DSA FINAL 2018

Date

ANSWER: 01

=> ANALYSIS OF SELECTION SORT:-

Pseudo code:

```
1. For  i=1 to n:
2.      min_index=i
3.      For J= i+1 to n:
4.          If  A[min_index] > a[J]:
5.                  min_index=J
6.      swap (A[i], A[min_index])
```

• DRY RUNNING:-

=> BEST CASE:-

A = [1, 2, 3, 4]

=> 1ST ITERATION OF FOR-LOOP

1. i=1
2. min_index = 1

1st iteration of for-loop

3. J = 2
4. If 1 > 2 : → False
5.

Date

=> 2nd iteration of for-loop:-

3. J=3
4. If 1>3: →False
5. _____

=> 3rd iteration of for-loop:

3. J=4
4. If 1>4: →False
5. _____

=> 4th iteration of for-loop

3. J=5

— x — x loop terminates — x — x —

6. swap (A[1], A[1]) → {A[1,2,3,4]

— x — 1st iteration of loop terminates — x — x —

=> 2ND ITERATION OF FOR-LOOP:-

1. i=2
2. min_index = 2

=> 1st iteration of for-loop:-

3. J=3
4. If 2>3:- →False
5. _____

=> 2nd iteration of for-loop:-

3. J=4
4. If 2>4:- →False
5. _____

3rd iteration of for-loop:-

J=5

—x —x — loop terminates — x — x —

6. swap(A[2], A[2]) → {A:[1,2,3,4]}

• 3RD ITERATION OF FOR-LOOP:-

1. i=3
2. min-index=3
3. 1st iteration of for-loop:-

   J=4
4. If 3>4:- → False
5. ———

2nd iteration of for-loop:-

3. J=5

— x — x — x loop terminates — x — x —

6. swap(A[3], A[3]) → {A:[1,2,3,4]}

• 4TH ITERATION OF LOOP:-

1. i=4
2. min-index=4

1st iteration of for-loop:-

3. J=5

— x — x — loop terminates — x — x —

6. swap(A[4], A[4]) → {A:[1,2,3,4]}

Date _____

5TH ITERATION OF FOR-LOOP:-

1. i=5

—— x —— x loop terminates —— x —— x ——

Date

• Dry Running Of Worst

CASE:-

A = [4, 3, 2, 1]

• 1st Iteration Of For-Loop:-

1. i = 1
2. min_index = 1

⇒ 1st iteration of for-loop

3. J = 2
4. If 4 > 3 : ⟶ True
5. min_index = 2

⇒ 2nd iteration of for-loop

3. J = 3
4. If 3 > 2 : ⟶ True
5. min_index = 3

⇒ 3rd iteration of for-loop :-

3. J = 4
4. If 2 > 1 : ⟶ True
5. min_index = 4

⇒ 4th iteration of for-loop :-

3. J = 5

— x — x — loop terminales — x — x —

6. swap(A[1], A[4]) ⟶ A = [1, 3, 2, 4]

2ND ITERATION OF FOR-LOOP:-

1. i=2
2. min-index=2

1st iteration of for-loop:

3. J=3
4. If A[2] > A[3] : → TRUE
5. min-index=3

=> 2nd iteration of for-loop:-

3. J=4
4. If 2>4 : -False
5. ———

=> 3rd iteration of for-loop:-

3. J=5
—x —x— loop terminates —x —x —
6. swap (A[2], A[3]) → A=[1,2,3,4]

3RD ITERATION OF FOR-LOOP:-

1. i=3

2. min-index=3

1st iteration of for-loop:-

3. J=4
4. If 3>4 :
5. ———

=> 2nd iteration of for-loop:-

3. J=5

Teacher's Signature

Date

—x——x— loop terminates —x —x ——

6. swap(A[3], A[3]) => A=[1,2,3,4]

•4TH ITERATION OF FOR-LOOP:-

1. i=4

2. min_index=4

=> 1st iteration of for-loop:-

3. J=5

—x ——x —loop terminates —x — x —x—

6. swap (A[4], A[4]) => A=[1,2,3,4]

•5TH ITERATION OF FOR-LOOP:-

1. i=5

——x ——— loop terminates — x —— x ——

ANALYSIS TABLE:-

Date

Line no	Time/ instruction	frequency	
		BEST CASE	WORST CASE
1	C	$n+1$	$n+1$
2	C	n	n
3	C	$\sum_{J=1}^{n} J \cdot \frac{n(n+1)}{2}$	$\sum_{J=1}^{n} J \cdot \frac{n(n+1)}{2}$
4	C	$\sum_{J=1}^{n} J-1 \cdot \frac{n(n-1)}{2}$	$\sum_{J=1}^{n} J-1 \cdot \frac{n(n-1)}{2}$
5	C	0	$\sum_{J=1}^{n} J-1 \cdot \frac{n(n-1)}{2}$
6	C	n	n

