LINCOLN
UNIVERSITY COLLEGE

Texas

COLLEGE OF MANAGEMENT & INFORMATION TECHNOLOGY

BACHELOR IN INFORMATION TECHNOLOGY

Submitted by:

Submitted to:

Saroj Ghimire sir

Name: ISHWOR KHATIWAD

Year/ Semester: Third

Submission Date: 22 May 2021

Q.N.1Contrast and compare algorithm and pseudo-code with suitable examples.

Algorithm	Pseudo code
It is a step by step process to solve a problem.	The process of writing algorithm is Pseudo code.
It simplifies the problem and makes easier to understand.	Pseudo code is just a method of developing algorithm.
Example:calculate the area of rectangle: 1: input L 2: input B 3: Area=L*B 4:DisplayArea 5: End	To calculate the area of rectangle: Area of Rectangle () Begin Read: width,length; Set area=width*length; Print area; End
Algorithm is a simple method to develop program.	Pseudo code is complex as compared to algorithm.

For example: Algorithm to print numbers from 1 to 10:

1:Initialize variable no. As integer number (n=1).
2:Read and Store the value of number.
3:Repeat the step until number <10.
4: N=(n+1).
5. print the value of N.

EXAMPLE: Pseudo code to print 1 to 5 number:

```
Print one to five()
Begin
Set
N=1;
While
  N<=5
  Print:N;
  Set
  N=N+1;
End while
end
```

Q.N.2.

Compare iteration and recursion. Write program to implement factorial number using recursion.

Recursion -----

```
// method to find factorial of given number
int factorialUsingRecursion(int n)
{
    if (n == 0)
        return 1;

    // recursion call
    return n * factorialUsingRecursion(n - 1);
}

// ----- Iteration -----
// Method to find the factorial of a given number
int factorialUsingIteration(int n)
{
    int res = 1, i;

    // using iteration
    for (i = 2; i <= n; i++)
        res *= i;

    return res;
}
```

	Recursion	Iteration
Definition	Function calls itself.	A set of instructions repeatedly executed.
Application	For functions.	For loops.
Termination	Through base case, where there will be no function call.	When the termination condition for the iterator ceases to be satisfied.
Usage	Used when code size needs to be small, and time complexity is not an issue.	Used when time complexity needs to be balanced against an expanded code size.
Code Size	Smaller code size	Larger Code Size.

Q.N.3.

Show how to implement a stack using array (code is needed, just a sketch and pseudo code). What are the complexities of pop () and push () operations?

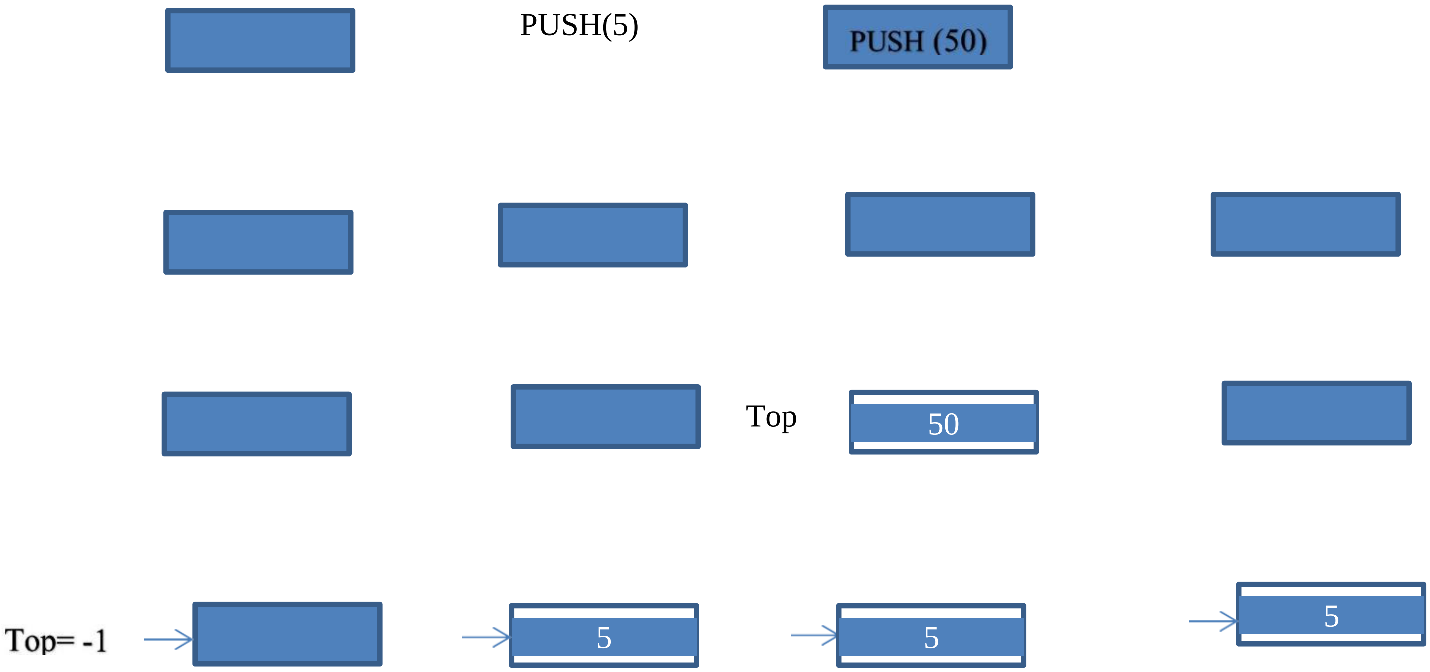
Note: A stack is a data structure with push (), pop (), is full () and is Empty () operations;

