Question Nos.

Marks Awarded

Q3] given expression:

(A + B * (C - D) ^ E) / (F - G + H * I)

character	stack	Expression
(	(	
A	(	A
+	(+	A
B	(+	AB
*	(+ *	AB
(	(+ * (	AB
C	(+ * (	ABC
-	(+ * (-	ABC
D	(+ * (-	ABCD
)	(+ *	ABCD -
^	(+ * ^	ABCD -
E	(+ * ^	ABCD - E
)		ABCD - E ^ * +

Question Nos.			Marks Awarded
	/	/	ABCD-E ^ * +
	(	/(	ABCD-E ^ * +
	F	/(	ABCD-E ^ * + F
	-	/(-	ABCD-E ^ * + F
	G	/(-	ABCD-E ^ * + F G
	+	/(+	ABCD-E ^ * + F G -
	H	/(+	ABCD-E ^ * + F G - H
	*	/(+*	ABCD-E ^ * + F G - H
	I	/(+*	ABCD-E ^ * + F G - H I
	)	/	ABCD-E ^ * + F G - H I * +
			ABCD-E ^ * + F G - H I * + /

∴ The final postfix expression is :

ABCD-E ^ * + F G - H I * + /

Q2] assuming that a circular linked list has ~~also~~ already been created with a head pointer pointing to the first node.

1. Insertion at the start

Algo Insert_beg ()

```
IF(AVAIL == NULL)
   Write "SPACE UNAVAILABLE"
ELSE
   NEWNODE = AVAIL
NEWNODE → DATA = NEWDATA
NEWNODE → NEXT = HEAD;
```

Question Nos.

Marks Awarded

WHILE (ptr → NEXT != HEAD)
ptr = ptr → NEXT
ptr → next = NEWNODE
HEAD = NEWNODE
}

code in C:

```
void insert_beg (struct node *head)
{
   struct node *ptr = head;
   int val;
   struct node *newnode = NULL;
   newnode = (struct node *) malloc (sizeof (struct node));
   newnode → data = val;
   newnode → next = head;
   if (newnode == NULL)
   {
      printf ("Memory allocation failed ");
      return;
   }
   newnode → data = val;
   newnode → next = head;
   while (ptr → next != head)
   {
      ptr = ptr → next;
   }
   if (ptr = NULL)
   {
      printf ("Not a circular linked list);
```

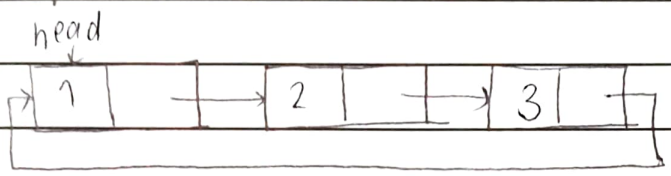
Question Nos.

Marks Awarde

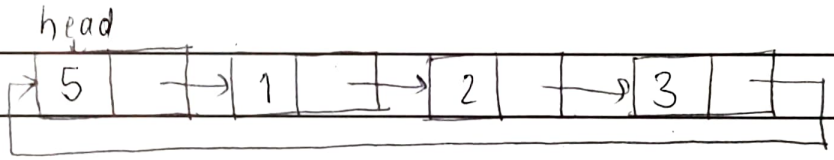
```
        }
        return;
    }
    ptr -> next = newnode;
    head = newnode;
}
```

~~2. Deletion of last node~~

Example



After insertion :



2. Deletion of last node

Algo ~~sat~~ DELETE_END()

```
PTR = HEAD
PREPTR = HEAD
WHILE(PTR -> NEXT != HEAD)
{
    PREPTR = PTR;
    PTR = PTR -> NEXT
}
IF PTR == NULL
    Write "Not a Circular Linked List"
    Go to ~~Step~~ END
```

Question Nos.