=> TIME COMPLEXITY OF

BEST CASE:-

$$T(n) = \left[n+1 + n + \frac{n(n+1)}{2} + \frac{n(n-1)}{2} + n \right] c$$

$$T(n) = \left[3n+1 + \frac{n(n+1)}{2} + \frac{n(n-1)}{2} \right] c$$

$$T(n) = \left[\frac{6n+2+n^2+\cancel{n}+n^2-\cancel{n}}{2} \right] c$$

$$T(n) = \left[\frac{2n^2+6n+2}{2} \right] c \Rightarrow T(n) = [n^2+3n+1]\, c$$

Teacher's Signature

Date

$T(n) = K_1n^2 + K_2n + K_3$

=> Where, K_1, K_2, K_3 are constants.

• EXPRESSING TIME COMPLEXITY IN BIG Ω NOTATION:-

$0 \leq c\,g(n) \leq f(n) \; ; \; \forall n \geq n_0$

$0 \leq c(n^2) \leq c_1n^2 + c_2n + c_3 \; ; \; \forall n \geq n_0$

↳ To give function lower bound.

$T(n) = \Omega(n^2)$

=> TIME COMPLEXITY OF WORST CASE:-

$$T(n) = \left[n+1+n+\frac{n(n+1)}{2}+\frac{n(n-1)}{2}+\frac{n(n-1)}{2}+n\right]c$$

$$T(n) = \left[3n+1+\frac{n^2+n+n^2-n+n^2-n}{2}\right]c$$

$$T(n) = \left[\frac{6n+2+2n^2+n^2-n}{2}\right]c$$

$$T(n) = \left[\frac{3n^2+5n+2}{2}\right]c \Rightarrow \left[\frac{3}{2}n^2+\frac{5}{2}n+1\right]c$$

$T(n) = K_1n^2 + K_2n + K_3$

Where K_1, K_2, K_3 are constants.

Date

Expressing Time Complexity

In Big-Oh Notation:-

$0 \leq f(n) \leq cg(n) \; ; \; \forall n \geq n_0$

$0 \leq c_1n^2 + c_2n + c_3 \leq c(n^2)$

↓

To give function upper bound

$T(n) = O(n^2)$

Discussion :-

(Best Case):-

=> Best case time complexity of selection sort is $\Omega(n^2)$.

=> $\Omega(n^2)$ means that function is covered by a lower bound. means that minimum time which a code can take.

=> selection sort in best case grows quadratically.

Date

(WORST CASE):-

=> worst case time complexity of selection sort is $O(n^2)$.

=> $O(n^2)$ means that function is covered by a upper bound means that maximum time which a code can take.

=> selection sort in best case and worst case grows quadratically.

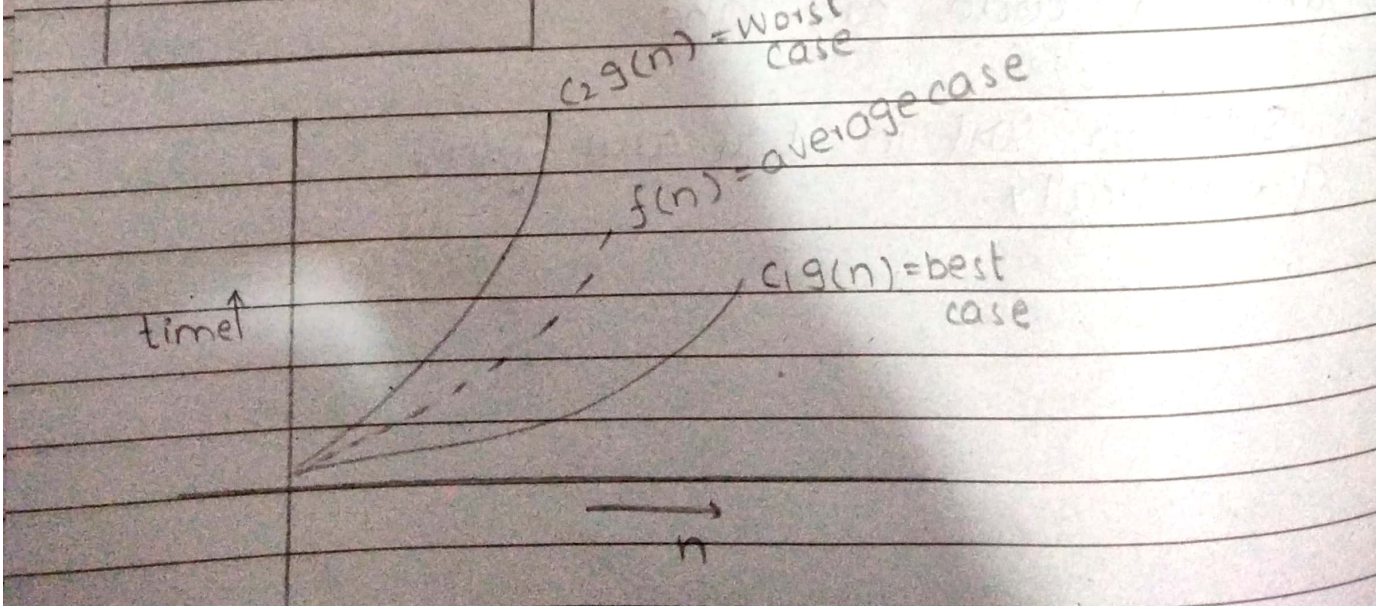
• EXPRESSING RUNNING TIME IN θ-NOTATION:-

