CLASS TEST

Sl. : 01SKCS_ABCDFGHI_03082023

MADE EASY

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

Delhi | Bhopal | Hyderabad | Jaipur | Pune | Bhubaneswar | Kolkata

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

PROGRAMMING & DATA STRUCTURES

COMPUTER SCIENCE & IT

Date of Test : 03/08/2023

ANSWER KEY

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

DETAILED EXPLANATIONS

1. (a)

In case of full or complete binary tree,

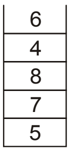
$$\text{Minimum height } (h_{min}) = \lceil \log_2(n+1) \rceil$$

Hence, last element will be stored at $2^{h_{min}} - 1$

$$\text{Minimum size} = 2^{\lceil \log_2(n+1) \rceil} - 1.$$

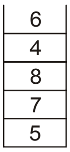
2. (d)

(a) 6, 8, 4, 7, 5



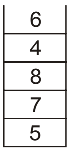
After popping element 6, only 4 can be popped, hence this permutation is not possible.

(b) 6, 4, 5, 7, 8



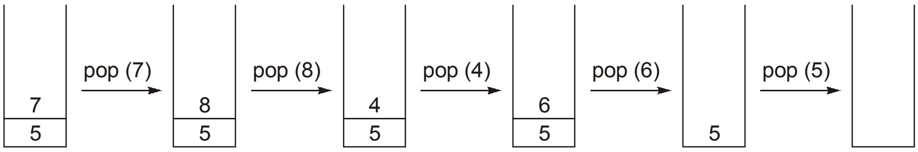
After performing pop operation on element 6, 4 now only element 8 can be popped.

(c) 6, 4, 7, 8, 5



After 6, 4 elements are popped, now element 7 can only be popped iff 8 has already been popped.

(d) 7, 8, 4, 6, 5



3. (d)

An object whose storage class is auto, is reinitialized at every function call whereas an object whose storage class static persist its value between different function calls.

When the function fun ( ) is called for the first time, value of i and j are printed and sequentially incremented. During the second function call, i retains its incremented value whereas j is reinitialized, hence i will print 2 and j will print 5 again. The same will happen at third function call, i will print 3 and j will print 5.

4. (b)

Here m represent the number of rows and n represents the number of column.

m = 2, n = 3

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

$*(A[1] + 0) = A[1][0] = 4$

Similarly it will access all the element.

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

5. (b)

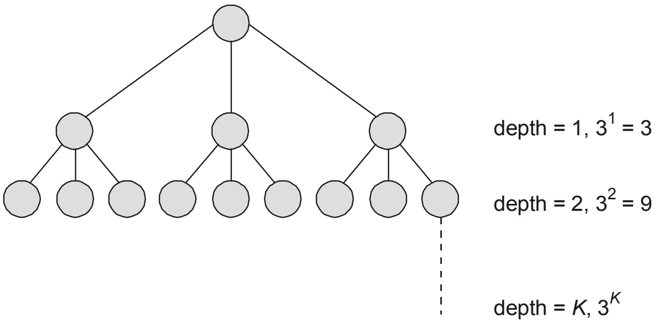
Copy both M_1 and M_2's element in new array of size 2n, then apply Build heap method to make min heap tree which take O (2n) ≅ O(n) time.

6. (a)

Number of children by every node = *n*

depth of tree = *k*

Let *n* = 3



Hence the maximum number of leaves that '*T*' an have in n^k.

7. (b)

Since the left and right subtrees are satisfying min heap property and we want the final output to be a max-heap.

Hence, the answer would remain unaffected.

To create a max-heap time required is O(n).

8. (a)

$$**p\,[10]$$

'*p*' is an array of 10 elements whose each element is a pointer to the pointer of the given return type. Since the precedence of [ ] is greater than *.

9. (a)

Initial value are $p = -3$

$q = 2$

$r = 0$

&& has more priority than ++

$++p = -2$

$++q = 3$

Since, both are non zero, hence expression becomes true. *r*++ need not be checked for calculating '*s*' because it's an OR operation so *s* = 1 i.e. the truth value of the expression.

$$\begin{aligned} t &= p + q + s{+}{+} \\ &= -2 + 3 + 1 \\ &= 2 \end{aligned}$$

10. (b)

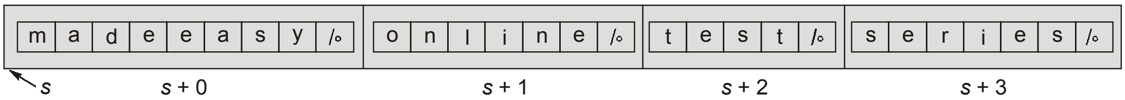
The code represent is the selection sort algorithm on an array.

