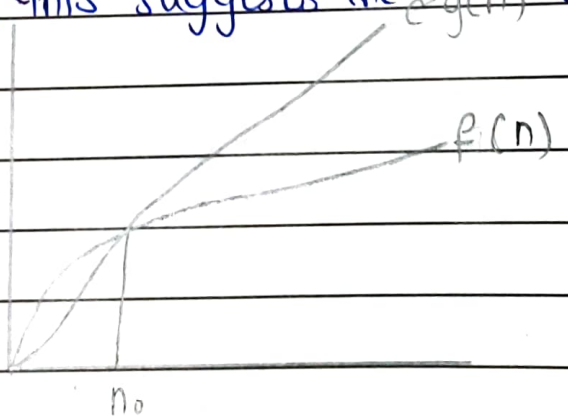
Question Nos.

Q.1] 1] Asymptotic notions are used to check the time etc complexity of algorithms and to find out which algorithm is the most time efficient when input size is large enough.

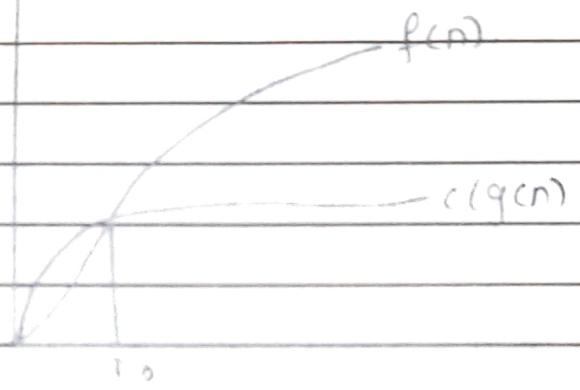
2] Big O → This is used to represent the worst case of the time complexity. This suggests the max time that an algorithm can take for completion



In the above graph $cg(n)$ and $f(n)$ are functions and we can say that $cg(n) = O(f(n))$ if and only if $cg(n) \geq f(n)$ and $n > n_0$.

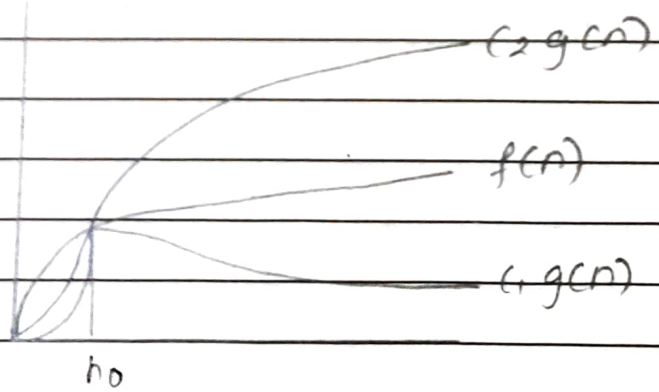
Question Nos. | Marks Awarded

2] Big. Ω → This is used to represent the best of time complexity of an algorithm. This suggests the minimum time algorithm can can ~~fo~~ take for its completion.



From the above graph we can say that ~~f(n) = Ω(c g(n))~~ ~~c g(n) = Ω(f(n))~~ if and only if $f(n) \geq c(g(n))$ and $n > n_0$.

3] Big θ → This is used to represent the average of time complexity of an algorithm. This suggests the average time an algorithm can take for its completion.



From above graph we can say that ~~c(g(n)) = f(n) = θ(c(g(n))~~ ~~c g(n) = θ(f(n))~~

$c(g(n)) = \theta(f(n))$ if and only if

Question Nos.		Marks Awarded

$c_1 g(n) <= f(n) <= c_2 g(n)$ and
$n > n_0$;

Question Nos. | Marks Awarded

Q.2] 1] Insertion at start

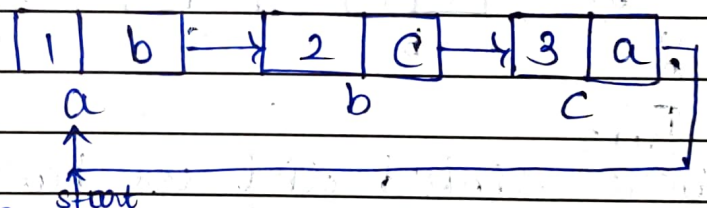
Algo insert_start (SPARP.)

Begin {

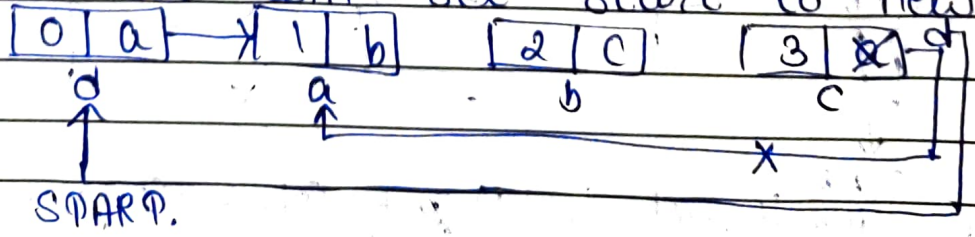
1. IF AVAIL = NULL
   PRINT ("No ~~node~~ available memory")
2. ELSE ~~AVA~~ NEWNODE = AVAIL and NEWNODE → DATA = VALUE
3. SET PTR = SPARP
4. WHILE (PTR → NEXT != ~~NULL~~ SPARP)
   SET PTR = PTR → NEXT
5. SET PTR → NEXT = NEWNODE
6. SET NEWNODE → NEXT = SPARP
7. SET SPARP = NEWNODE
8. RETURN SPARP

} END.