such that,

$0 \leq C_2 g(n) \leq f(n) \leq C_1 g(n) \; ; \; \forall n \geq n_0$

$0 \leq C_2(n^2) \leq C_1 n^2 + C_2 n + C_3 \leq C_1(n^2)$

$T(n) = \theta(n^2)$	= Average bound.



Ans: 02

Date

• Merge Analysis:-

```
MERGE (L,R,A):
1. nL = length (L)
2. nR = length (R)
3. i=J=K=0

4. While  i < nL  and  J < nR :
5.          if L[i] < R[J]:
6.                A[K] = L[i]
7.                i = i+1
8.          else:
9.                A[K] = R[J]
10.               J = J+1
11.         K = K+1
12.   While  i < nL:
13.           A[K] = L[i]
14.           i = i+1
15.           K = K+1
16.
17.   While  J < nR:
18.           A[K] = R[]
19.           J = J+1
                K = K+1
```

• Dry Running:-

L = [1,2]
R. [3,4]
A. [ ]

1. nL = 2
2. nR = 2
3. i=0, J=0, K=0

1ST ITERATION OF WHILE

4. While 0<2 and 0<2 : → True
5. If 1<3 : → True
6. A[0] = 1 → A = [1]
7. i = 1
8. _____
9. _____
10. _____
11. K = 1

2ND ITERATION OF WHILE :-

4. While 1<nL and 0<2 : → True
   (nL = 2)
5. If 2<3 : → True
6. A[1] = 2 → A = [1, 2]
7. i = 2
8. _____
9. _____
10. _____
11. K = 2

3RD ITERATION OF WHILE :-

4. While 2<2 and 0<2 : False

— x — x — loop terminates — x — x —

Date

1ST ITERATION OF WHILE :-

12. 2 < 2 :- False

— x —— x — loop terminates —— x —— x ——

1ST ITERATION OF WHILE :-

17. While 0 < 2 : → True

18 A[2] = 3 → A = [1, 2, 3]

18 J = 1

19 K = 3

2ND ITERATION OF WHILE :-

16. While 1 < 2 : → True

17. A[3] = 4 → A = [1, 2, 3, 4]

18. J = 2

19 K = 4

3RD ITERATION OF WHILE :-

16. While 2 < 2 : False

—— x —— x —— x loop terminates —— x —— x

Analysis Table

Date

Line no	Time/instruction	frequency
1	c	1
2	c	1
3	c	1
4	c	nL + 1
5	c	nL
6	c	nL
7	c	nL
8	c	0
9	c	0
10	c	0
11	c	nL = 2
12	c	1
13	c	0
14	c	0
15	c	0
16	c	nR + 1
17	c	nR
18	c	nR
19	c	nR

Time Complexity :-

$T(n) = 6 + 2 + 4nL + 4nR$

$T(n) = 6 + 2 + 8\ \frac{4(nL + nR)}{n}$

$T(n) = 6 + 2 + 4n$

$T(n) = 8 + 4n$ $\Rightarrow$ linear growth $\Rightarrow$ $T(n) = K_1 n + K_2$

→ K_1 and K_2 are constant

Teacher's Signature

Date

converting time complexity in O notation

$0 \leq f(n) \leq c(g(n)) \; ; \; \forall \; n \geq n_0$

$0 \leq 8+4n \leq 12n \quad ; \forall n \geq 1$

$T(n) = O(n)$

• DISCUSSION:-

⇒ Time complexity of merge procedure is $O(n)$

⇒ merge procedure in best and worst case grows linearly.

