S.No. : 04 BS_CS_A_111019

Programming and Data Structures

MADE EASY
India's Best Institute for IES, GATE & PSUs

Delhi | Noida | Bhopal | Hyderabad | Jaipur | Lucknow | Indore | Pune | Bhubaneswar | Kolkata | Patna

**Web:** www.madeeasy.in | **E-mail:** info@madeeasy.in | **Ph:** 011-45124612

# CLASS TEST 2019-2020

## COMPUTER SCIENCE & IT

**Date of Test : 11/10/2019**

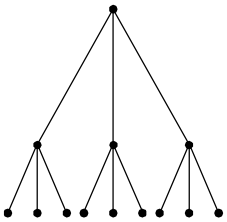
## ANSWER KEY ➤ Programming and Data structures

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 1. | (c) | 7. | (d) | 13. | (c) | 19. | (c) | 25. | (b) |
| 2. | (a) | 8. | (b) | 14. | (c) | 20. | (b) | 26. | (a) |
| 3. | (a) | 9. | (b) | 15. | (a) | 21. | (a) | 27. | (c) |
| 4. | (c) | 10. | (d) | 16. | (d) | 22. | (d) | 28. | (b) |
| 5. | (c) | 11. | (b) | 17. | (a) | 23. | (c) | 29. | (c) |
| 6. | (a) | 12. | (c) | 18. | (c) | 24. | (c) | 30. | (d) |

## Detailed Explanations

**1. (c)**

**Example:** 3-ary tree



d = 0 $\Rightarrow 3^0 = 1$ node (root)
d = 1 $\Rightarrow 3^1 = 3$ leaf nodes
d = 2 $\Rightarrow 3^2 = 9$ leaf nodes, etc.

At depth 5 $\Rightarrow$ The maximum number of nodes = $3^5$
For K-ary tree $\Rightarrow$ The maximum number of nodes at depth d = $K^d$
If 5 is maximum depth of any node of T, then maximum number of leaves = $3^5$.

**2. (a)**

Iterations of for statement

| for: | i = 0 | | i = 1 | | i = 2 | | i = 3 | |
|---|---|---|---|---|---|---|---|---|
| | P | Q | P | Q | P | Q | P | Q |
| P << 1; | 2 | 2 | 2 | 2 | 2 | 3 | 2 | 4 |
| Q = P + i; | 2 | 2 | 2 | 3 | 2 | 4 | 2 | 5 |

P = 2, Q = 5

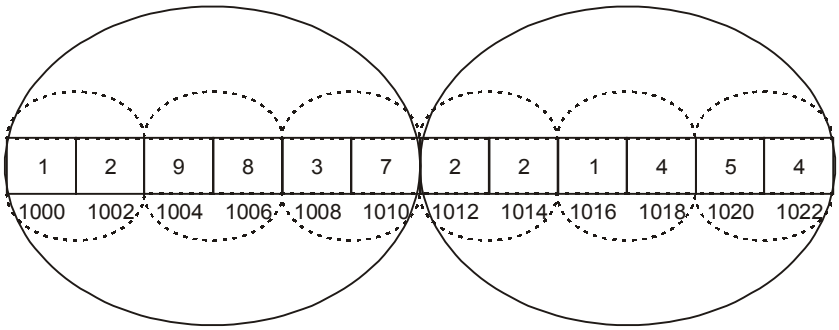
[**Note:** P << 1 will not change the value of P, but P = P << 1 will changes the value of P]

**3. (a)**

int (**p) [ ];

A pointer to a pointer to an array of integers.

**4. (c)**




The output of the statement

printf ("%d%d%d", (a[1] – a[0]), (a[1][0] – a[0][0]), (a[1][0][0] – a[0][0][0]));

it a[1] – a[0] = 1012 – 1000 = $\frac{12}{2}$ = 6 [$\because$ 1012 is the address of 'a[1]' and 2 is the size of element]

Similarly, we can calculate

a[1][0] – a[0][0] = 6

a[1][0][0] – a[0][0][0] = 1

So, the output is 6, 6, 1.