11. (c)

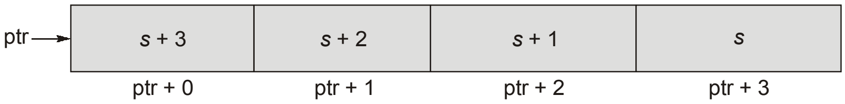
- Merge sort on linked list take O(n log n) time to sort input of length n.
- Merge sort on linked list give better space complexity then on array.
- Inplace merge sort on array will take O(n^2) time.

12. (d)

In this problem we have an array of char pointers pointing to start of 4 strings i.e.,



We have ptr which is pointer to a pointer of type char and a variable *p* which is a pointer to a pointer of type char.



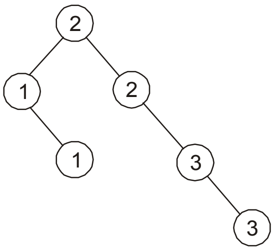
p = ptr; $p\,\boxed{\text{ptr}}$

++*p*; $p\,\boxed{\text{ptr+1}}$

Printf("%s", * – – * ++ *p* + 3);

In printf statement the expression is evaluated *++*p* cause gets value (*s*+1) then now pre-decrement is executed and we get (*s*+1) – 1 = *s*. The indirection pointer now gets the value from the array of *s* and add 3 to the starting address. The string is printed starting from this position. Thus, the output is 'eeasy'.

13. (c)

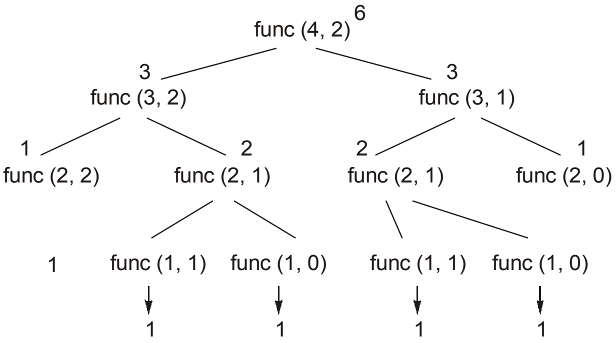


This will be the tree after execution of above code.

The level order traversal of the tree will be 2, 1, 2, 1, 3, 3.

14. (a)

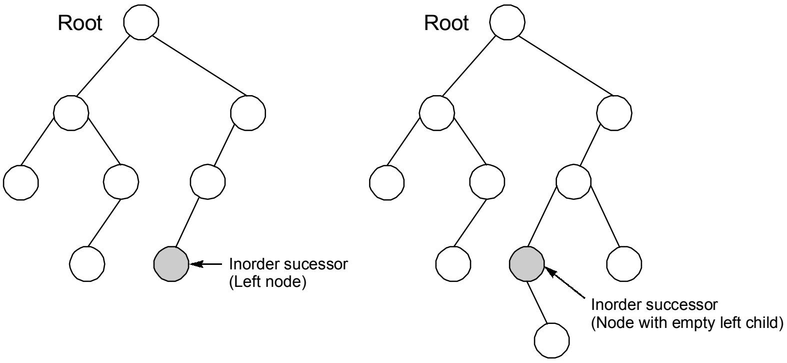
Take m = 4 and n = 2



So, correct vaue of E is func (m – n, n) + func (m – 1, n – 1).

15. (d)

Successor of Root element is always the smallest element of the Right subtree. Because it will be the next largest element after the element to be deleted.



16. (b)

Operation 1

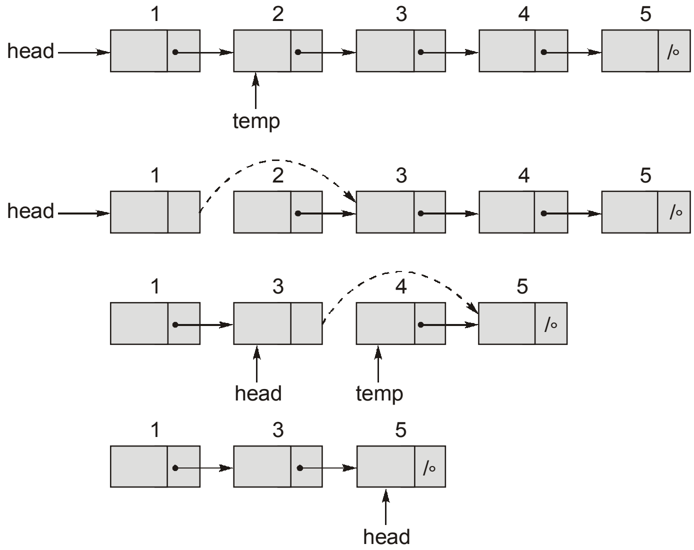
- Search k^{th} element in array take O(1) time.
- Delete take O(1) time
- Shift all element to left if space is there O(n)

  Total time = O(1) + O(1) + O(n) = O(n)

Operation 2

- Take constant time just change first = (P – x)mod n last = P.