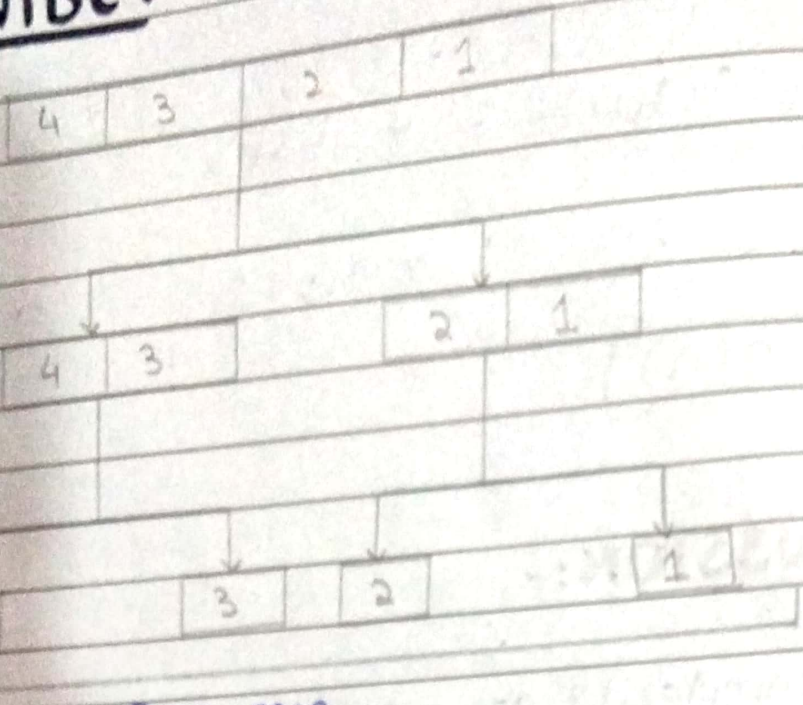
• MERGE SORT ANALYSIS

(DIVIDE AND CONQUER APPROACH)

⇒ DRY RUNNING:-

⇒ Worst case = A = [4,3,2,1]

DIVIDE :-



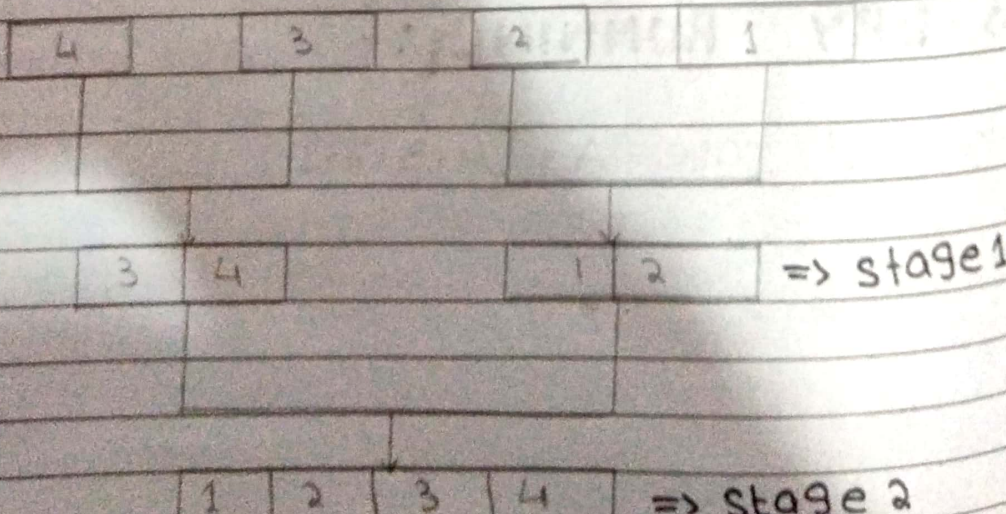
Base case

=> For a base case we have "c" seconds.