**5. (c)**

(*p) is a constant value, we cannot modify a constant value.

**6. (a)**

Initial values i = –3, j = 2, k = 0.

&& has more priority than ++

∴ m = {(++i) && (++j)} || (++k);

Since both ++i and ++j are non-zero hence expression becomes true.

++k is not checked and m = truth value of expression = 1.

So, i = –2, j = 3, k = 0, m = 1.

**7. (d)**

```
Push (C)
{
    insert C;
}
Pop ()
{
    delete minimum element;
}
```

So for sequenc of operation the key choosen are in strictly decreasing order.

**8. (b)**

The given expression is: $(a - b)\uparrow (p + q) \uparrow (r * s * t)$

$= (ab-) \uparrow (pq+) \uparrow ((rs*) * t)$

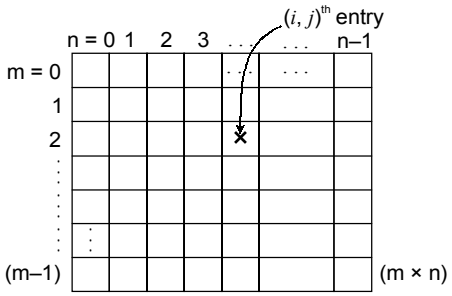
$= (ab-)\uparrow (pq+) \uparrow ((rs*)*t)$

$= (ab-) \uparrow (pq + rs*t*)\uparrow$

$= (ab - pq + rs* t*) \uparrow\uparrow$

$= ab - pq + rs * t * \uparrow\uparrow$

**9. (b)**




It is stored in column major order.

We need to cross $(j - 1)$ columns, since we have 'm' elements in 1 coloum.

∴ $[m(j - 1)]$ + in $j^{th}$ column $(i - 1)$ elements to be crossed.

∴ $(i, j)^{th}$ location is = m * $(j - 1) + i$.

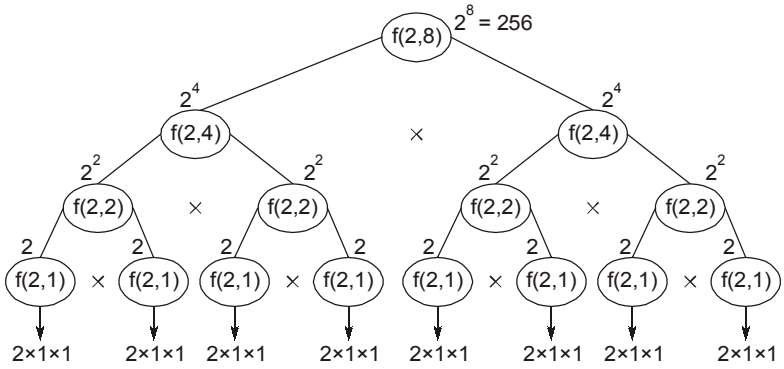
**10. (d)**

**Stack, Queue, Linked list:** Insertion and deletion of an element can be done only at one place.

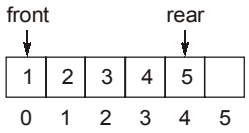
**Deque:** Insertions and deletions can take place at both ends but not in the middle.

**11. (b)**

f(2, 8) returns $2^8$ value = 256




**12. (c)**




[After first **for** loop execution]

$x = 1$ ⇒ q.dequeue ( ); 1 will be deleted form location 0
q.enqueue (q.dequeue ( )); 2 will be deleted and inserted at location 5

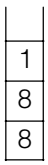
$x = 2$ ⇒ 3 will be deleted from location 2
4 will be inserted at location 0 [circular queue]

$x = 3$ ⇒ 5 will be deleted
2 will be inserted at location 1

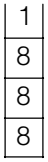

∴ The value present in circular queue = 4 and 2. Therefore the sum = 4 + 2 = 6.