Marks Awarded

FREE PTR
PREPTR → NEXT = HEAD
END
}

code in C:

```
void delete_last(struct node *head)
{
    struct node *preptr = head;
    struct node *ptr = head;

    while (ptr -> next != head)
    {
        preptr = ptr;
        ptr = ptr -> next;
    }
    if (ptr == NULL)
    {
        printf("Not a circular linked list");
        return;
    }
    free(ptr);
    preptr -> next = head;
}
```

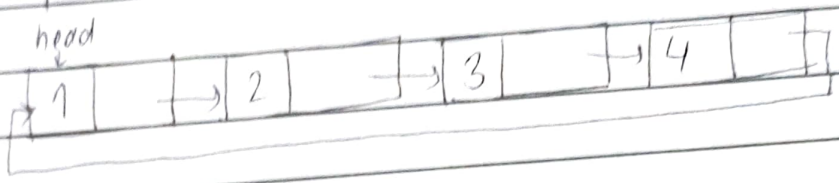
Exam

NEXT PAGE →

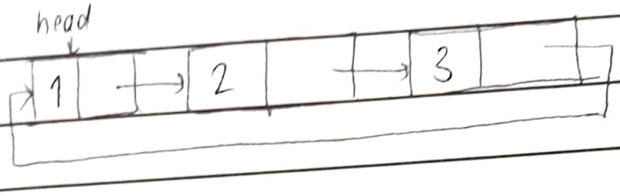
Question Nos.

Marks Awarded

Example



After deletion:



For insertion at the start, a new node was created using malloc. Newnode points to head and the last node points to the newnode. After this, we shift head to the new node. This ensures that the new node becomes the first node of the circular linked list.

For deletion of the last node, we make 2 pointers. ptr points to the last node while preptr points to the node before ptr. the last node. We use free() to delete the last node and the new last node (preptr) now points to head.

Q1] asymptotic notations are used to represent time complexity of algorithms. They are as follows:

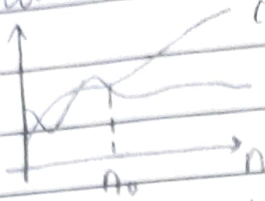
(i) Big-Oh (O) - Upper bound

This represents the worst-case time complexity. Thus, this is the highest amount of time required

Question Nos.

Marks Awarded

by the algorithm in order to execute.

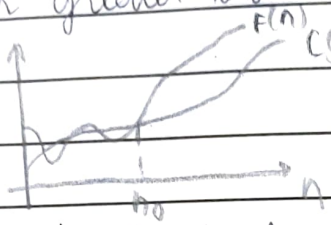


$n_0 < n$ $f(n) \leq cg(n)$ $c > 0$

Therefore, this is the worst-case scenario

(ii) Big-Omega (Ω) - Lower bound

This shows the best case time-complexity. This shows the lower bound. The time taken will be equal to or greater than this.

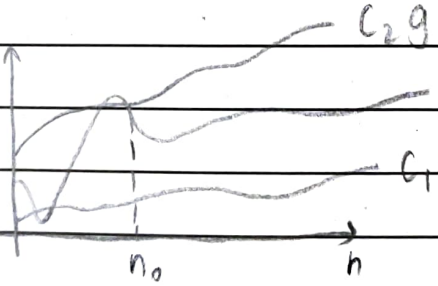


$n_0 < n$ $f(n) \geq cg(n)$ $c > 0$

Therefore, this is the best-case scenario.

(iii) Big-Theta (Θ) - Upper and lower bound.

This shows the tight bound time complexity, that is the average case. It mentions an upper and a lower bound.



$n_0 < n$, $c > 0$

$c_1g(n) \leq f(n) \leq c_2g(n)$

Therefore, this gives the average time complexity. The time complexity of the function $f(n)$ will always be between $c_1g(n)$ and $c_2g(n)$. Therefore, it gives a range for the time complexity.

These notations help us analyse the efficiency of algorithms