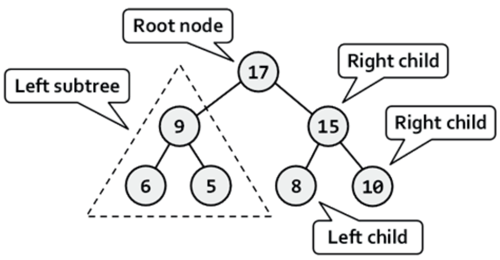
GATE | PSUs

COMPUTER SCIENCE & INFORMATION TECHNOLOGY

Data Structures

Text Book : Theory with worked out Examples and Practice Questions



HYDERABAD | AHMEDABAD | DELHI | BHOPAL | PUNE | BHUBANESWAR | BANGALORE | LUCKNOW
PATNA | CHENNAI | VISAKHAPATNAM | VIJAYAWADA | TIRUPATHI | KOLKATA

Data Structures

(Solutions for Text Book Practice Questions)

1. Arrays

01. Ans: 1010

Sol: Loc. of A (i) = L_0 + (i-lb) * C

Loc of A [0] = 1000+(0+5) × 2 = 1010

02. Ans: 1024 and 1024

Sol: (i) By RMO, the loc. of

$A[i, j] = L_0 + [(i-b_1)(u_2-b_2+1) + (j-b_2)]*C$

A [0, 5] = 1000 + [(0+2) × 5+ (5-3)] × 2

= 1000+24 = 1024

(ii) By CMO, the loc of

$A[i, j] = L_0 + [(j-b_2)(u_1-b_1+1) + (i-b_1)]*C$

A [0, 5] = 1000 +[(5–3) × 5 + (0+2)] × 2

= 1024

03. Ans: (a)

Sol: In general

RMO = L_0 + (i – 1) r_2 + (j – 1)

= 100 + (i – 1) 15 + (j –1)

= 100 + 15 i – 15 + j – 1

= 15 i + j + 84

04. Ans: (c)

Sol: Lower triangular matrix

$$\begin{pmatrix} a & 0 & 0 \text{----} 0 \\ b & c & 0 \text{----} 0 \\ d & e & f \;\; 0-0 \\ g & h & i \;\; j-0 \\ \text{-------------------} \\ \text{-------------------} \end{pmatrix}$$

RMO = L_0 + the number of elements in (i – 1) rows + one dimensional elements

$= L_0 + (1 + 2 + \ldots\ldots + i - 1) + (j - 1)$

$= L_0 + i\,\dfrac{(i-1)}{2} + (j - 1)$

05. Ans: (c)

Sol: CMO:

Storage:

$a_{11}\ a_{21}\ a_{31}\ a_{41} \mid a_{22}\ a_{32}\ a_{42} \mid a_{33}\ a_{43} \mid a_{44}$

0 1 2 3 4 5 6 7 8 9

Retrieval:

loc of $A[i, j] = L_0 + 2D + 1D$

$= L_0 + [(j-1)\ \text{cols} + (i - i\ell b)]$

In each col., $i\ell b = j$.

Loc. of $A[i, j] = L_0 + [(j-1)\ \text{cols} + (i - j)]$

In (j–1) cols

The no. of elements is

$n + (n-1) + \ldots + (n - (j-1-1))$

$= (j-1)n - [1 + 2 + \ldots + j - 2]$

$= n(j-1) - \dfrac{(j-1)(j-2)}{2}$

loc. of A[i, j]

$= L_0 + \left[n(j-1) - \dfrac{(j-1)(j-2)}{2} + (i-j)\right]$

06. Ans: (d)

Sol: RMO:

Storage:

$a_{11}\ a_{12} \mid a_{21}\ a_{22}\ a_{23} \mid a_{32}\ a_{33}\ a_{34} \mid a_{43}\ a_{44}$

0 1 2 3 4 5 6 7 8 9

Retrieval:

loc of $A[i, j] = L_0 + 2D + 1D$

$= L_0 + \text{number of elements in } (i-1) \text{ rows} + (j - j\ell b)$

Row	**j/b**
4	3
3	2
2	1
except 1^{st} row	
i^{th}	(i–1)

loc. of $A[i, j] = L_0 + [(3i-4) + j - (i-1)]$

$= L_0 + (2i + j - 3)$

07. Ans: (a)

Sol: CMO:

Storage:

$a_{11}\ a_{21}\ |\ a_{12}\ a_{22}\ a_{32}\ |\ a_{23}\ a_{33}\ a_{43}\ |\ a_{34}\ a_{44}$

0 1 2 3 4 5 6 7 8 9

Retrieval:

loc. of $A[i, j] = L_0 + 2D + 1D$

$= L_0 + (j-1)\text{cols} + (i - i\ell b)$

Since i is Varying

Col	**i/b**
4	3
3	2
2	1
except	1^{st} column
j^{th}	j-1

$\therefore \text{loc of } A[i, j] = L_0 + [3(j-1) - 1 + i - (j-1)]$

$= L_0 + [2j + i - 3]$

08. Ans: (b)

Sol: Storage & Retrieval:

$a_{21}\ a_{32}\ a_{43}\ |\ a_{11}\ a_{22}\ a_{33}\ a_{44}\ |\ a_{12}\ a_{23}\ a_{34}$

0 1 2 3 4 5 6 7 8 9

If $i - j = 1$

loc of $A[i, j] = L_0 + 0 + (i - i\ell b)$

or

$(j - j/b)$

i.e.,

loc. of $A[i, j] = L_0 + 0 + (i-2)$

or

$(j-1)$

If $i - j = 0$

loc. of $A[i, j] = L_0 + (n-1) + (i-1)$

or

$(j-1)$

If $i - j = -1$ // upper diagonal

loc. of $A[i, j] = L_0 + 2n - 1 + (i-1)$

or

$(j-2)$

09. Ans: (a)

Sol: A sample 5 × 5 S-matrix is given below.

1	8	3	2	1
3	0	0	0	0
6	1	7	4	3
0	0	0	0	1
9	6	5	4	1

The compact representation is

[1,8,3,2,1, 6,1,7,4,3, 9,6,5,4,1, 3, 1].

10. Ans: 9

Sol: $2n - 1 = 10 - 1 = 9$

1	2	3	4
5	1	2	3
6	5	1	2
7	6	5	1

11. Ans: 190900

Sol: $n + (a - 1)(2n - a)$

$1000 + (101 - 1)(2.1000 - 101)$

$1000 + 100 \times (2000 - 101)$

$1000 + 100 \times 1899$

$1000 + 189900 = 190900$

12. Ans: (a)

Sol: Square Band Matrix:

It is denoted by $D_{n,a}$ where n is size of matrix, a is (a-1)'s diagonals exists above and below the matrix diagonal.

$$D_{5,4} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} \\ a_{51} & a_{52} & a_{53} & a_{54} & a_{55} \end{bmatrix}_{5\times5}$$

Size:- N (diagonal elements)

$+ 2[N-1+N-2+N-3+\ldots.+N-(a-1)]$

$= N+2[(a-1)N-(1+2+3+\ldots\ldots.+a-1)]$

Total no. of elements

$= n+ 2 [n-1 + n-2 + \ldots\ldots. + n- (a-1)]$

$= n + 2 [(a-1) n - [1+2+ \ldots\ldots. + (a-1)]]$

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

$= n+2n(a-1) - a(a-1) = n + (a-1)[2n-a]$

(b) $D_{6,4}$

In $D_{6,4} \rightarrow A(5,4)$ in 3rd diagonal

In $D_{6,5} \rightarrow A(5,4)$ in 4th diagonal

(i) i – j value is remained constant through out the diagonal

(ii) As 'a' value changes, accordingly K value also changes.

$\therefore$ K value = a – (i –j)

$\therefore$ loc. of A[i,j] = L_0 + no. of elements in(k–1) dig + 1D cross

loc. of A[i, j] =L_0 +no. of elements in (k–1) dig + (i – i*l*b) or (j– j*l*b)

but here i*l*b is varying diagonal by diagonal so prefer j*l*b which is always 1 in each diagonal

= L_0 + no. of elements in (k–1) dig +(j–1)

No. of elements in (k–1) diagonals is

$[n-(a-1)]+[n-(a-1)+1] + [n - (a - 1) + 2] + \ldots. [n-a+(k-1)]$

$[n-(a-1)] \rightarrow 1^{st}$ diagonal because it is$(a-1)^{th}$ diagonal

$(n-(a-1)+1) \rightarrow 2^{nd}$ diagonal

$[n-(a-1)+2] \rightarrow 3^{rd}$ diagonal

$n-a+(k-1) \rightarrow (k-1)^{th}$ term i.e. $(k-1)^{th}$ diagonal

$= (k - 1)(n-a) + 1 + 2 +\ldots.. +k - 1$

$= (k - 1)(n - a) + \frac{k(k-1)}{2}$

$\therefore$ loc. of A[i, j]

$= L_0 + \left[(k-1)(n-a) + \frac{k(k-1)}{2} + (j-1)\right]$

2. Stacks & Queues

01. (i). **Ans: (a)** (ii). **Ans: (c)**

Sol: Given array size m, say 9

Number of stacks n, say 3

$$0 \le i < n \quad T[i] = B[i] = i.\left\lfloor \frac{m}{n} \right\rfloor - 1$$

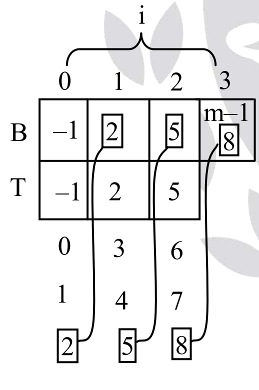
$$i = 0 \Rightarrow T[0] = B[0] = 0\left\lfloor \frac{9}{3} \right\rfloor - 1 = 0 - 1 = -1$$