Example :



Suppose I have the above circular linked list and I want to insert [0] ... According to the algorithm pt 3, there will ptr variable that will traverse the list and as soon as it reaches the last node with next address as start, it will replace that with newnode address d and newnode next will have address of first node and we will set start to newnode :



Question Nos. | Marks Awarded

In this way we have our node inserted in start.

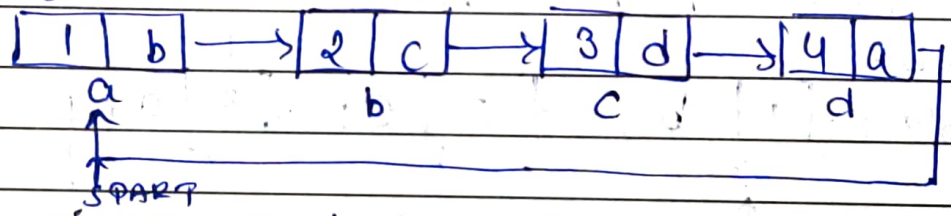
2] Algo Delete_last (START)

Begin {

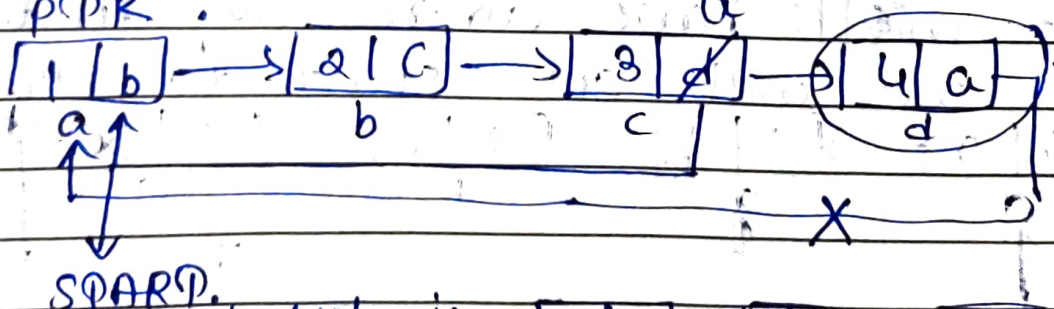
1. If ~~SET PTR~~ = START = NULL
   PRINT ("No node to delete")
2. ELSE SET PTR = START
3. WHILE (PTR → NEXT != START
   SET PREP = PTR
   SET PTR = PTR → NEXT
4. SET PREP → NEXT = START
5. FREE (PTR)
6. RETURN START

} END.

Eg:



Using above linked list we have to delete [4|a]. We set two pointer ptr and prep. PTR goes ahead and PREP follows by keeping behind PTR. We traverse using PTR and set PREP according. Once we reach [4|a] our PREP is on [3|d]. We change its address to start and free the space of PTR.



∴ New linked list is 

Question Nos | Marks Awarded

Q 3] INFIX → (A+B*(C-D)^E)/(F-G+H*I)

REVERSE → (I*H+G-F)/(E^(D-C)*B+A)

SYMBOL	STACK	EXPRESSION
(	(	-
I	(	I
*	(*	I
H	(*	IH
+	(+	IH*
G	(+	IH*G
-	(-	IH*G+
F	(-	IH*G+F
)	-	IH*G+F-
/	/	IH*G+F-
(	/(	IH*G+F-
E	/(;	IH*G+F-E
^	/(^	IH*G+F-E
(	/(^(	IH*G+F-E
D	/(^(	IH*G+F

Question Nos. | Marks Awarded

Q.3] INFIX → (A + B * (C - D)^ E) / (F - G + H * I)

SYMBOL	STACK	EXPRESSION
(	(	
A	(	A
+	(+	A
B	(+	AB
*	(+*	AB
(	(+*(	AB.
C	(+*(	ABC
-	(+*(-	ABC
D	(+*(-	ABCD
)	(+*	ABCD-
^	(+^	ABCD-*
E	(+^	ABCD-*E
)	-	ABCD-*E^+
/	/	ABCD-*E^+
(	/(	ABCD-*E^+
F	/(	ABCD-*E^+F
-	/(-	ABCD-*E^+F
G	/(-	ABCD-*E^+FG
+	/(+	ABCD-*E^+FG-
H	/(+	ABCD-*E^+FG-H
*	/(+*	ABCD-*E^+FG-H
I	/(+*	ABCD-*E^+FG-HI
)	/	ABCD-*E^+FG-HI*+
-	-	ABCD-*E^+FG-HI*+/

∴ Postfix expression is ABCD-*E^+FG-HI*+/