=> W for 'n' number of elements we will call base for n times

Time for all base cases = nxc

MERGE :-



Date

Count Stages Of Mergte:-

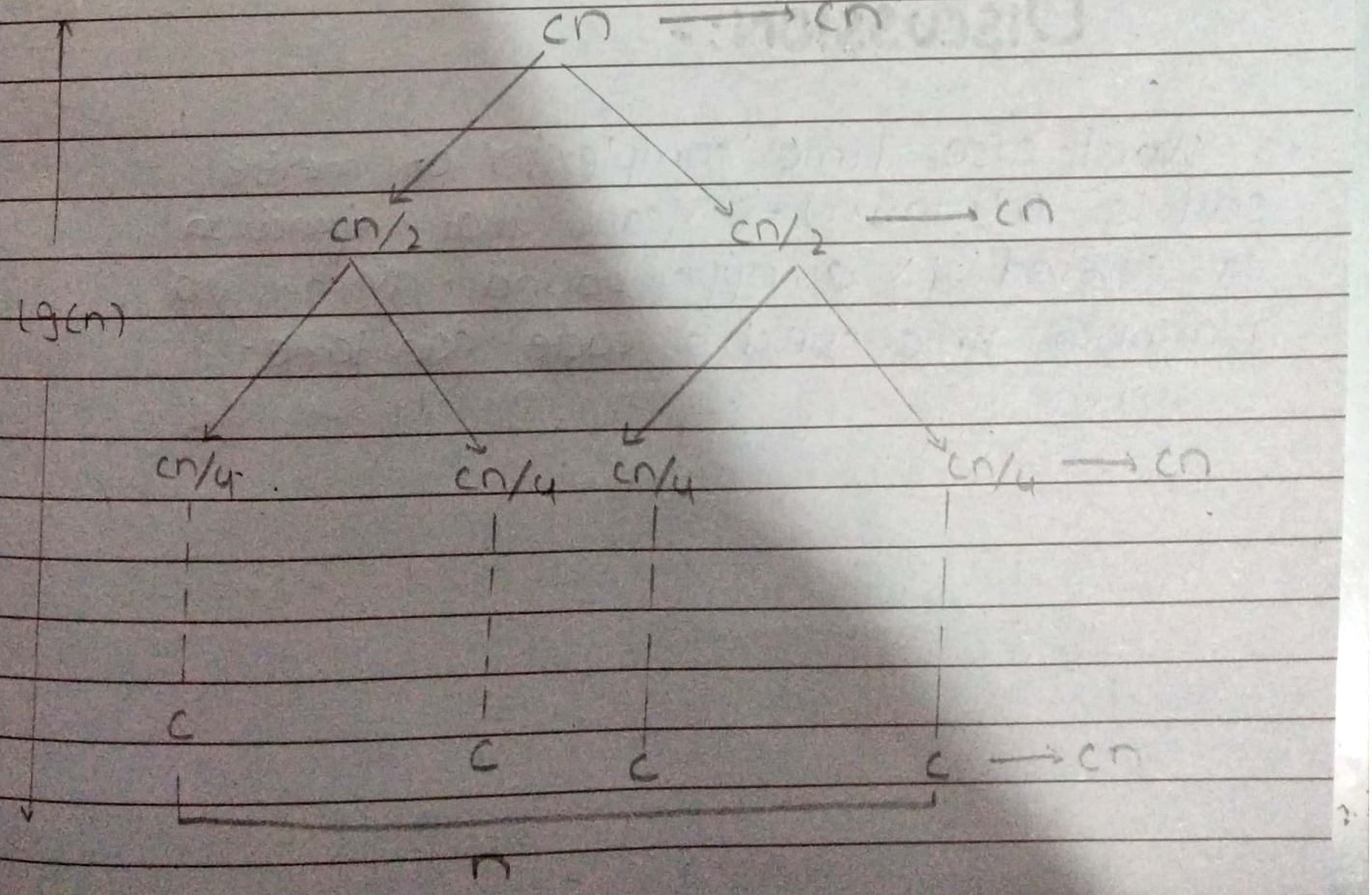
=> When we have 4 entries, we got 2 stages of merge

=> If we have 8 entries, we will got 3 stages of merge

=> If we have 16 entries, we will got 4 stages of merge

=> If we have n entries, we will got $\log_2(n)$ stages of merge.

Total Time:-



Date

Total time, $cn\lg n + cn$

$$T(n) = cn(1+\log n)$$

EXPRESSING TIME COMPLEXITY IN BIG-OH NOTATION:-

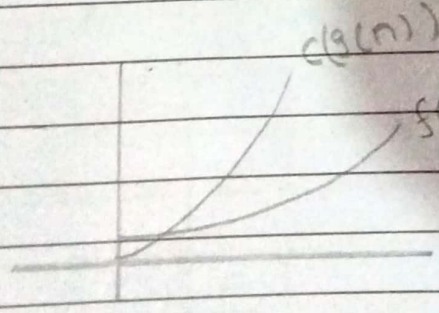
$0 \leq f(n) \leq c(g(n)) \; ; \forall \; n \geq n_0$

$0 \leq cn(1+\log(n)) \leq cn(\log(n)) \; ; \forall \; n \geq n_0$

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

• DISCUSSION:-

=> Worst case time complexity of merge sort is $O(n\log n)$ => Means that function is covered by a upper bound. Means that maximum time which a code can take.