17. (c)

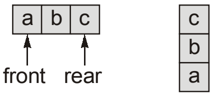


The above program deletes every even number node in the linked list (In particular second, fourth, sixth... soon nodes will be deleted)

18. (c)

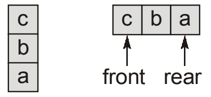
```
while ( !Q. isEmpty ( ))
     S. Push(Q.dequeue ( ));
```

Pushes all Q elements onto S by deleting from Q



```
while ( !S. isEmpty ( ))
     Q.enqueue (S.Pop ( ));
```

Pops stack elements and inserts into Q



∴ Content of queue(Q) will be reversed after the execution of shift(Q).

19. (b)

$$\begin{array}{cc} p & q \\ \boxed{\cancel{50}}\ {}^{\cancel{52}}_{54} & \boxed{20} \\ 1000 & 2000 \end{array}$$

1st function call pass parameters as call by reference

$$\begin{array}{cc} *a & *b \\ \boxed{\cancel{1000}}\ 2000 & \boxed{2000} \\ 3000 & 4000 \end{array}$$

$a = b$ ⇒ Now a is also pointing to 2000 address.
$*a+ = 2$ ⇒ $*a = *a + 2$
⇒ $*a = 20 + 2 = 22$
$a = b$ ⇒ Now a is also pointing to 2000 address.
2nd function call by reference

$$\begin{array}{cc} *a & *b \\ \boxed{1000} & \boxed{1000} \\ 1000 & 6000 \end{array}$$

$a = b$ 'a' will now store b value which is 1000, already contain by 'a'.

- $*a+ = 2;$
  $*a = *a + 2;$
  $*a = 50 + 2$
  $*a = 52$
- $a = b$
  'a' will now store b value which is 1000, already contain by 'a'.

$$\begin{array}{cc} *a & *b \\ \boxed{\cancel{2000}}\ 2000 & \boxed{2000} \\ 3000 & 4000 \end{array}$$

$a = b$ ⇒ Now a is also pointing to 2000 address.
$*a+ = 2$ ⇒ $*a = *a + 2$
⇒ $*a = 22 + 2 = 24$
$a = b$ ⇒ Now a is also pointing to 2000 address.
So, output will be 52 and 24.

20. (d)

Inserting keys in the hash table, we get :

33	0
23	1
68	2
47	3
48	4
60	5
104	6
62	7
19	8
97	9
120	10

Since, there is no collision in the hash table.

- Using linear probing, maximum comparison will be 1.
- Using chaining, the average chain length will be 1.

21. (d)

1000	S
1000	

10	20	30	40	50	60
1000	1001	1002	1003	1004	1005

str 1006 = (int ∗) (&S + 1)

⇒ = (int ∗) (Base address of (S) + 1 ∗ (size of (S))

⇒ = (1000 + 6 B)

⇒ = (1000 + 6)

⇒ = 1006

Now, ∗(S + 2) print 3rd element from start.

∗(Str – 2) print ∗(1006 –2) = ∗(1004) element at address 1004 i.e. 50.

22. (a)

arr[ ] =

GATE%	CAT%	IES%	IAS%	PSU%	IFS%
1000	1004	1008	1012	1016	1020

**ptr = arr ⇒ **ptr = 1000;

*ptr1 = (ptr+ = size of (int)) [–2];

= (1000 + 4) [–2]

= [1000 + 4 × 4] [–2]

= [1016] [–2]

= [1015 – 2 × 4]

*ptr1 = [1008]

print(*ptr1) = IES

23. (d)

1. Using static scoping:

x ~~10~~ ~~150~~ ~~151~~ ~~302~~ 303 Global variable
1000

Part 2 (1000)

x 15 Local variable
2000

**b* (* 1000)
= 10 × 15
= 150

Part 1 (2000)
**a* (*2000)
= 15 + (150 ++)
= 165
print (165)

print (165)
Part 1 (1000)
**a* (*1000)
= 151 + (151 ++)
= 302
print (303)
"165, 165, 303"

