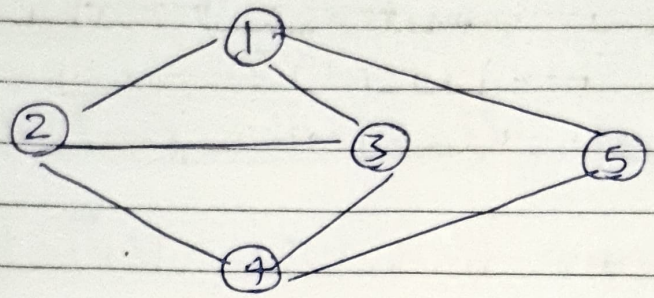
SET-B

NAME- ISHITA CHANDRA
ROLL NO - 08701012020
EXAMINATON PAPER - DS
DATE - 10/01/22 2:00PM

PROG.- BTECH CSE
SEM - 3rd
PAPER CODE- BCS-201

Ans No.1 :
(a)



Adjacency Matrix: Square matrix with rows and cols equal to no. of vertices. Here, there are 5 vertices.

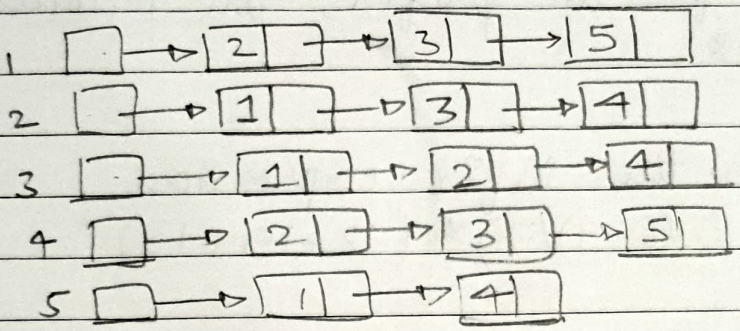
	1	2	3	4	5
1	0	1	1	0	1
2	1	0	1	1	0
3	1	1	0	1	0
4	0	1	1	0	1
5	1	0	0	1	0

Here 0 represents that there is no edge between i and j [i → row, j → col] and adj[i][j] = 1 indicates that there is an edge from i to j

PAGE No
DATE : / 201

08/01/2020

Adjacency list :- Here, we make use of array of lists. The size of the array will be equal to the no. of vertices.



I would recommend to make use of Adjacency list ~~because of the following reasons:~~ when in case of sparse graph

(i) More efficient for storage of graph where there are more nodes and less edges. i.e for sparse graphs.

But adjacency matrix are more efficient for storing ~~sp~~ dense matrix. But

Adjacency matrix is slower when iterating over all edges as well as slow when adding / deleting a node as it is $O(n^2)$

In case of list, it is faster to iterate over all edges and adding/deleting a node

Ans No 1 → (b) 16, 12, 98, 10, 14, 20, 15.

comparisions:

1st Pass

(1) 16, 12, 98, 10, 14, 20, 15 swap 12, 16, 98, 10, 14, 20, 15

(2) 12, 16, 98, 10, 14, 20, 15 No swap, 16 < 98.

(3) 12, 16, 98, 10, 14, 20, 15 swap 12, 16, 10, 98, 14, 20, 15

(4) 12, 16, 10, 98, 14, 20, 15 swap 12, 16, 10, 14, 98, 20, 15

(5) 12, 16, 10, 14, 98, 20, 15 swap 12, 16, 10, 14, 20, 98, 15

(6) 12, 16, 10, 14, 20, 98, 15 swap 12, 16, 10, 14, 20, 15, 98

2nd Pass

(7) 12, 16, 10, 14, 20, 15, 98 No swap, 12 < 16.

(8) 12, 16, 10, 14, 20, 15, 98 swap 12, 10, 16, 14, 20, 15, 98

(9) 12, 10, 16, 14, 20, 15, 98 swap 12, 10, 14, 16, 20, 15, 98.

(10) 12, 10, 14, 16, 20, 15, 98. No swap 16 < 20.

(11) 12, 10, 14, 16, 20, 15, 98. swap 12, 10, 14, 16, 15, 20, 98.