ANS:

- We initially have an empty stack. The top of an empty stack is set to -1.
- We push the element 5 into the stack. The top of the stack will points to the element 5.
- We push the element 50 into the stack. The top of the stack shifts and points to element 50.
- When we perform pop operation, removing the top element from the stack. The element 50 is pop from the stack. The top of the stacks now points to the elements 5

PUSH (5)

POP()



For Push-

```
if(Top == last index of Array)
{
Printf("data sent")
}Else
{Top = Top + 1
a[Top] = element you want to insert
```

}**For Pop-**

```
if(Top == -1)
{
Printf("data uploded")
}
Else
{
Temp = a[Top]
Top = Top -1
return Temp
}
```

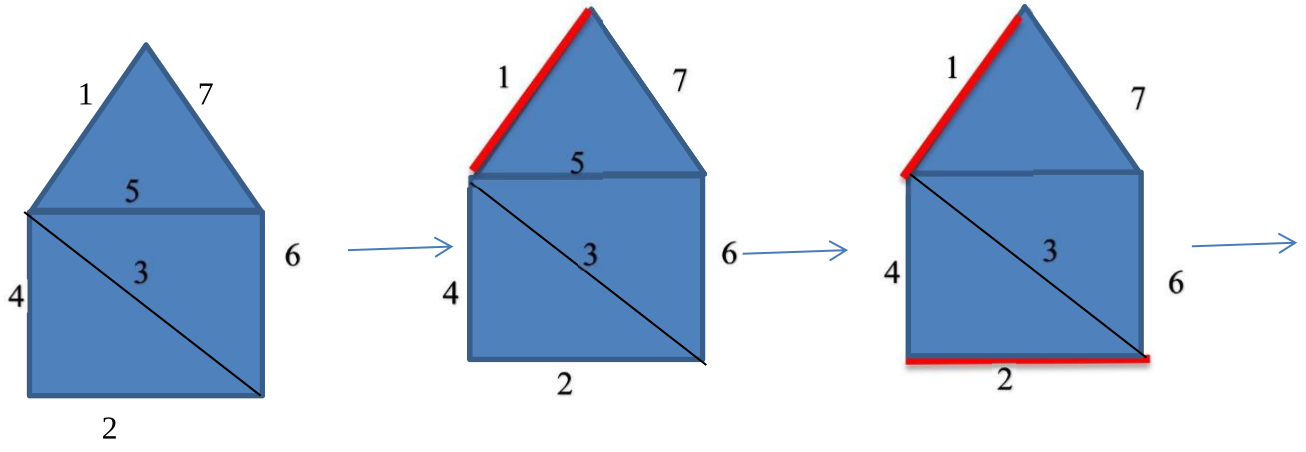
As you can observe, both Push and Pop need just a few simple array operation which take constant time.

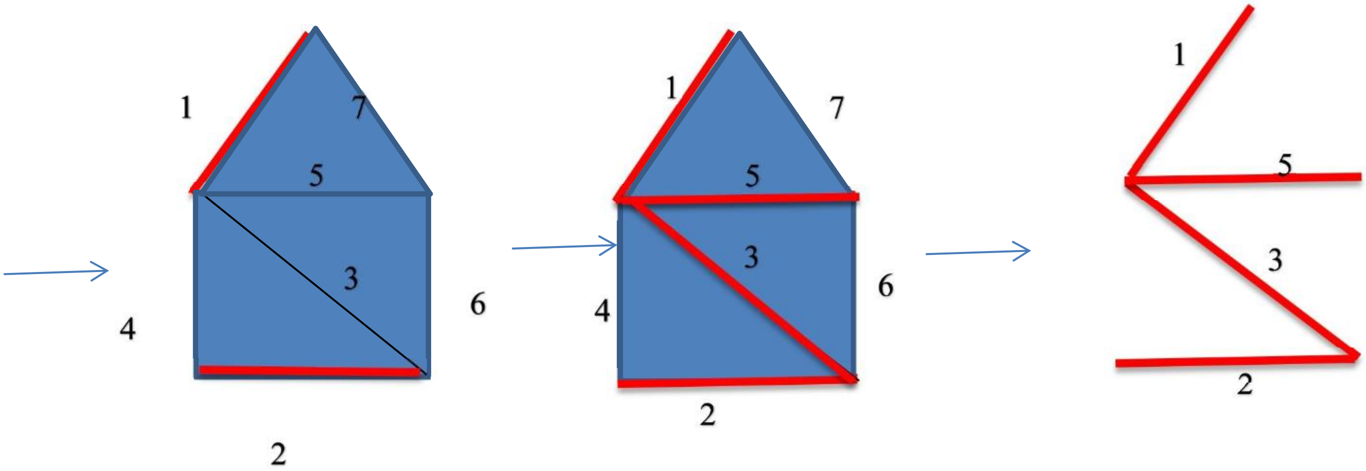
Q.N.4.
Write function or pseudo code for finding minimum spanning tree using Kruskal algorithm. Include example.