**13. (c)**

To evaluate the postfix expression we used operand stack that is when operand comes push it into the stack, when 1st operator come pop the top two elements of the stack and perform the operation and result push back into the stack.

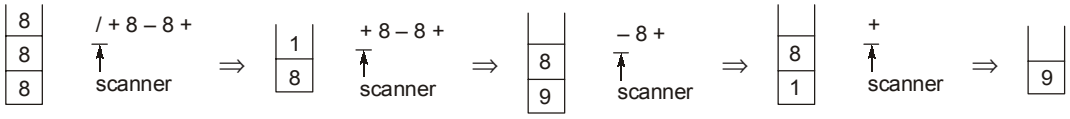



Push the operands one by one when operator comes, POP the top two operands, evaluate the value and then push result on to the stack.




↑ / + 8 – 8 +

Scanner scans power operator then pop 1 and 9, evaluate 9^1 and then push result back on the stack.



**14. (c)**

| | $i$ | $j$ | $k$ |
|---|---|---|---|
| Let | $A[r_1]$ | $[r_2]$ | $[r_3]$ |
| | 91 | 31 | 41 |
| | (planes) | (rows) | (columns) |

**For row major order**

$$\text{loc}(A(i, j, k)) = \text{Base Address} + (i-1)r_2r_3 + (j-1)r_3 + (K-1)$$
$$= 100 + [50-0] \times 31 \times 41 + [30-0] \times 41 + [20-0]$$
$$= 100 + 63550 + 1230 + 20$$
$$= 64900$$

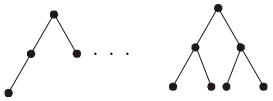
**15. (a)**

The output sequence printed by the given code is: 7 3 5 8 6 0 7 1 8 2.
Last output is : 2.

**16. (d)**

Complete binary tree can make smallest possible height of Binary Search Tree (T).


$\Rightarrow 2 \le n \le 3 \quad \Rightarrow h = 1$ (smallest)


$\Rightarrow 4 \le n \le 7 \quad \Rightarrow h = 2$

Similarly $8 \le n \le 15 \quad \Rightarrow h = 3$

$\therefore$ Smallest height possible: [when complete tree]

$$h = \left\lceil \log_2^{(n+1)} \right\rceil - 1$$

$\therefore$ Largest height possible: [When skew tree]

$$h = n - 1$$

Smallest possible height $= \left\lceil \log_2^{(280+1)} \right\rceil - 1 = 9 - 1 = 8$

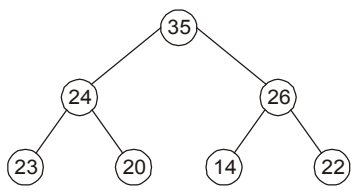
**17. (a)**

In given code 2nd else if statement return 1 when both tree values are different.
It returns 1, when both trees are different
It returns 0, when both trees are same recursively.

**18. (c)**

$$\text{Number of distinct BST's} = \frac{^{2n}C_n}{n+1} = \frac{^{10}C_5}{5+1} = 42$$

**19. (c)**




Only option (c) represents max-heap.

**20. (b)**

q! = NULL is enough to detect the loop in given program.

**21. (a)**

p = 2

q = 3

* (A[0] + 0) = A[0] [0] = *(*(A + j) + i) = 1

* (A[1] + 0) = A[1] [0] = 2

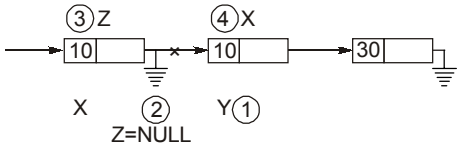
Similarly it will access all the elements.

∴ 1 2 3 4 5 6 is the output printed by the program.