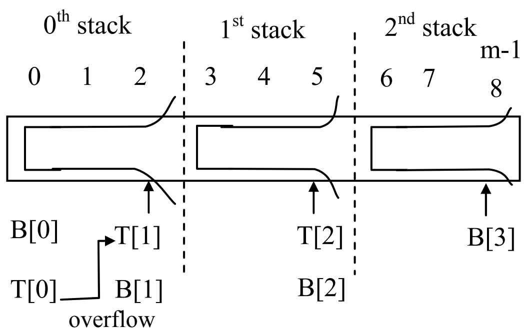
$$i = 1 \Rightarrow T[1] = B[1] = 1\left\lfloor \frac{9}{3} \right\rfloor - 1 = 3 - 1 = 2$$

$$i = 2 \Rightarrow T[2] = B[2] = 2\left\lfloor \frac{9}{3} \right\rfloor - 1 = 2[3] - 1 = 5$$

when $i = 3 \Rightarrow B[3] = m - 1 = 9 - 1 = 8$

(i) Push = overflow = size



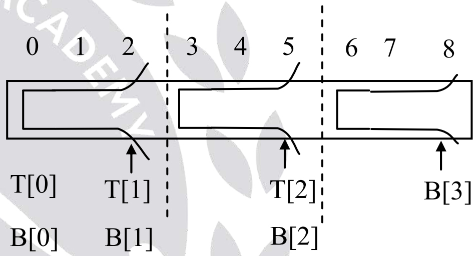


$$\left.\begin{aligned} T[0] &= B[1] \\ T[1] &= B[2] \\ T[2] &= B[3] \end{aligned}\right\} \text{overflow cases}$$

$\therefore$ T[i] = B [i +1]

(ii) POP = underflow = initial

	0	1	2
B	–1	2	5
T	–1	2	5
	0	3	6
	1	4	7
	2	5	8



$$\left.\begin{aligned} T[0] &= B[0] \\ T[1] &= B[1] \\ T[2] &= B[2] \end{aligned}\right\} \text{underflow cases} \;\; \therefore\; T[i] = B[i]$$

02. Ans: (b)

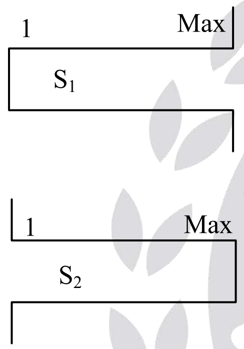
Sol:

Stack operation	Push (10)	Push (20)	Pop	Push (10)	Push (20)	Pop	Pop	Pop	Push (20)	Pop
Stack	10	10 20	10	10 10	10 10 20	10 10	10		20	
Pop list			20	20	20	20, 20	20, 20, 10	20, 20, 10, 10	20, 20, 10, 10	20, 20, 10, 10, 20

The sequence of popped out values $\Rightarrow$ 20, 20, 10, 10, 20

03. Ans: (d)

Sol:

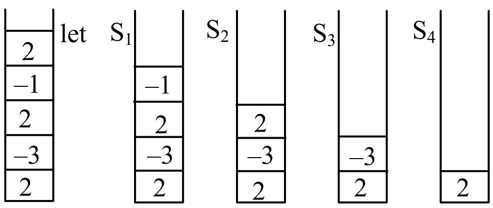


An instance of array having two stacks is shown above. Stack1 occupied from 1 to MAXSIZE and stack2 occupied from MAXSIZE to 1. Above shown array is filled completely. So condition for 'stack full' is

Top 1 = Top 2 – 1

04. Ans: (c)

Sol: Stack S is



f (0)= 0

$f(S_4) = \max\{f(0),0\} + 2 = 2$

$f(S_3) = \max\{f(S_4),0\} + -3 = -1$

$\{\because S_3 = \text{push}(S_4, -3)\}$

$f(S_2) = \max\{f(S_3),0\} + 2 = 2$

$f(S_1) = \max\{f(S_2),0\} + (-1) = 1$

$f(S) = \max\{f(S_1),0\} + 2 = 3$

$f(S) = 3$

05. Ans: (b)

Sol: Stack insertion order $\Rightarrow$ 1,2,3,4,5. The only possible output sequence 3,4,5,2,1

That occurs when

Push (1)
Push (2)
Push (3)
Pop (3) $\rightarrow$ 3

($\because$ There is no constraint on the order of deletion operations)

Push (4)
Pop (4) $\longrightarrow$ 3, 4
Push (5)
Pop (5) $\longrightarrow$ 3,4,5
Pop (2) $\longrightarrow$ 3,4,5,2
Pop (1) $\longrightarrow$ 3,4,5,2,1

Other remaining combinations are not possible

06. Ans: 321

Sol: invocation tail (3)

T (3) = 3

T (2) = 2

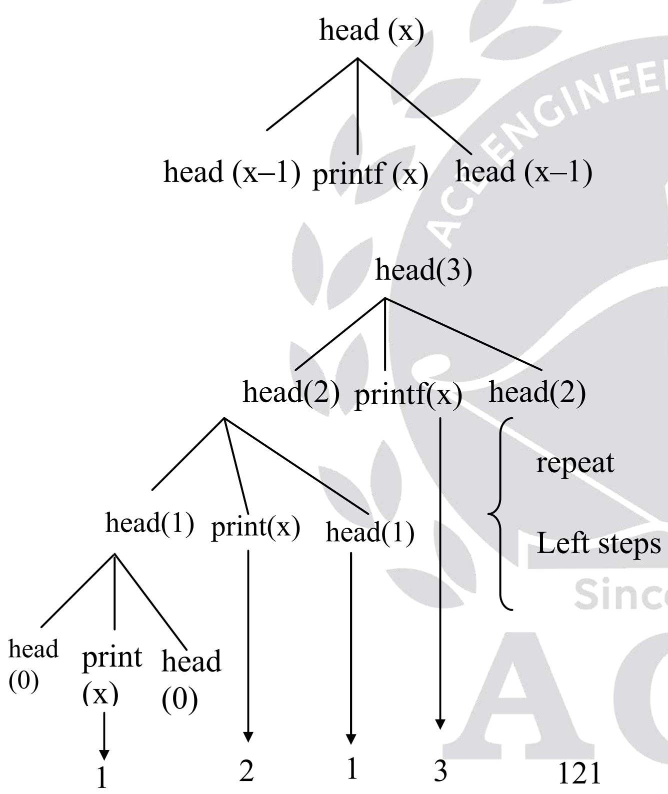
T (1) = 1

T (0) = stop

Output: 3, 2, 1

07. Ans: 1213121

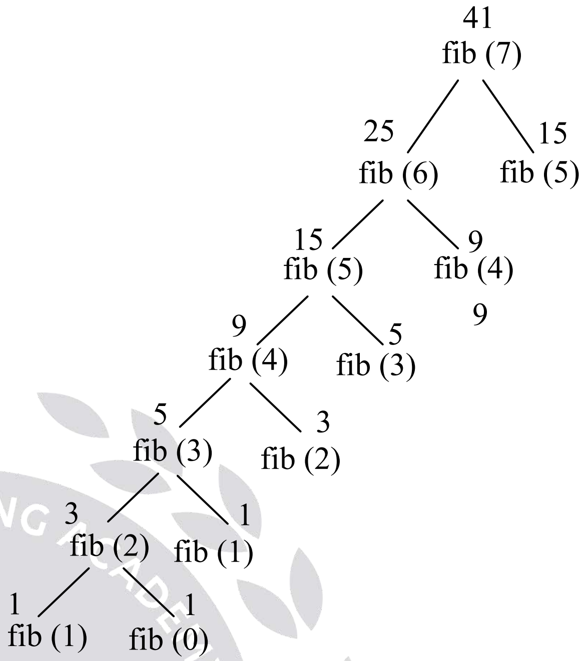
Sol:



Output: 1213121

08.

Sol: (i) Ans: 41



(ii) Ans: 67

Sol: Number of calls for evaluating

f(n) = 2 ×f (n+1) – 1

The total number of calls in

Fibonacci (8) = 2 f(9) – 1

= 2 × 34 – 1 = 68 – 1 = 67

(iii) Ans: 54

n	0	1	2	3	4	5	6	7	8	9	10
Fib(n)	0	1	1	2	3	5	8	13	21	34	55
Call:	1	1	3	5	9			41	35		
Add:				2	4						

Additions = f (n+1) – 1

f(9) = f(10) –1 = 55 – 1 = 54

09.

Sol: Ackerman(m , n) =

$$\begin{cases} n+1 & \text{if } m=0 \\ \text{Ackerman}(m-1,1) & \text{if } n=0 \\ \text{Ackerman}(m-1,\text{Ackerman}(m,n-1)) & \text{otherwise} \end{cases}$$

(i) Ans: 9

Sol: Ackerman(2, 3) = A(1, A(2, 2)) = A(1, 7)

A(2,2) = A(1, A(2,1)) = A(1, 5) = 7

A(2,1) = A(1, A(2,0)) = A(1, 3) = 5

A(2, 0) = A(1,1) = 3

A(1, 1) = A(0, A(1, 0)) = A(0, 2) = 2 +1 = 3

A(1, 0) = A(0, 1) = 2

A(0, 1) = 1 + 1 = 2

A(1,3) = A(0,A(1, 2)) = A(0, 4) = 4 + 1 = 5

A(1,2) = A(0,A(1, 1)) = A(0, 3)= 3 + 1 = 4

A(1, 5) = A(0, A(1, 4))

= A(0, A(0, A(1, 3)))

= A(0, A(0, 5))

= A(0, 6) = 6 +1 = 7

A(1, 7) = A(0, A(1, 6))

= A(0, A(0, A(1, 5)))

= A(0, A(0, 7))

= A(0, 8) = 9

Ackerman(2, 3) = 9

(ii) Ans: 13

Sol: Ackerman(2, 5) = A(1, A(2, 4))

= A(1, A(1, A(2, 3)))

= A(1, A(1, 9))

A(1, 9) = A(0, A(1, 8))

= A(0, A(0, A(1, 7)))

= A(0, A(0, 9))

= A(0, 10)

= 11

A(1, 11) = A(0, A(1, 10))

= A(0, A(0, A(1, 9)))

= A(0, A(0, 11))

= A(0, 12)

= 13

Ackerman(2, 5) = 13

(iii) Ans: 4

Sol: Ackerman(0, 3) = 4

(iv) Ans: 5

Sol: Ackerman(3, 0) = A(2, 1)

A(2,1) = A(1, A(2, 0))

A(2, 0) = A(1, 1)

A(1, 1) = A(0, A(1, 0))

A(1, 0) = A(0, 1) = 2

A(1, 1) = A(0, 1) = 3

A(2, 0) = 3

A(2, 1) = A(1, 3)

A(1, 3) = 5 from (i)

A(2, 1) = 5

Ackerman(3, 0) = 5

10.