$1 < \log n < n < n\log n < n^2 < n^3 < 2^n < n^n$

Date

Ans: 03

• Asymptotic Notation :-

⇒ Asymptotic notation give us an idea about how good a given algorithm is as compared to some other algorithm.

⇒ primarily there are three types of widely used asymptotic notation.

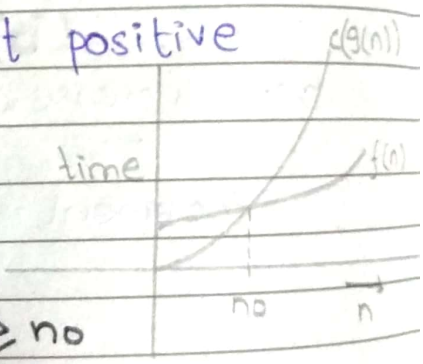
- BigOh notation 'O'
- Big omega notation 'Ω'
- Big theta notation 'θ'

• Big-oh Notation :-

Big oh notation is used to describe the asymptotic upper bound.

$f(n)$ is $O(g(n))$ iff there exist positive constant C and no

such that,



$0 \leq f(n) \leq C(g(n)) \; ; \; \forall \; n \geq no$

⇒ lets consider an example

Represent $f(n) = 2n+3$ in big Oh notation

$0 \leq 2n+3 \leq 5(n) \; ; \; \forall \; n \geq 1$

$f(n) = 2n+3 = O(n)$

Teacher's Signature

Date

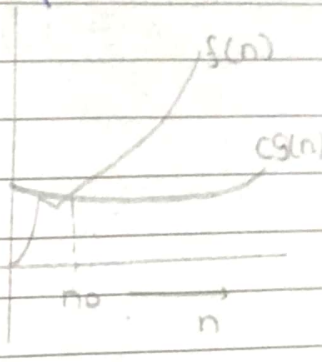
· Big - omega Notation:-

⇒ Big omega notation is used to describe the asymptotic lower bound.

⇒ $f(n)$ is $\Omega(g(n))$ iff there exist positive constant c and n_0

such that:

$$0 \leq c(g(n)) \leq f(n) \; ; \; \forall \; n \geq n_0$$



· Now represent $f(n) = 2n+3$ in big omega notation

$$0 \leq 1n \leq 2n+3 \; ; \; \forall \; n \geq 1$$

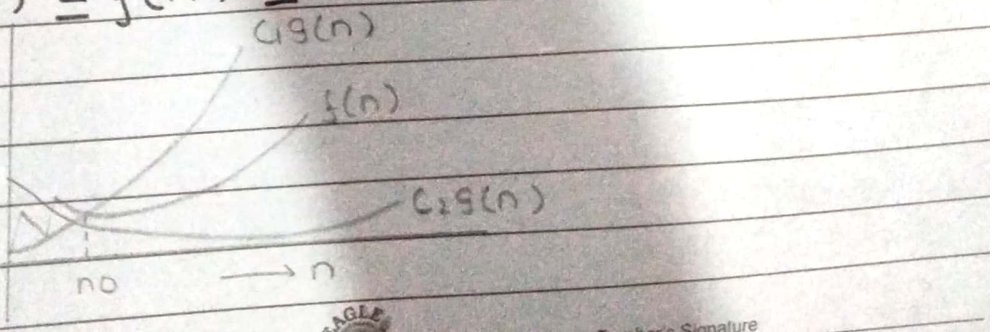
$$f(n) = 2n+3 = \Omega(n)$$

· Big Theta Notation:-

⇒ $f(n)$ is $\theta(g(n))$ iff there exists positive constants c_1, c_2 and n_0

such that:

$$0 \leq c_2 g(n) \leq f(n) \leq c_1 g(n) \; ; \; \forall n \geq n_0$$

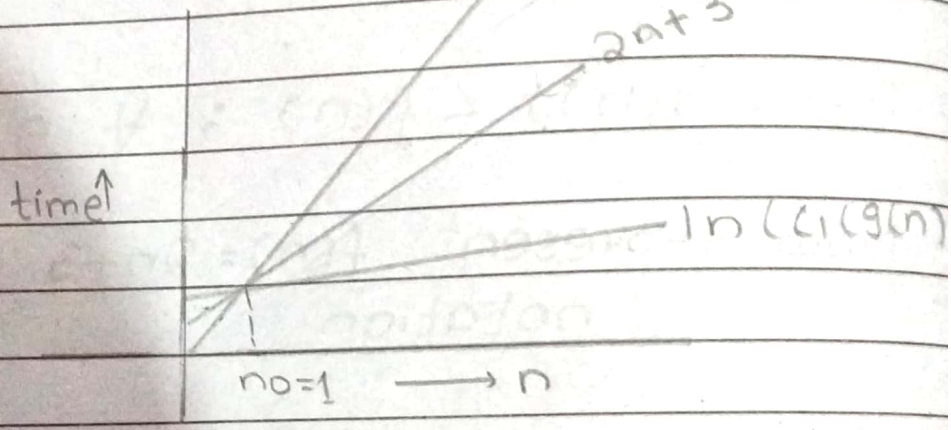


Teacher's Signature

=> Now considering our previous example

$0 \leq 1n \leq 2n+3 \leq 5n \quad \forall n \geq 1$

$\theta(n) = 2n+3 = f(n)$

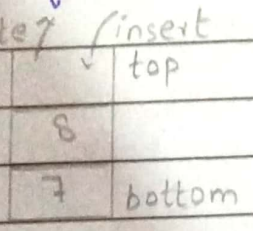


Ans: 4

Date

• Stack :-

Stack is a linear data structure which follows a particular order in which the operations are performed. The order may be LIFO (Last In First Out). Stack is list or collection with restriction that insertion and deletion both can be performed at one end called the top of the stack.



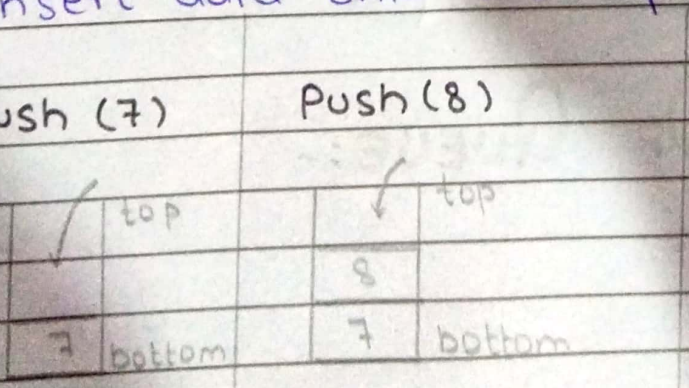
- A pile of books, a stack of dinner plates, can of tennis ball all are examples of stacks.

• Operations Of Stacks :-

- Push operation
- Pop operation

• Push (DATA) :-

Insert data onto the top of stack.



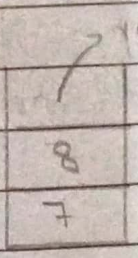
Page No.

Teacher's Signature

=> The last inserted element is 9.

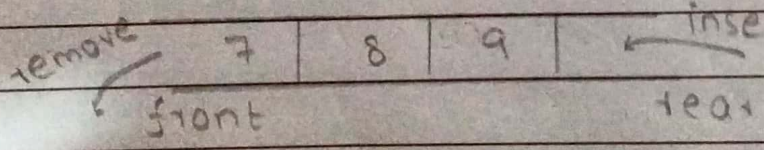
• Pop ( ):-

Removes the last inserted element from the stack.



• Queue:-

Queue is a linear data structure which follows a particular order in which the operations are performed. The order may be FIFO (First IN First Out). Queue is a list or collection with restriction that insertion can be performed at one end called rear and deletion can be performed at other end called front.



A good example of queue is any queue of consumers for goods where the person that came first is served first.

