loop is outside a loop.

For an example

```
for i=1 to n {          ——— (1)
    for j=1 to n {      ——— (2)
    //- - - -
    For m=1 to n {      ——— (3)
    }
    }
    For k=1 to n/2 {    ——— (4)
    }
}
```

loops 1, 2 & 4 runs n times while loop 3 runs $n/2$ times

$T(n) = n \cdot n/2 + n \cdot n \cdot n = n^3 + n^2/2 = \Theta(n^3)$

(1) (4) (1) (2) (3)