Sol: (a) After N +1 calls we have the first move. So after 4 calls we have the first move.

(b) After total calls –1 calls, we have the last move.

(c) Total moves $2^N - 1 = 7$

(d) Total invocations = $2^{N+1} - 1$

$= 2^4 - 1 = 15$

11. Ans: (b)

Sol: Postfix expression A B C D + * F /+DE * +

12. Ans: (a)

Sol: a = – b + c * d/e + f ↑ g ↑ h – i * j

Prefix:

a = – b + c * d/e +(↑f ↑gh) – i * j

a = – b + *cd/e + (↑f↑gh) – i * j

a = – b + /*cde + ↑f↑gh – *ij

a = +–b/*cde + ↑f↑gh – *ij

a = ++–b/*cde↑f↑gh – *ij

a = – + + – b/*cde ↑f↑gh*ij

⇒ = a – + + – b/*cde ↑f ↑gh*ij

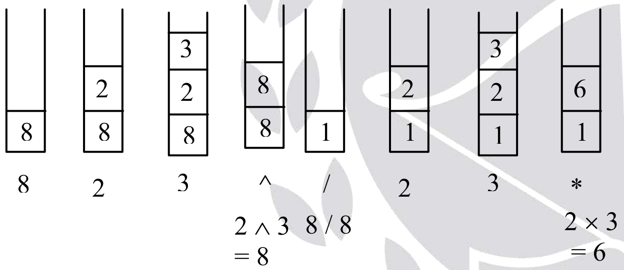
13. Ans: (a)

Sol: Infix expression: [a+(b×c)]–(d^(e^f))

Postfix expression: abc × +def ∧∧–

14. Ans: (a)

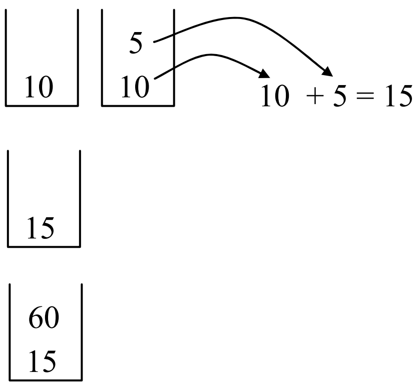
Sol:

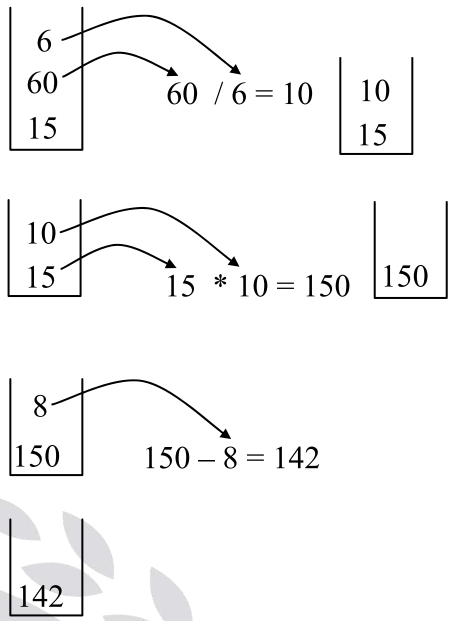


⇒ The top two elements are 6, 1

15. Ans: (c)

Sol: 10, 5, +, 60, 6, /, *, 8, –





Value of the postfix expression is 142

16. Ans: (c)

Sol: (i) ab

(ii) b

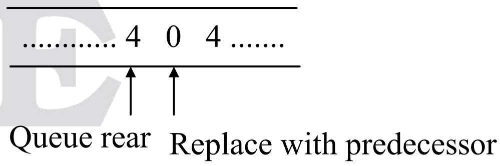
(iii) byz

(iv) yz

Output is ⇒ yz

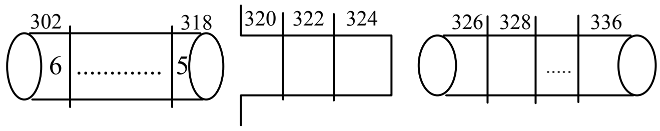
17. Ans: (b)

Sol:

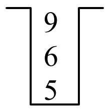


18. Ans: (i) 322 and (ii) 326

Sol:

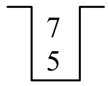


Until first '0' is encountered, stack contains



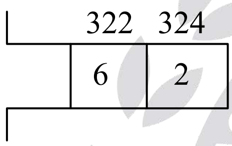
So 5+6+ 9 = 20 is enqueued in Q_2 @ loc 326

Until second '0' is encountered, stack contains



So 5 + 7 = 12 is enqueued is Q_2 @ loc 328

Then simply 2 and 6 are pushed in stack

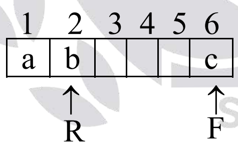


So the location of 6 and 20 are 322 and 326

19. Ans (c)

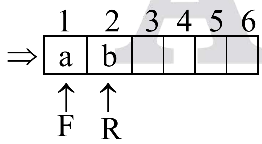
Sol: Suppose that array contains

Initial configuration:



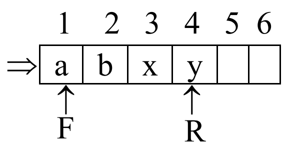
Delete element

Dequeue( ) ↘ c



Now two added

enqueue (x) and enqueue (y) :



$\therefore$ (R, F) = (4,1)

Option (c).

20. Ans: (b)

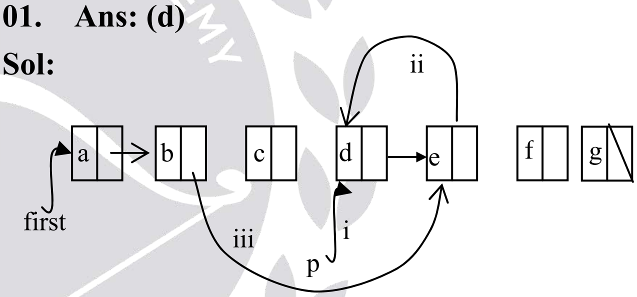
Sol: The given recursive procedure simply reverses the order of elements in the queue. Because in every invocation the deleted element is stored in 'i' and when the queue becomes empty.

Then the insert ( ) function call will be executed from the very last invoked function call. So, the last deleted element will be inserted first and the procedure goes on

3. Linked Lists

01. Ans: (d)

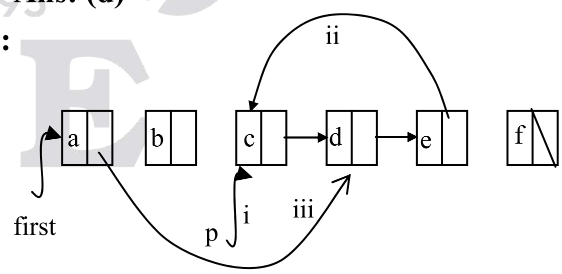
Sol:



Print = d

02. Ans: (d)

Sol:



Print = d

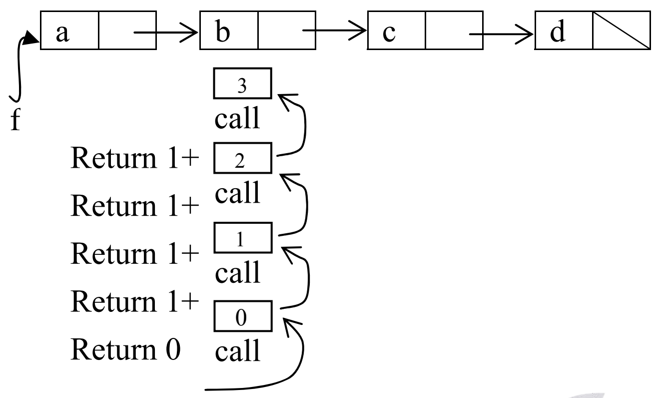
03. Ans: (a)

Sol: while (P) or while (P!= Null)

while P is pointing to somebody

04. Ans: (d)

Sol: Recursive routine for 'Count'



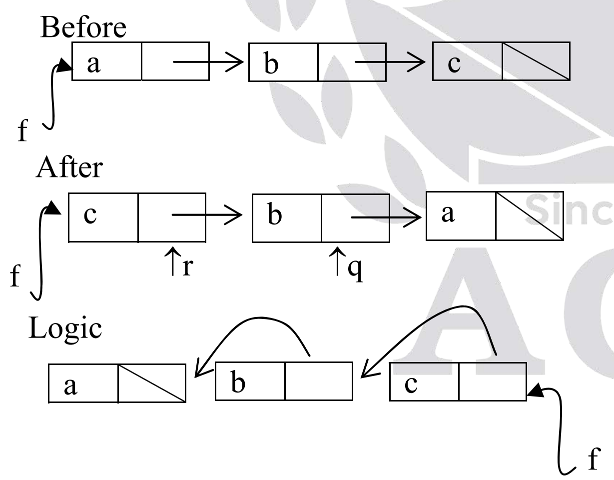
No. of nodes = 4

05. Ans: (b)

Sol: either causes a null pointer dereference or append list m to the end of list n.