~~(12) 12, 10, 14, 16, 15, 20, 98 No swap 20 < 98.~~

3rd Pass

(12) 12, 10, 14, 16, 15, 20, 98 swap 10, 12, 14, 16, 15, 20, 98

(13) 10, 12, 14, 16, 15, 20, 98 No swap 12 < 14

(14) 10, 12, 14, 16, 15, 20, 98 14 < 16 No swap

(15) 10, 12, 14, 16, 15, 20, 98. swap 10, 12, 14, 15, 16, 20, 98

4th pass :

(16) 10, 12, 14, 15, 16, 20, 98 No swap 10 < 12

17. 10, 12, 14, 15, 16, 20, 98. 12 < 14 No swap

18. 10, 12, 14, 15, 16, 20, 98. 14 < 15 No swap.

5th pass :

19. 10, 12, 14, 15, 16, 20, 98. 10 < 12 No swap.

20. 10, 12, 14, 15, 16, 20, 98. 12 < 14 No swap

6th pass :

21. 10, 12, 14, 15, 16, 20, 98. 10 < 12 No swap.

Array after performing bubble sort is :
10, 12, 14, 15, 16, 20, 98.

Therefore, there are a total of 21 operati~~ons~~ number of comparisions involved
There will be 18 swaps here.

Time complexity

(1) Let 'n' be the size of the array to be sorted

→ we need (n-1) passes to completely sort the array

→ The number of comparision in each pair is decremented by 1
because after each pass, the largest el. is fixed at the end in the correct pos.

so, for n elements the total no. of comp.
$= (n-1) + (n-2) + (n-3) + \ldots$

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

Therefore, we can say that the time complexity in terms of Big O is

$$O\left(\frac{n(n-1)}{2}\right) \text{ or } \underline{\underline{O(n^2)}} \quad \text{Ans.}$$

Answer 2(c) → Infix Exp : (13 + 15 − 12) * (7 / 19 − 6))

For Infix to prefix, the following steps are used :

(i) Reverse the Infix expression
i.e ((6 − 9) / 7) * (12 − 15 + 13)

(ii) we will scan the expression from left to right and when we see operands we will add them to the string.

(iii) When we see an operator −
(a) If stack is empty, push it onto the stack
(b) Otherwise, has higher precedence than top of stack then, push the operator onto the stack
(c) Same precedence − push it onto the stack
(d) Lower precendence − Keep popping and adding top of stack to the string, and finally push it onto the stack.

(iv) After reaching end of the expression, pop & print all op. from top of stack

(v) If operator is ')', push it into the stack
(vi) If operator is '(', pop all the top till ')' is reached

(vii) Reverse in end.

Reversed exp : ((6-9)/7)*(12-15+13)

Stack	String
(← top	
((	6
((	6
((-	69
((-	69-
(	69-
(/	69-7
(/	69-7/
	69-7/
*	69-7/
*(	69-7/12
*(	69-7/12
*(-	69-7/12 15
*(-	69-7/12 15-
*(+	69-7/12 15-13
*(+	69-7/12 15-13+
*	69-7/12 15-13+*

The final prefix expression is after reversing the string

⇒ *+13-15 12/7-96

For evaluation of prefix expression:
- Start from last element of expression.
- Check if its operand, then push it onto stack.
- If it is an operator, pop two operands, perform operation and push result into stack.
- Finally after reaching end of expression the top of the stack is the answer.

Prefix exp: * + 13 - 15 12 / 7 - 9 6

Element scanned - 6
Stack - 6

El. scanned - 9
Stack - 6, 9 ↑ top.

El. scan - -
Op: Pop two operands and perform -.
Op - 9 - 6 = 3.
Push into stack
stack: 3.

El-scan - 7
Stack: 3 7 ↑ top

El scan - /
Op: 7/3
Stack push: 2
Stack: 2

El- scan - 12
Stack : 2 12

El scan- 15
stack = 2 12 15

El scan : -
Op : 15-12 = 3.
Stack = 2 3

El scan : 13
Stack : 2 3 13

El scan : +
Op : 13+3 = 16
Stack : 2 16

El scan = *
op : 16 * 2 = 32
Stack : 32

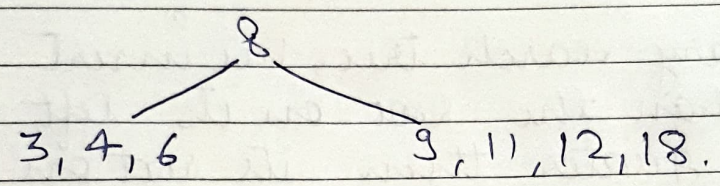
Answer = 32

Ans No. 2 :- 18, 12, 6, 3, 8, 4, 9, 11, 7, 16, 99, 56, 98.