**22. (d)**

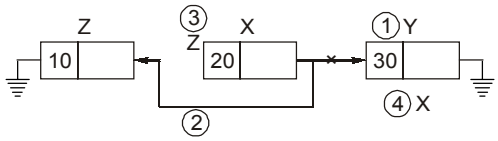
Assume initially X is pointing to the first node.

1. Y = X → next;
2. X → next = Z;
3. Z = X;
4. X = Y;

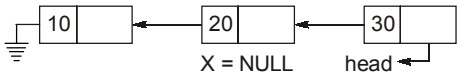



In the first iteration of loop, list is modified as above.

In the second iteration of the loop, second node next is the first, which is shown below.




Similarly after the third iteration, 3rd node next is the second node. After the third iteration the list is reversed as following.




While loop exit due to X = NULL and finally executes *head = Z, so head will be double pointer to the node 30.

∴ list is reversed

**23. (c)**

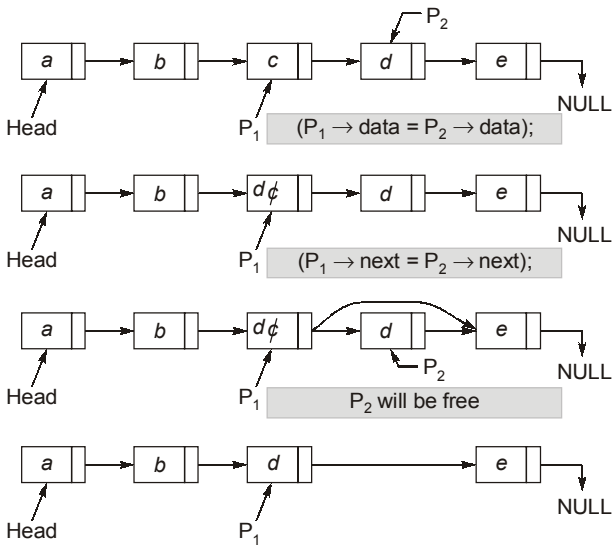
The function prints first n fibbonacci numbers. Note that 0 and 1 are initially there in the queue. This is the initial condition, for fibonacci series. In every iteration of loop, sum of two queue items is enqueued and the front item is dequeued i.e., sum of previous 2 numbers as in fibonacci series.

**24. (c)**

Here two pointers are used gate1 and gate2. Gate 1 pointer points to beginning and gate 2 points to the end. Loop is set up that compares the characters pointed by these two pointers. If the characters do not match, then it's returning 0 i.e. it is not a palindrome.

It returns1 for both even and odd palindrome (i.e., when match occurs for entire loop).

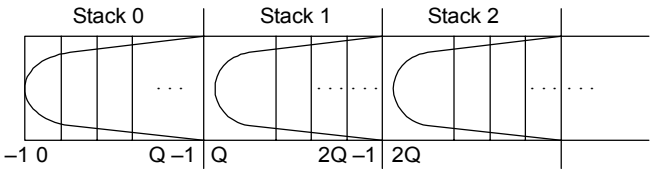
**25. (b)**




**26. (a)**

While performing pop operation on to stack we always perform "Underflow condition check", to ensure if stack is empty.

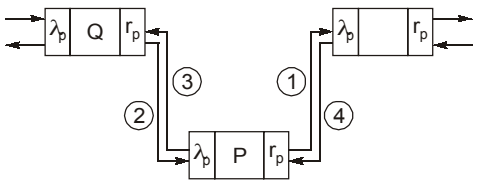
Number of stacks possible is $\frac{P}{Q}$.