06. Ans: (b)

Sol:

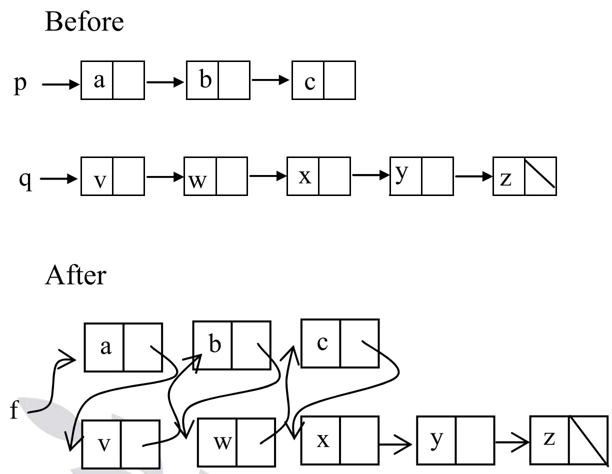


07. Ans: (b)

Sol: This is recursive routine for reversing a SLL.

08. Ans: (a)

Sol:



concatenation of two single linked lists by choosing alternative nodes.

09. Ans: (d)

Sol: Cur. Next = New Node (X, Cur. Next)

(i) Struct Node * n = Get Node ( ) ;

(ii) n → data = X ;

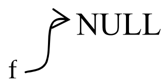
(iii) n → Next = Cur → Next ;

(iv) Cur → Next = n

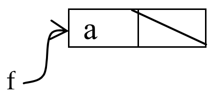
10. Ans: (a)

Sol: Linked stack push ( ) = insert front ( )

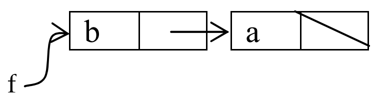
Initial **Do( )**



Do(a)



Do(b)



11.

Sol:

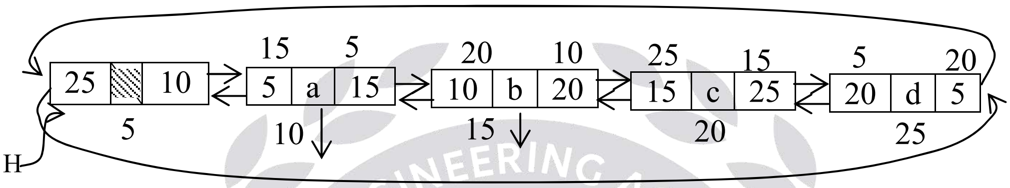
Operation	Left most	Right most	Middle
Insert	3	3	4
Delete	1	1	2

12. Ans: (b)

Sol: Inserts to the left of middle node in doubly linked list.

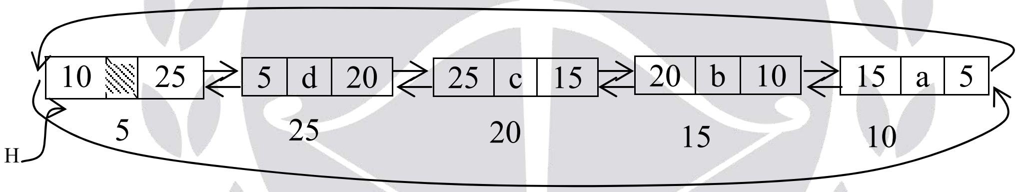
13. Ans: (a)

Sol: Before reverse:



t = t→ Llink

After reverse:



4. Trees

01. Ans: (d)

Sol: 1. Traverse the left subtree in postorder.

2. Traverse the right subtree in postorder.

3. Process the root node

02. Ans: (c)

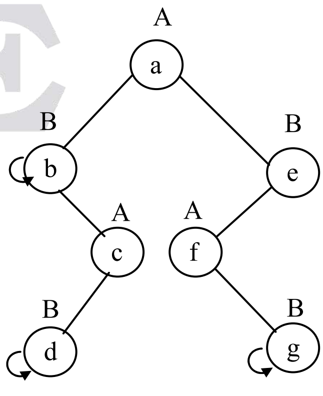
Sol: 1. Traverse the right subtree in postorder.

2. Traverse the left subtree in postorder.

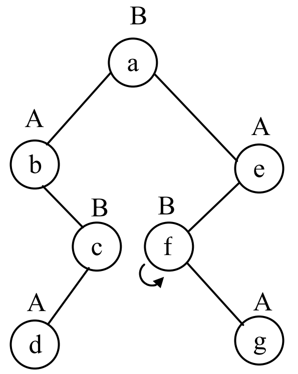
3. Process the root node

03. Ans: (c)

Sol:



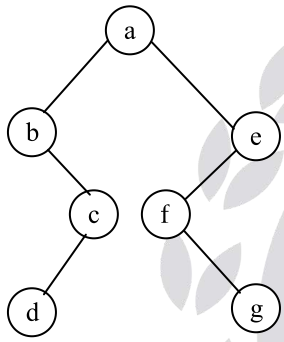
A(t) =a b c d f g e



B(t) =b d c a e f g

04. Ans: (b)

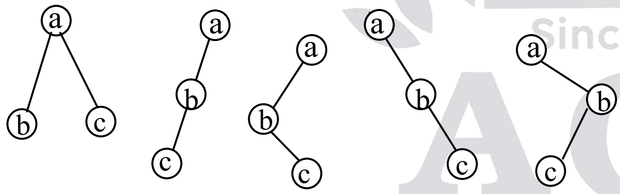
Sol:



B(t) = egfabcd

05. Ans: 5

Sol:



∴ Totally 5 distinct trees possible

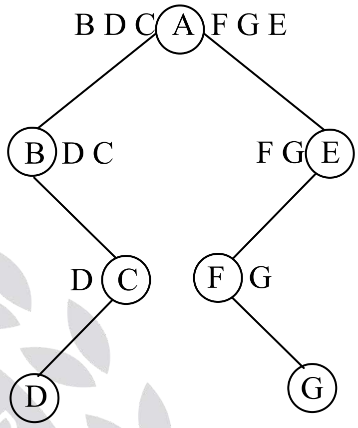
Note: The number of binary trees can be formulated with unlabeled nodes are $\frac{^{2n}C_n}{n+1}$.

06. Ans: (c)

Sol: Preorder : A B C D E F G

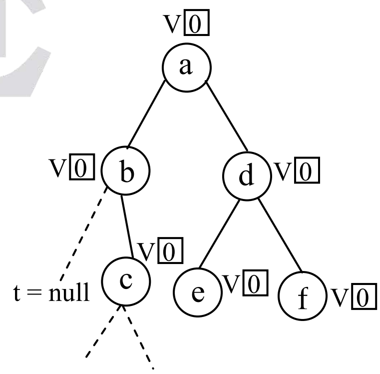
In-order : B D C A F G E

Post-order: D C B G F E A



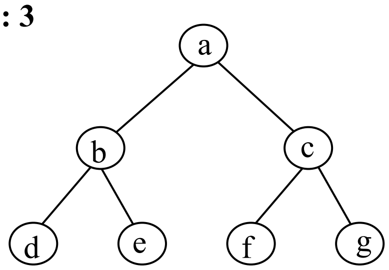
07. Ans: 3

Sol: **Note:** If pre-order is given, along with terminal node information & all right child information the unique pattern can be found. If post-order is given along with terminal information and all left child information the unique pattern can be identified.



08. Ans: 3

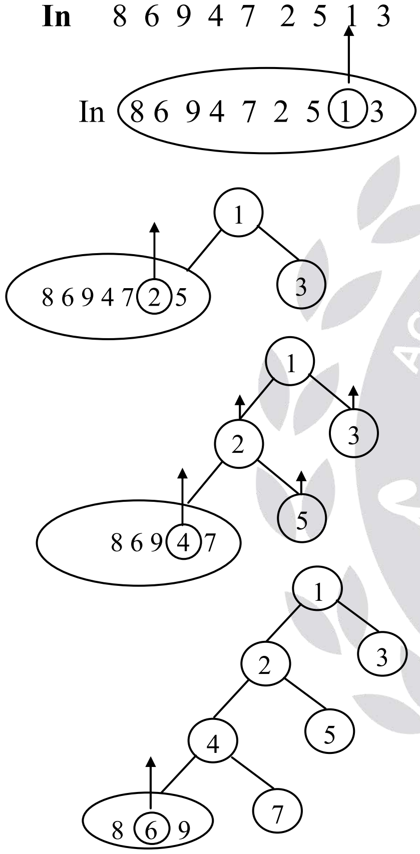
Sol:

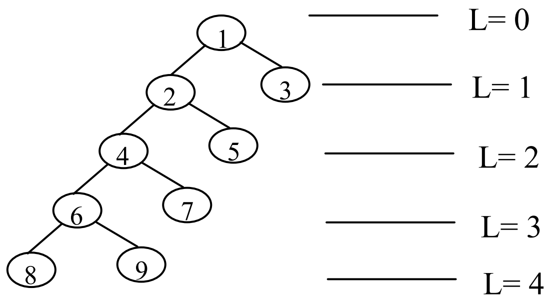


09. Ans: 4

Sol: Post 8 9 6 7 4 5 2 3 1

In 8 6 9 4 7 2 5 1 3





Height = 4

10. (a) Ans: 19

Sol: Leaf nodes (L) = Total nodes – internal nodes

$L = In+1-I$

$L = I(n-1)+1$

$L = 20$

$I = ?$

$20 = I(2 – 1) +1$

$20 = I + 1$

$I = 19$

10. (b) Ans: 199

Sol: $L = I(n-1)+1$

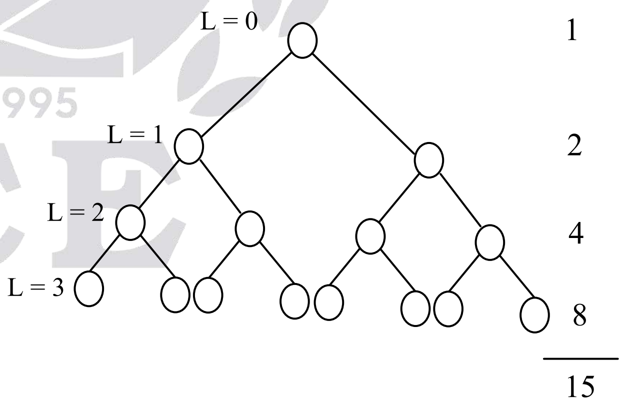
$L = 200$

$200 = I + 1$

$I = 199$

11. Ans: (b)

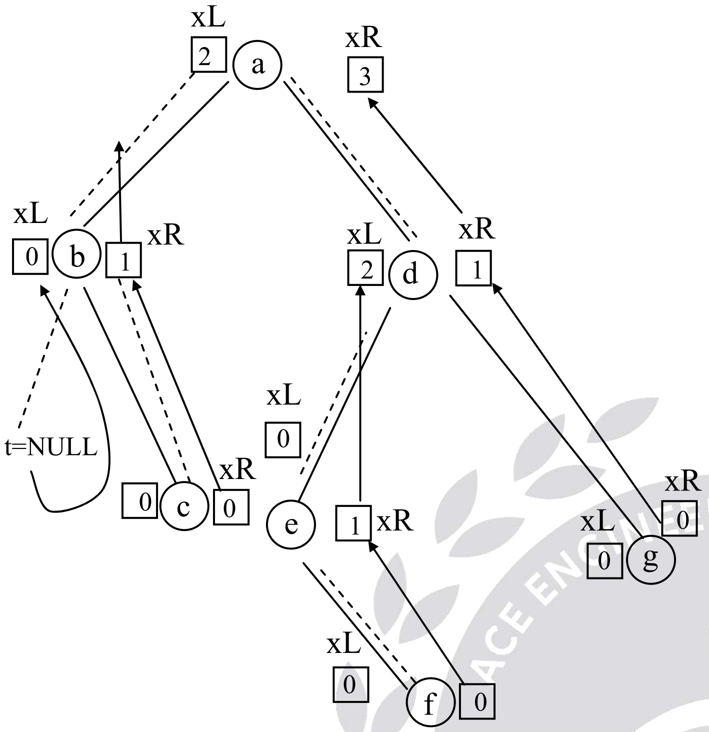
Sol:



Minimum = 3, Maximum = 14

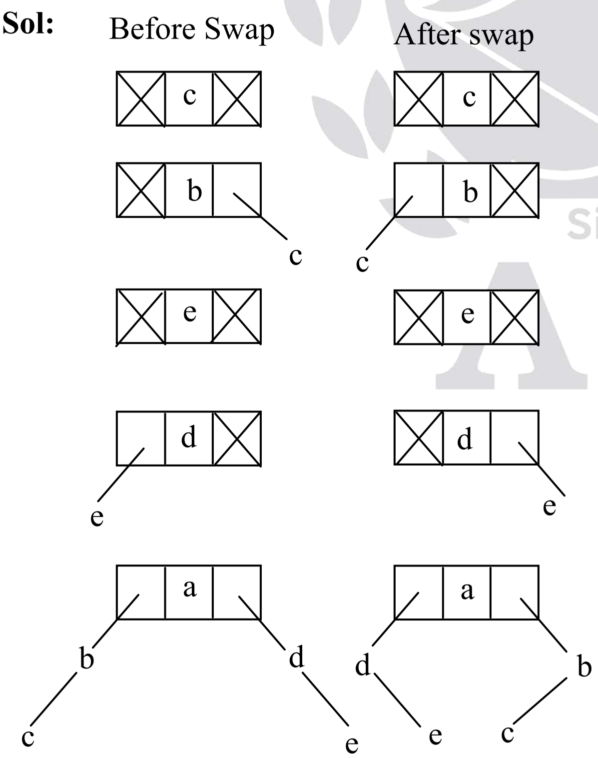
12. Ans: 2 & 1

Sol:



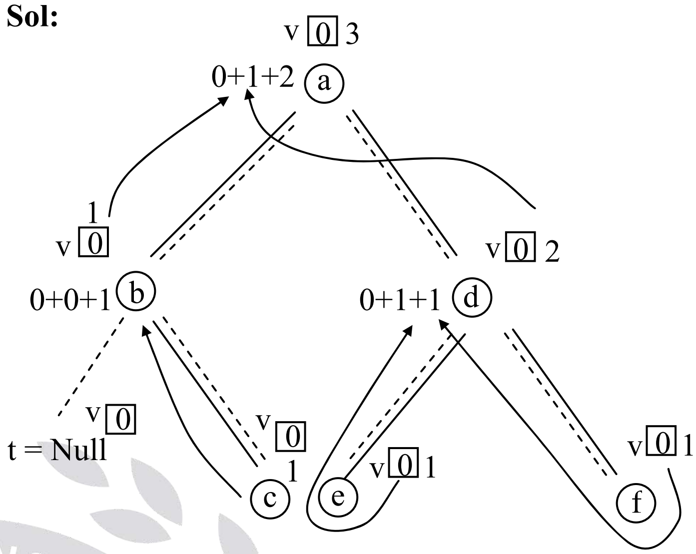
13. Ans: (a)

Sol:



14. Ans: (d)

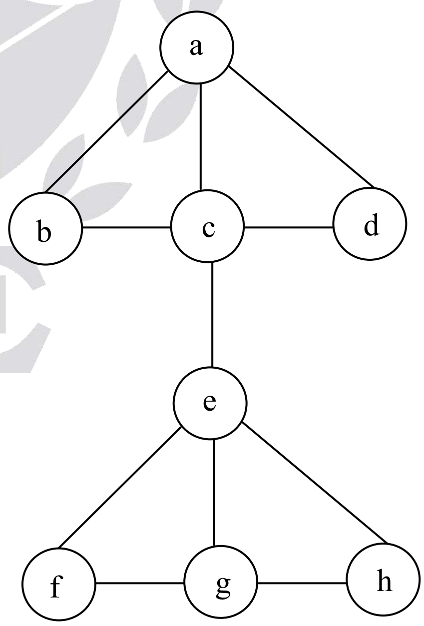
Sol:



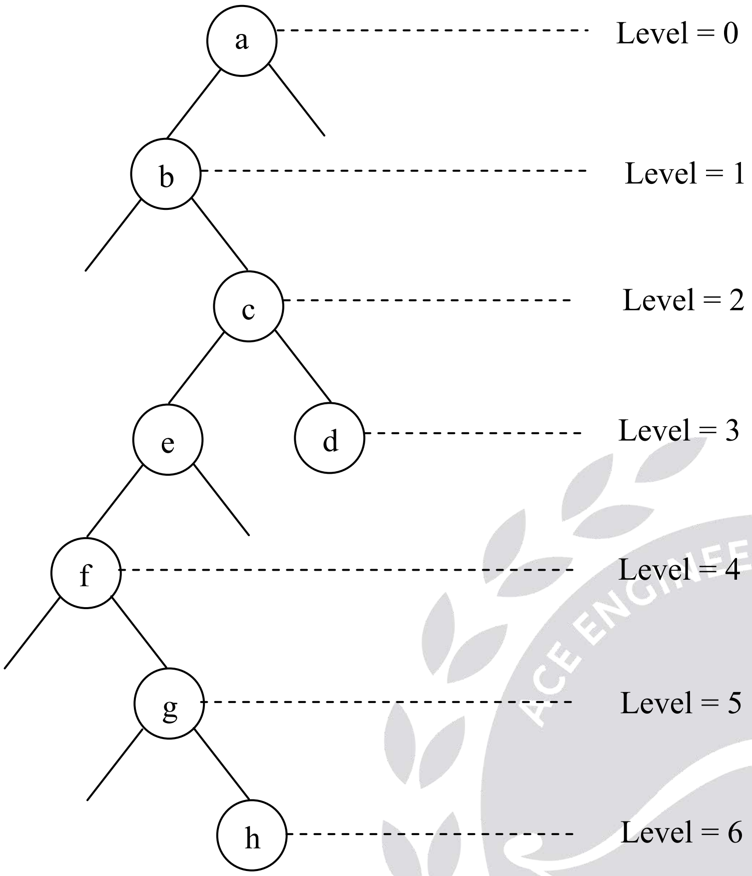
15. Ans: 6

Sol: a (b, c (e (f, g, h)),d)

Parent of f, g, h is e. i.e. internal parenthesis has children of parent which is out of parenthesis.