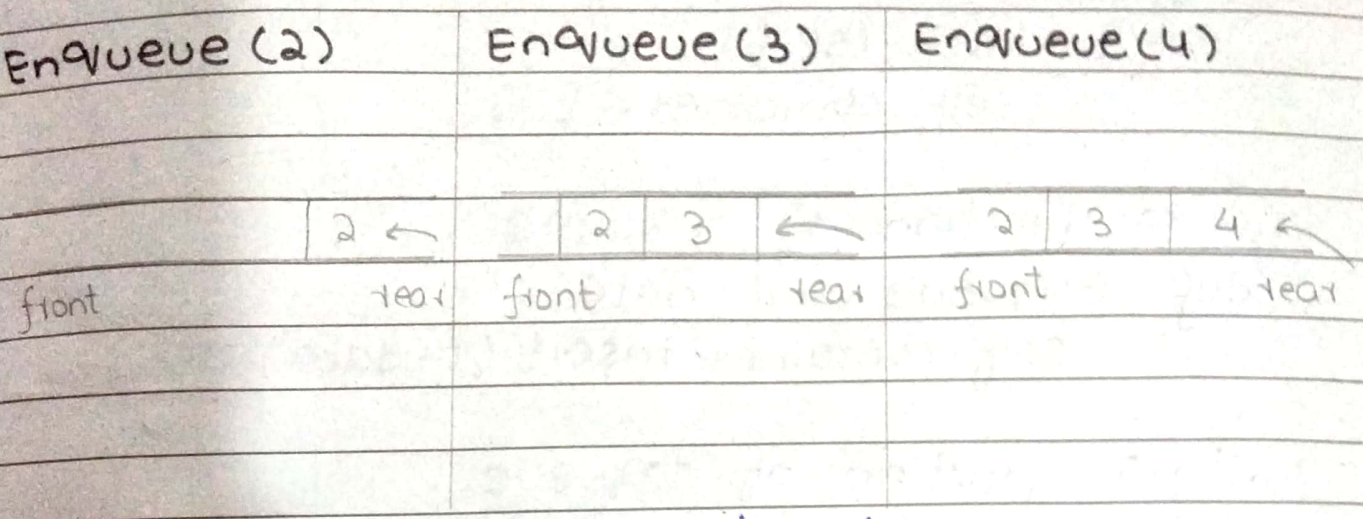
• Operations Of Queue:-

- EnQueue
- DeQueue

Date

ENQUEUE (DATA):-

Insert data in queue.

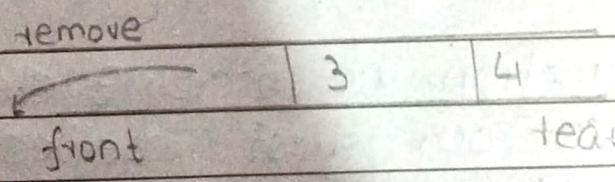


⇒ The last inserted element is 4
⇒ The first inserted element is 2.

• DEQUEUE ( ):-

Removes the first inserted element from the Queue.

⇒ Dequeue ( )



• CONCLUSION:-

The difference between stacks and queues is in removing. In stacks we remove the element the most recently added. In queues we remove the element the least recently added.

PYTHON CLASS OF

Date

QUEUE :-

```
Class Queue ( ):
    def --init-- (self):
        self. container = [ ]

# Writing function of enqueue
    def enqueue (self, data):
        self. container. insert (0, data)

# Writing function of dequeue
    def dequeue (self):
        se = len (self. container)

        self. container. remove (self. container [se-1])

# Writing function to print Queue
    def show-all (self):
        return self. container.
```

insert

[5, 6, 7]

0

[5, 6, 7]

removed

=> In enqueue function, we inserting element at 0th index so our first inserted element was at last index. that's why in dequeue function we remove last element of list. to fulfill the principle of (FIFO).

Ans: 5

Date

My preference for developing a system which mantain data of employees in an organization would be OOP. (Object Oriented Programming).

(a)

```
Class Employee ( ):
    def --init-- (self, name, pay, job):
        self.name = name
        self.pay = pay
        self.job = job
```

=> constructor is used to store data in object variables / instance variables. Here, through constructor we can store the data of employees. The attributes of class is name, pay and job.

(b)

```
def increase_salary (self, percentage):

    increment = self.pay / 100 x Percentage

    self.pay . self.pay + increment
```

=> The above method will increase the salary of a person with desired percentage.

ANS:06

Date

DYNAMIC SET:-

A set that supports insertion and deletion of elements.

OR

A data structure frequently used in relational database access.

• OPERATIONS:-

- SEARCH (S, K)
- INSERT (S, x)
- DELETE (S, x)
- MAXIMUM (S)
- PREDECESSOR (S, x)

* Suppose S is a set of data, x is a data object and K is a key.

• SEARCH (S, K):

```
def Search(S,K):
    n=len(S)
    for i in range(n):
        If S[i]==K:
            Print("List is Present")
            return
    Print("Key is absent")
```

WORKING:- The function search for a key (K) in set (S).

• INSERT (S, x):-

```
def INSERT (S, x):
    S.insert (0, x)
```

WORKING:- The function insert data object (x) in set (S).

• DELETE (S, x):-

```
def DELETE (S, x):
    S.remove (x)
```

WORKING:- This function remove data object (x) from set (S).

• MAXIMUM (S):-

```
def Maximum (S):
    n = len(S)
    for i in range (n):
        for j in range (n-i-1):
            If S[j] > S[j+1]:
                S[j], S[j+1] = S[j+1], S[j]
return S[n-1]
```

swap

→ Return last index of set

WORKING:- This function returns the maximum value of set (S).

Date

• PREDECESSOR (S, x):
Return the element of set (S) whose key is the next smallest than that of x.

—— x —— PAPER —— x —— x ——

END

=>hope i will get full marks