First element of next stack pointer $T_{i+1} = \frac{P}{Q}(i+1)$

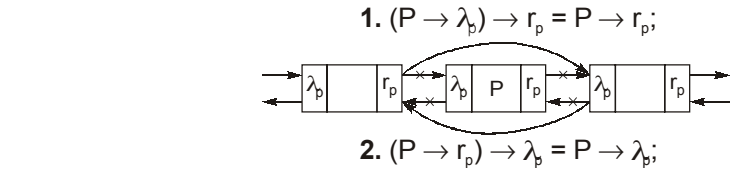
$\therefore$ Underflow condition for stack $i$ is $T_i == \left(\frac{P}{Q} \times i - 1\right)$

**27. (c)**

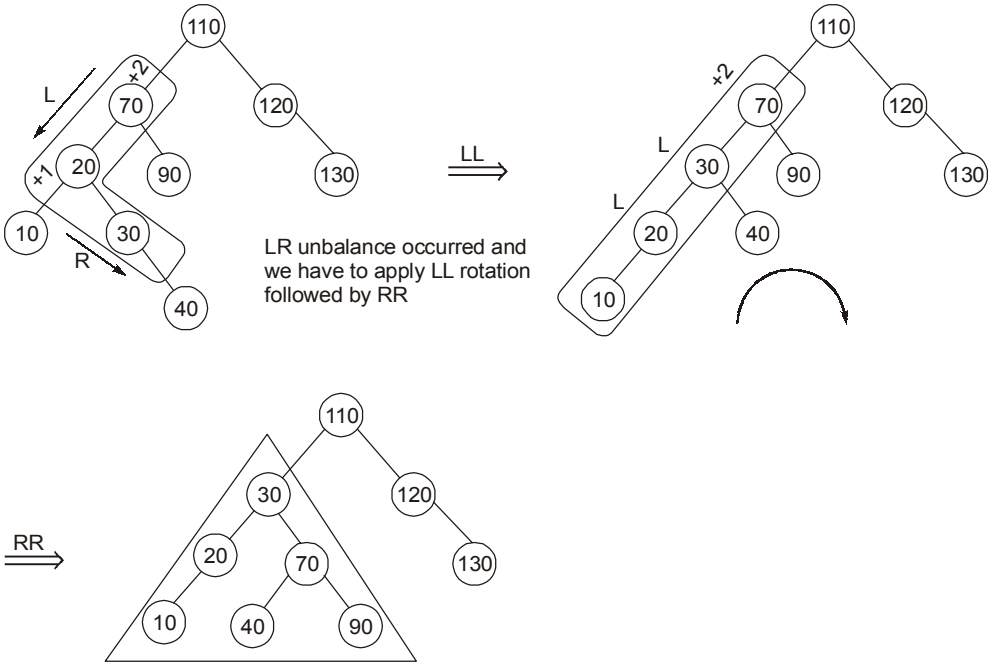



(1) $P \rightarrow r_p = Q \rightarrow r_p$;
(2) $Q \rightarrow r_p = P$;
(3) $P \rightarrow \ell_p = Q$;
(4) $(P \rightarrow r_p) \rightarrow \ell_p = P$;
So option (c) is correct.

28. (b)

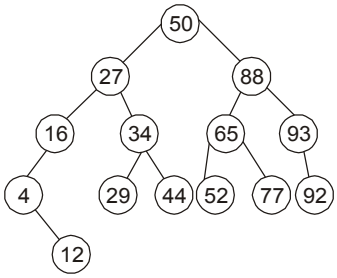



29. (c)




Left subtree postorder traversal is: 10, 20, 40, 90, 70 and 30. So the last element is 30.
Left subtree preorder traversal is: 30, 20, 10, 70, 40 and 90. So the last element is 90.
The difference between last element of postorder and the last element of preorder traversal of left subtree of the resultant AVL tree = 90 – 30 = 60.

30. (d)




Array A: **Preorder:**

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 50 | 27 | 16 | 4 | 12 | 34 | 29 | 44 | 88 | 65 | 52 | 77 | 93 | 92 |

Array B: **Postorder:**

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 12 | 4 | 16 | 29 | 44 | 34 | 27 | 52 | 77 | 65 | 92 | 93 | 88 | 50 |

The one of the longest common sequence between array A and B = 12,34,52,77and 92.
The length of the LCS = 5.

■■■■