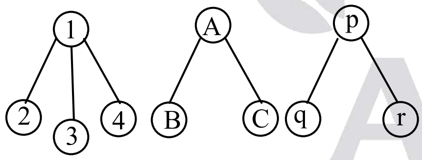


Converted Binary Tree:



16. Ans: 4

Sol: Given are 3 trees

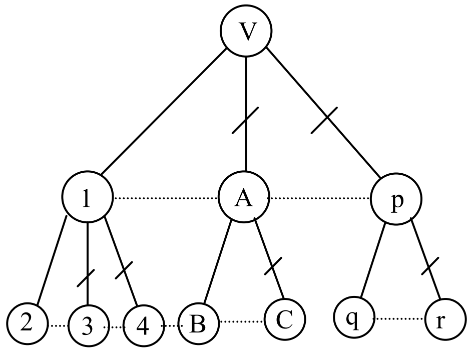


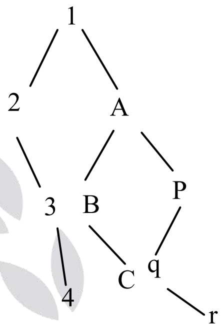
To get the converted binary tree of these given trees

∵ parent is not given we have to assume virtual parent

Among siblings

- Keep the leftmost as it is,
- Cut and connect right siblings as shown in diagram



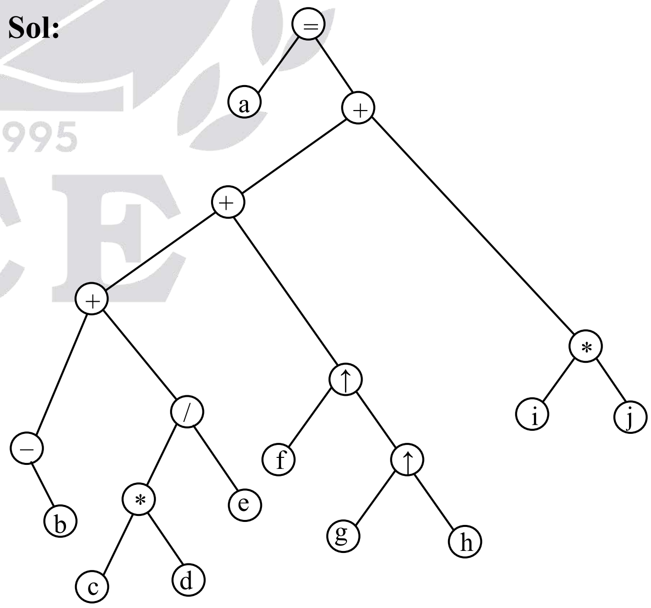


17. Ans: (d)

Sol: Count the number of trees in forest.

18. Ans: (b)

Sol:



19. Ans: 6

Sol: Expanded as

((1+1) – (0 – 1)) + ((1 – 0) + (1+1))

= 3 + 3 = 6

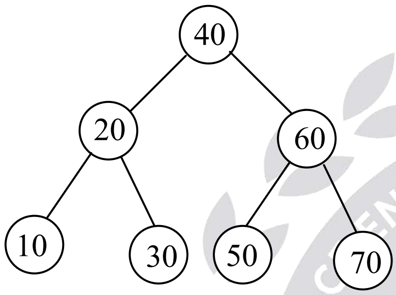
20. Ans: –2

Sol: (0 + 0) – (1 – 0) + (0 – 1) + (0 + 0)

= –1 + (–1) = –2

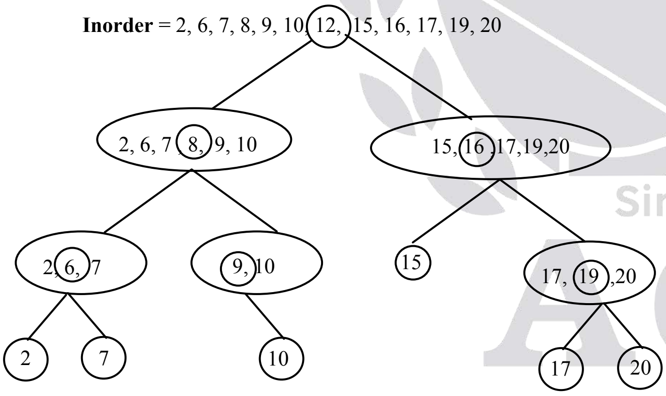
21. Ans: 4

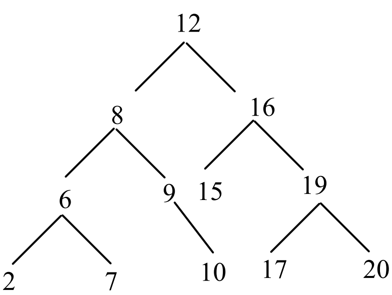
Sol:



22. Ans: (b)

Sol: Preorder = 12, 8, 6, 2, 7, 9, 10, 16, 15, 19, 17, 20

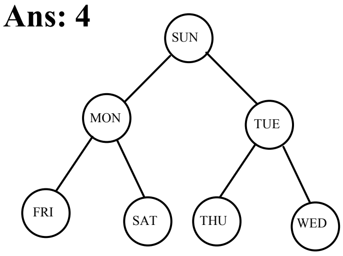




= 2, 7, 6, 10, 9, 8, 15, 17, 20, 19, 16, 12

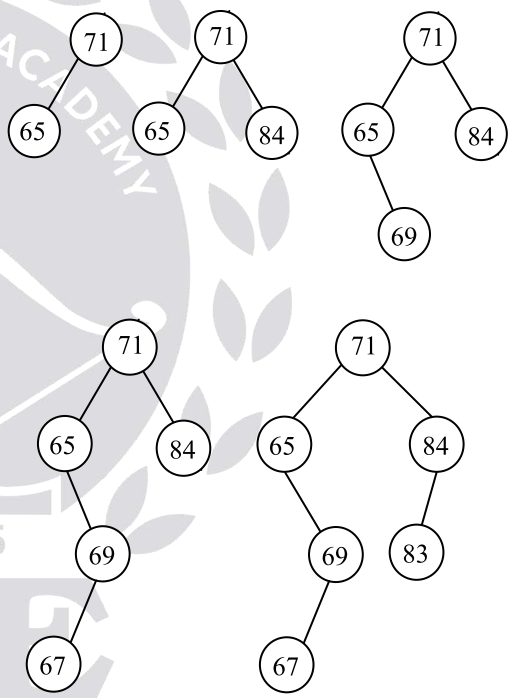
23. Ans: 4

Sol:



24. Ans: 67

Sol: 71, 65, 84, 69, 67, 83 insert into empty binary search tree

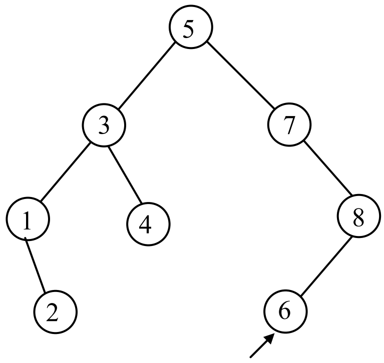


∴ Element in the lowest level is 67

25. Ans: 30

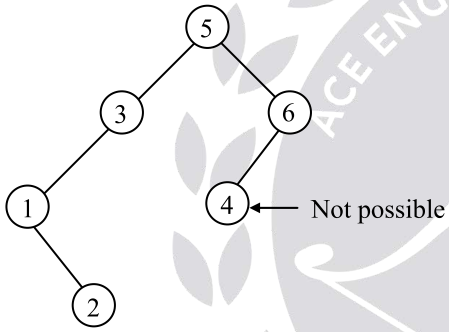
26. Ans: (d)

Sol: (a) 5 3 1 2 4 7 8 6

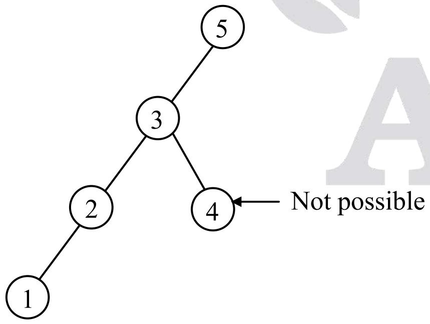


Not possible
IN: not sorted order

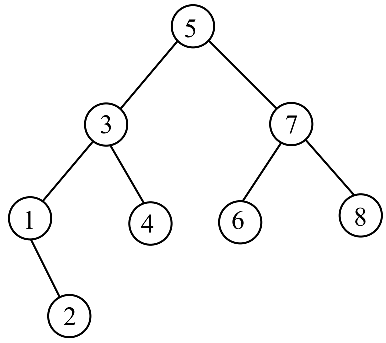
(b) 5 3 1 2 6 4 8 7



(c) 5 3 2 4 1 6 7 8



(d) 5 3 1 2 4 7 6 8



27. Ans: 15

Sol: 1. Jump right

2. Go on descend left

28. Ans: 88

Sol: $$N_{min}(H) = \begin{cases} 1 & L = H = 0 \\ 2 & L = H = 1 \\ 1 + N(H-1) + N(H-2) & (L = H) > 1 \end{cases}$$

N(H) = 1 + N(H-1) + N (H-2)

N(2) = 1 + N(1) + N(0)

= 1 + 2 + 1

= 4

N(3) = 1+N(2)+N(1)