2. Using dynamic scoping:

$x\ \boxed{\cancel{10}}\ \cancel{150}\ \cancel{300}\ 301$

1000

Part 2 (1000)

$x\ \boxed{\cancel{15}\ \cancel{30}}\ 31$

2000

$*b = 10 \times 15$

$= 150$

Part 1 (2000)

$*a = 15 + (15\,{+}{+})$

$= 30$

print (31)

print (31)

Part 1 (1000)

$*a = 150 + (150\,{+}{+})$

$= 300$

print (301)

"31, 31, 301"

24. (b)

Consider Random array $a[\ ] = \{1, -2, 1, 1, -2, 1\}$

Output is 2 i.e. $\{1, 1\} = 2$

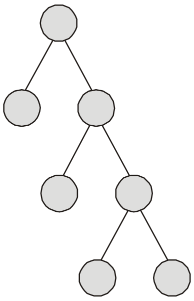
Consider Random array $a[\ ] = \{-2, -3, 4, -1, -2, 1, 5\}$

Output is 7 i.e. $\{4, -1, -2, 1, 5\} = 7$

i.e. sum of largest sum of contiguous sub array.

25. (d)

- Rotation operation in always preserves the inorder numbering so 1st is true.
- AVL tree doesnot guarantee that both left and right subtree has equal number of nodes, so statement is false.
- Consider



satisfying the property of statement 3, in this tree if element present is at last level the time complexity will be $c \times n/2 \simeq O(n)$. So S_3 is false.

- $$\begin{aligned}\text{Total nodes} &= 3 \times \text{internal nodes} + 1\\ &= 3 \times 20 + 1 = 61\end{aligned}$$

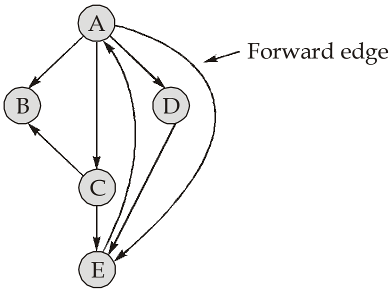
and $20 + 41 = 61$

(Leaf + internal = total) so S_4 is true.

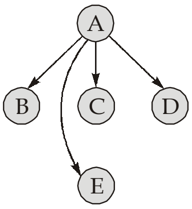
26. (a)

Since for undirected graph, breadth first search does not have back edge and forward edge but for directed graph we have back edge.

Ex: Consider Random Graph (Directed):



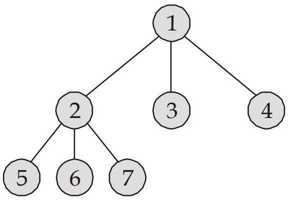
BFS of graph: Assume (A) is start node.



Here in graph no forward edge present, but E → A back edge present, so (b) is false.
Since undirected graph for BFS does not create back edge so statement is false.

27. (b)

Consider 3 array heap:



Array = A	1	2	3	4	5	6	7	....	
	1	2	3	4	5	6	7	....	n

So, root at at A[1] i.e. $d(i-1)+j+1$
$= 3(1-1)+j+1$
$= 0+0+1 = 1$

1st child of root $= d(i-1)+j+1$
$= 3(1-1)+1+1 = 2$

3rd child of root $= d(i-1)+j+1$
$= 3(1-1)+3+1$
$= 0+4 = 4$

Index for node 7 $= d(i-1)+j+1$
$= 3(2-1)+3+1$
$= 3+3+1 = 7$

So, index maps to $d(i-1)+j+1$.

28. (b)

The program X prints the ternary equivalent of 1023. Program P_1 also prints the ternary equivalent of 1023. However, program P_2 prints the ternary equivalent of 1023 in reverse order.

Hence the answer is (b).

29. (b)

The catch here is that, some of the contents of the array are written in octal format. If a number is preceded by a zero, then the number is interpreted as an octal number in C. The code simply adds all the numbers up, and produces the output in decimal format.

Hence, the output will be:

$$\begin{aligned}(0 + 1 + 8 + 9 + 10 + 100) &= 128\\ \text{Thus } \log(128) &= 7\end{aligned}$$

30. (d)

Since P_1 returns the address of a variable which is declared locally, P_1 may cause problems.

P_2 will cause a problem because *px* doesn't have any address and is being dereferenced.

P_3 also will cause problems because even though malloc has been used to allocate the memory into the heap, free( ) has been called and returning that address is simply asking for trouble.

■■■■