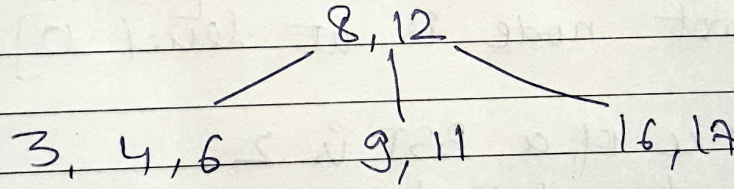
As the order is 5
no. of keys in one node will be 4 (5-1).
Now.

| 3 | 6 | 12 | 18 |

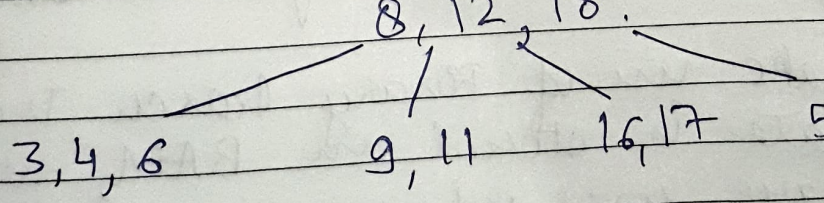
on adding 8, it gets splitted.



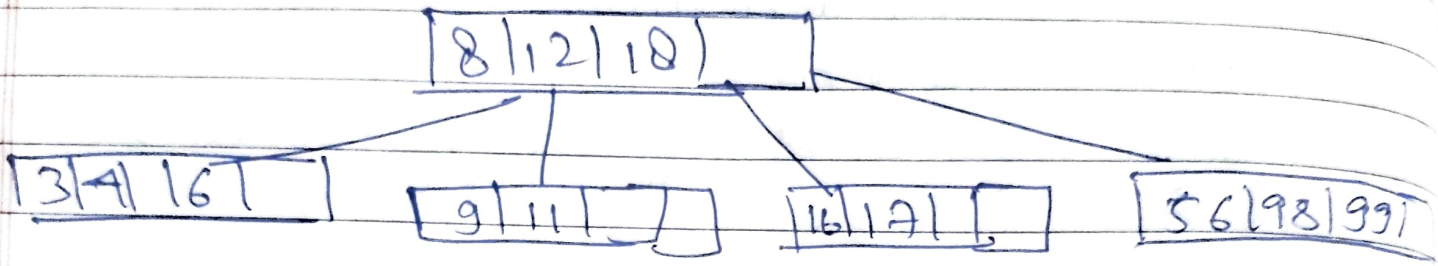
When we add 17, the right node of 8 gets splitted.



On adding 56 to the rightmost node, it gets splitted as well



The final Btree is

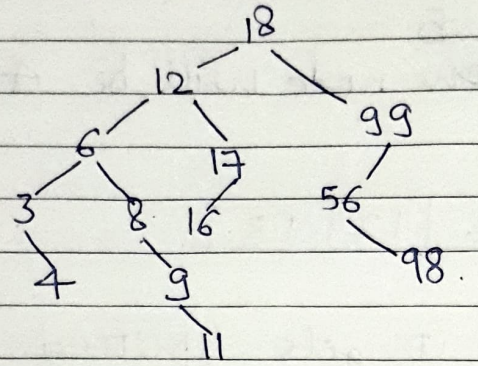


Height = 1
(If we consider root node is at level 1)

No. of nodes = 5

Degree = 4.

2b Inserting Elements: 18, 12, 6, 3, 8, 4, 9, 11, 17, 99, 56, 98.



In binary search tree, we insert values less than the root on its left and value greater than its root on its right. The left and right subtrees are also BST.

→ Height of BST here is 5 [Considering that the root node is at level 0].

→ The degree of a BST is 2
→ The number of nodes are 13

2→(c)(i) B-Trees give reduced height as compared to BST.

(ii) We make use of Binary Search Tree when data is stored in RAM while B-Trees are used when data is stored in the disk.

(iii) Btrees are an advantage over BSTs in terms of time and memory usage.