= 1 + 4 + 2

= 7

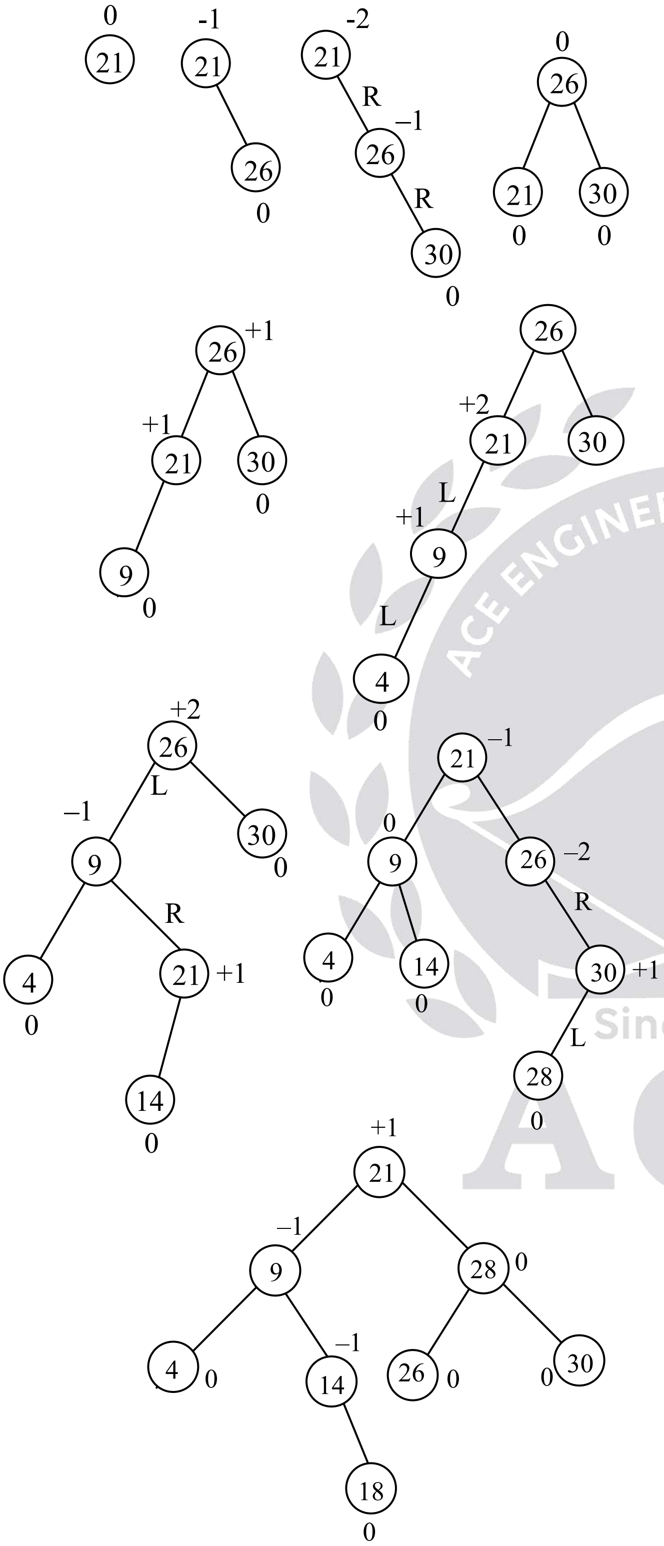
N(8) = 1 + N(7) + N(6)

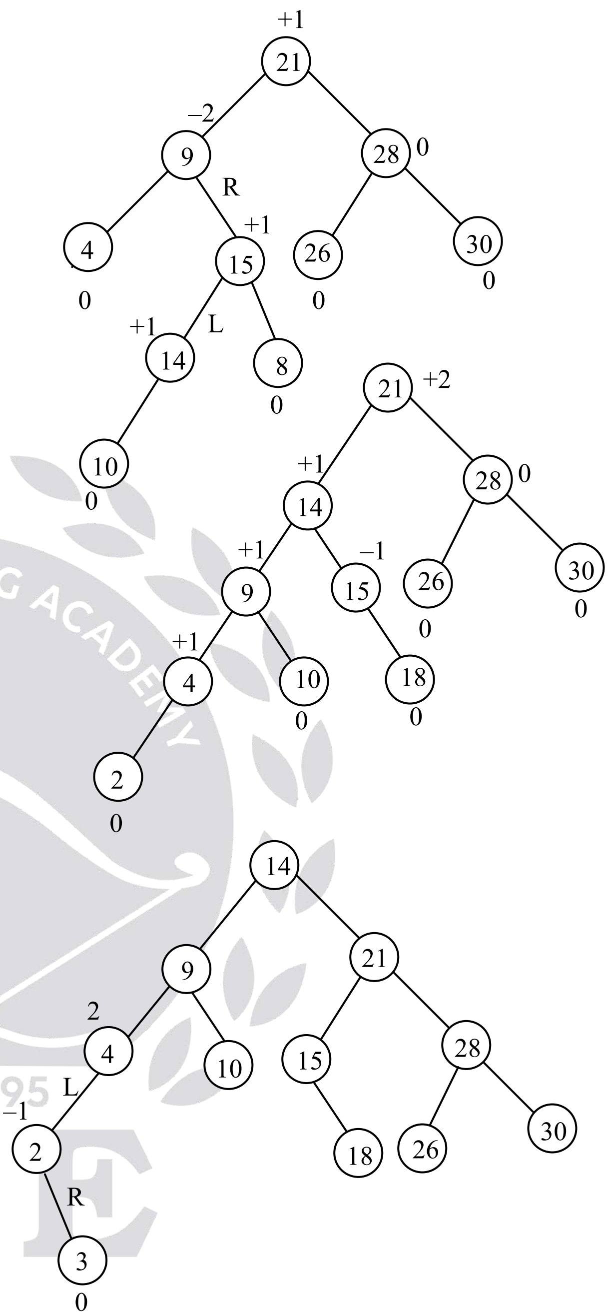
= 1 + 54 + 33 = 88

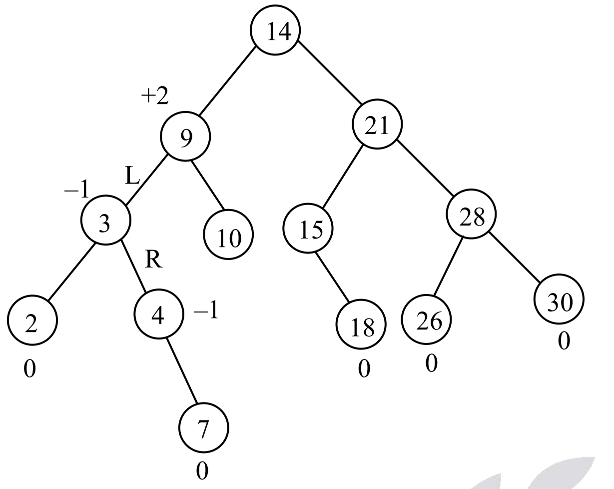
H	0	1	2	3	4	5	6	7	8
N(H)	1	2	4	7	12	20	33	54	88

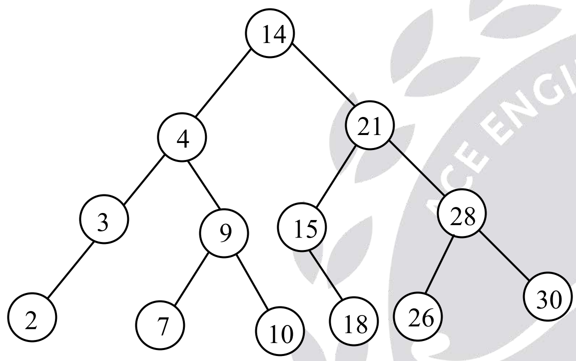
29. Ans: 14

Sol: 21, 26, 30, 9, 4, 14, 28, 18, 15, 10, 2, 3, 7





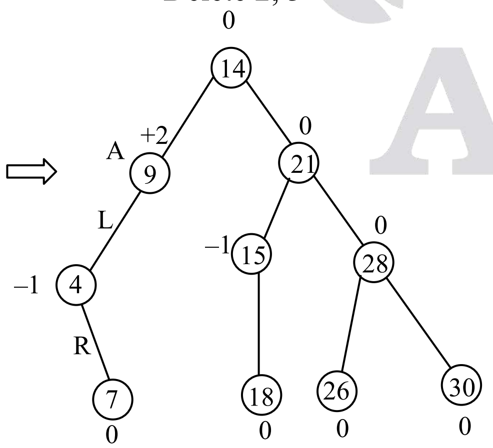




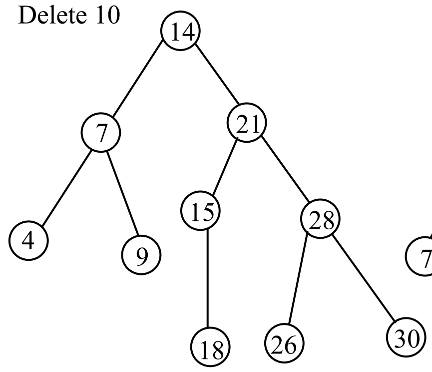
30. Ans: 28

Sol:

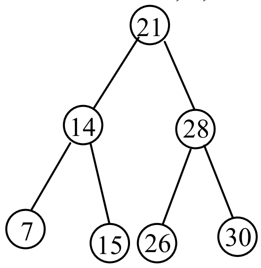
Delete 2, 3



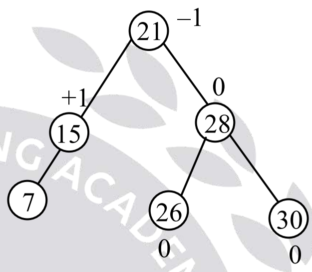
Delete 10



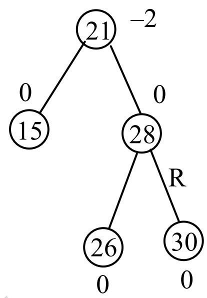
Delete 18, 4, 9



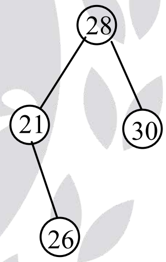
Delete 14



Delete 7



Delete 15

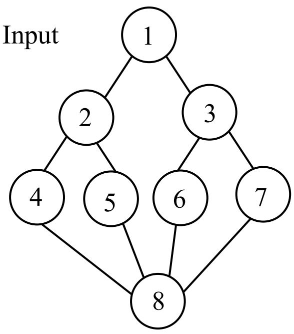


5. Graphs

01. Ans: (b)

Sol: V_8 is pushed in for two times

Input



Output: <u>1, 2, 4, ⑧, 5</u> step back
<u>⑧, 6, 3, 7</u>.

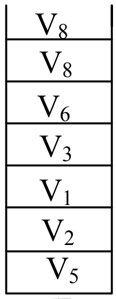
V_8 : Two times

02. Ans: 2

Sol: Sequence of exploration

$V_5 \rightarrow V_2 \rightarrow V_1 \rightarrow V_3 \rightarrow$
$V_6 \rightarrow V_8 \rightarrow V_7 \rightarrow V_4$

Sequence of stack contents



Not pushed vertices are → V_4, V_7

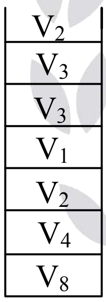
Vertices are pushed in more than once
→ V_1, V_2, V_3, V_6, V_5

03. Ans: 3

Sol: Sequence of exploration

$V_8 \rightarrow V_4 \rightarrow V_2 \rightarrow V_1 \rightarrow V_3 \rightarrow V_6 \rightarrow V_7 \rightarrow V_5$

Sequence of stack contents

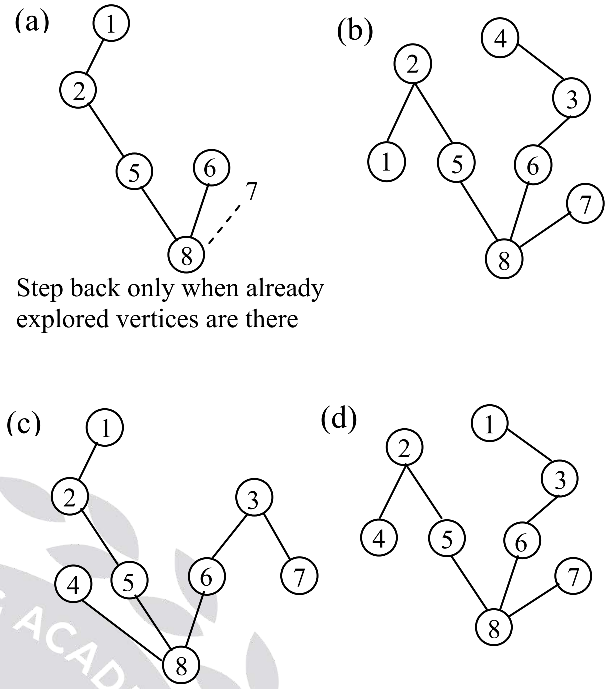


Not pushed vertices are → V_6, V_7, V_5

Vertices are not pushed in more than once
→ V_1, V_4, V_8

04. Ans: (a)

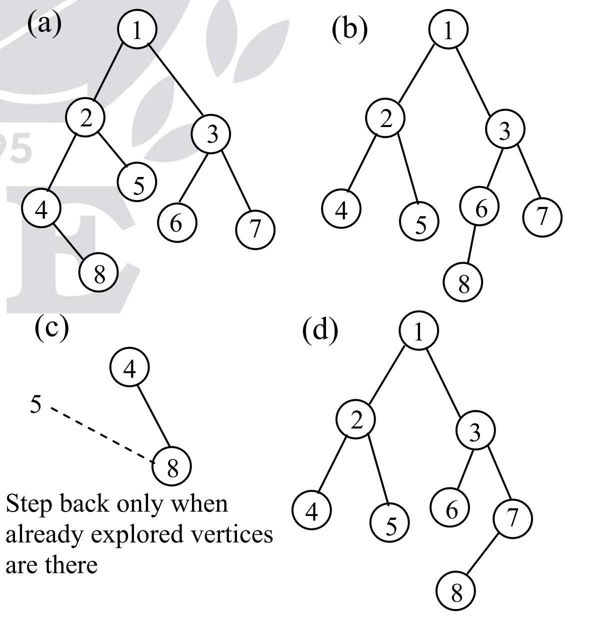
Sol: (a) invalid (b) valid
(c) valid (d) valid



05. Ans:(c)

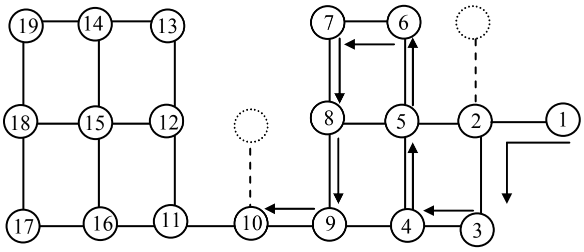
Sol: (a) valid (b) valid

(c) invalid (d) valid



06. Ans: 19

Sol:

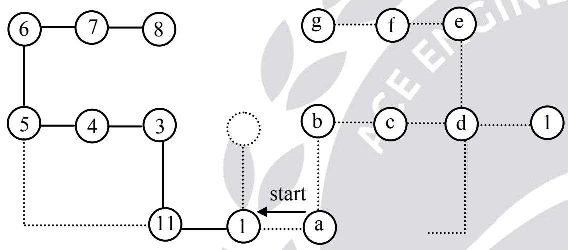


Maximum possible recursion depth = 19

(The dashed link 'nodes' are explored while stepping backward.)

07. Ans: 8

Sol:

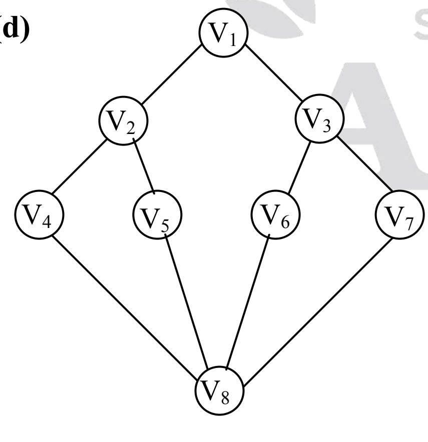


Minimum possible recursion depth = 8

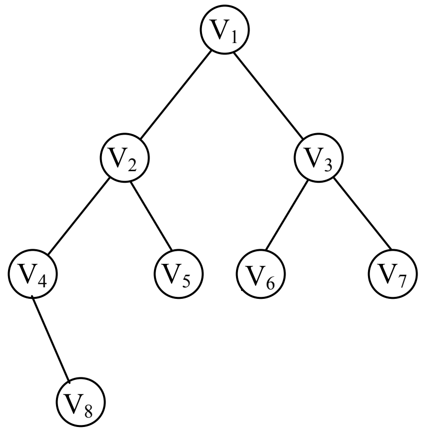
(The dashed link 'nodes' are explored while stepping backward.)

08. Ans: (d)

Sol:



Traversal: BFS



queue

V_2	V_3	

Dequeue

queue

V_3	V_4	V_5	

Dequeue

queue

V_4	V_5	V_6	V_7

Dequeue

queue

V_5	V_6	V_7	V_8

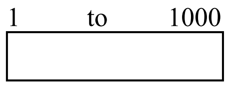
09. Ans: (d)

10. Ans: (d)

6. Hashing

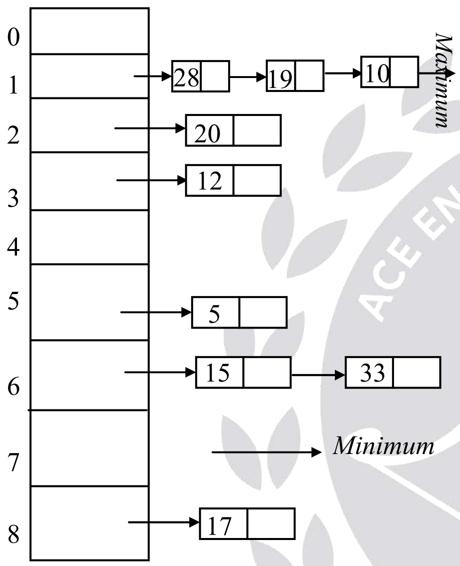
01. Ans: (d)

Sol:



02. Ans: (a)

Sol:



$$\frac{3+1+1+1+2+1}{9} = 1 \text{ (average)}$$

03. Ans: 80

Sol: Slots = 25

Elements = 2000

$$\text{Load factor} = \frac{\text{elements}}{\text{slots}}$$

$$= \frac{2000}{25} = 80$$

04. Ans: (b)

Sol: Hash function

h (x) = (3x + 4) % 7

h (1) = (3+4) % 7 = 0

h (3) = (9 + 4) % 7 = 6

h (8) = (24 +4) % 7 = 0

h (10) = (30 +4) % 7 = 6

Assume Linear probing for collision resolution

The table will be like

0	1	2	3	4	5	6
1	8	10				3

05. Ans: (d)

Sol: After inserting all keys, the hash table is

Key	43	36	92	87	11	4	71	13	14
Loc	10	3	4	10	0	4	5	2	3

0	1	2	3	4	5	6	7	8	9	10
87	11	13	36	92	4	71	14			43

Last element is stored at the position 7

06. Ans: (c)

Sol: Resultant hash table.

In linear probing, we search hash table sequentially starting from the original location. If a location is occupied, we check the next location. We wrap around from the last table location to the first table location if necessary.

07. Ans: (c)

Sol:

	× A	× B	✓ C	× D
0				
1				
2	42	42	42	42
3	52	23	23	23
4	34	34	34	23
5	23	52	52	34
6	46	33	46	46
7	33	46	33	52
8				
9				

08. Ans: (c)

Sol: Case (I): To store 52

Variable part			**Fixed part**		
42	23	34	52	46	33
42	34	23	52	46	33
23	42	34	52	46	33
23	34	42	52	46	33
34	42	23	52	46	33
34	23	42	52	46	33

3! = 6

Case (II): To store 33

Variable part			Fixed part		
42	23	34	52		33
42	23	34		46	

! =24

Since 46 is not getting collided with any other key, it can be moved to the variable part.

Case (I) & Case (II) are mutually exclusive

Case (I) + Case (II) = 24 + 6 = 30

Total 30 different insertion sequences.