Algorithm for minimum spanning tree using Kruskal method:

- Sort the graph edges with respect to their weights.
- Start adding edges to the Minimum Spanning Tree (MST) from the edge with the smallest weight until the edge of largest weight.
- Only add edges which does not form a cycle, edges which connect only disconnected components.

For eg.





Minimum Spanning is: 1+2+3+5= 11

1. Infix Expression: ((AX + (B * CY)) / (D E)) ; Convert into postfix and prefix using algorithm. Write pseudo code.

Conversion Infix to Postfix: ((AX + (B * CY)) / (D E))

Move	Scan Operator	Operator Stack	Output Stack
(		(	-
(		((	-
A		((	A
X		((	AX
+		+((	AX

(	(+((	AX
B	(+((	AXB
*	*(+((	AXB
C	*(+((	AXBC
Y	*(+((	AXBCY
)	+((	AXBCY*
)	(	AXBCY*+
/	/(	AXBCY*+
(	(/(	AXBCY*+
D	(/(	AXBCY*+D
E	(/(	AXBCY*+DE
)	/(	AXBCY*+DE
)	-	AXBCY*+DE/

Conversion Infix to Prefix**:** ((AX + (B * CY)) / (D E))

=((E D) /((YC*B)+XA))

Move	Scan Operator	Operator Stack	Output Stack
1	(	(	-
2	(	((	-
3	E	((	E
4	D	((	ED
5	)	(	ED
6	/	/(	ED

7	(	(/(	ED
8	(	((/(	ED
9	Y	((/(	EDY
10	C	((/(	EDYC
11	*	*((/(	EDYC
12	B	*((/(	EDYCB
13	)	(/(	EDYCB*
14	+	+(/(	EDYCB*
15	X	+(/(	EDYCB*X
16	A	+(/(	EDYCB*XA
17	)	/(	EDYCB*XA+
18	)	-	EDYCB*XA+/
	(	(	-

EDYCB*XA+/=/+AX*BCYDE

Pseudo-code to convert infix to prefix:

```
1. infix = reverse(infix)
2. loop i = 0 to infix.length
3. if infix[i] is operand → prefix+= infix[i]
4. else if infix[i] is '(' → stack.push(infix[i])
5. else if infix[i] is ')' → pop and print the values of stack till the symbol ')' is not found
6. else if infix[i] is an operator(+, -, *, /, ^) then
```

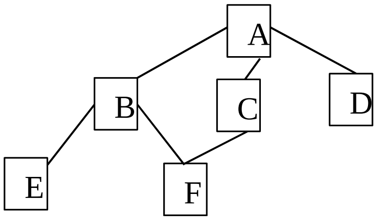
```
7. if the stack is empty then push infix[i] on the top of the stack.
8. Else →
9. If (infix[i] >stack.top)
10. → Push infix[i] on the top of the stack
11. else if(infix[i] == stack.top&& infix[i] == '^')
12. → Pop and print the top values of the stack till the condition is true
13. → Push infix[i] into the stack
14. else if(infix[i] == stack.top)
15. → Push infix[i] on to the stack
16. Else if(infix[i] < stack.top)
17. → Pop the stack values and print them till the stack is not empty and infix[i] < stack.top
18. → Push infix[i] on to the stack
19. End loop
20. Pop and print the remaining elements of the stack
21. Prefix = reverse(prefix)
```

Pseudo-code for infix to postfix:

```
1) loop i = 0 to infix.length
2) if infix[i] is operand → postfix+= infix[i]
3) else if infix[i] is '(' → stack.push(infix[i])
4) else if infix[i] is ')' → pop and print the values of stack till the symbol ')' is not found
5) else if infix[i] is an operator(+, -, *, /, ^) then
6) if the stack is empty then push infix[i] on the top of the stack.
```

```
7) Else →
8) If (infix[i] >stack.top)
9) → Push infix[i] on the top of the stack
10) else if(infix[i] == stack.top&& infix[i] == '^')
11) → Pop and print the top values of the stack till the condition is true
12) → Push infix[i] into the stack
13) else if(infix[i] == stack.top)
14) → Push infix[i] on to the stack
15) Else if(infix[i] < stack.top)
16) → Pop the stack values and print them till the stack is not empty and infix[i] < stack.top
17) → Push infix[i] on to the stack
18) End loop
```

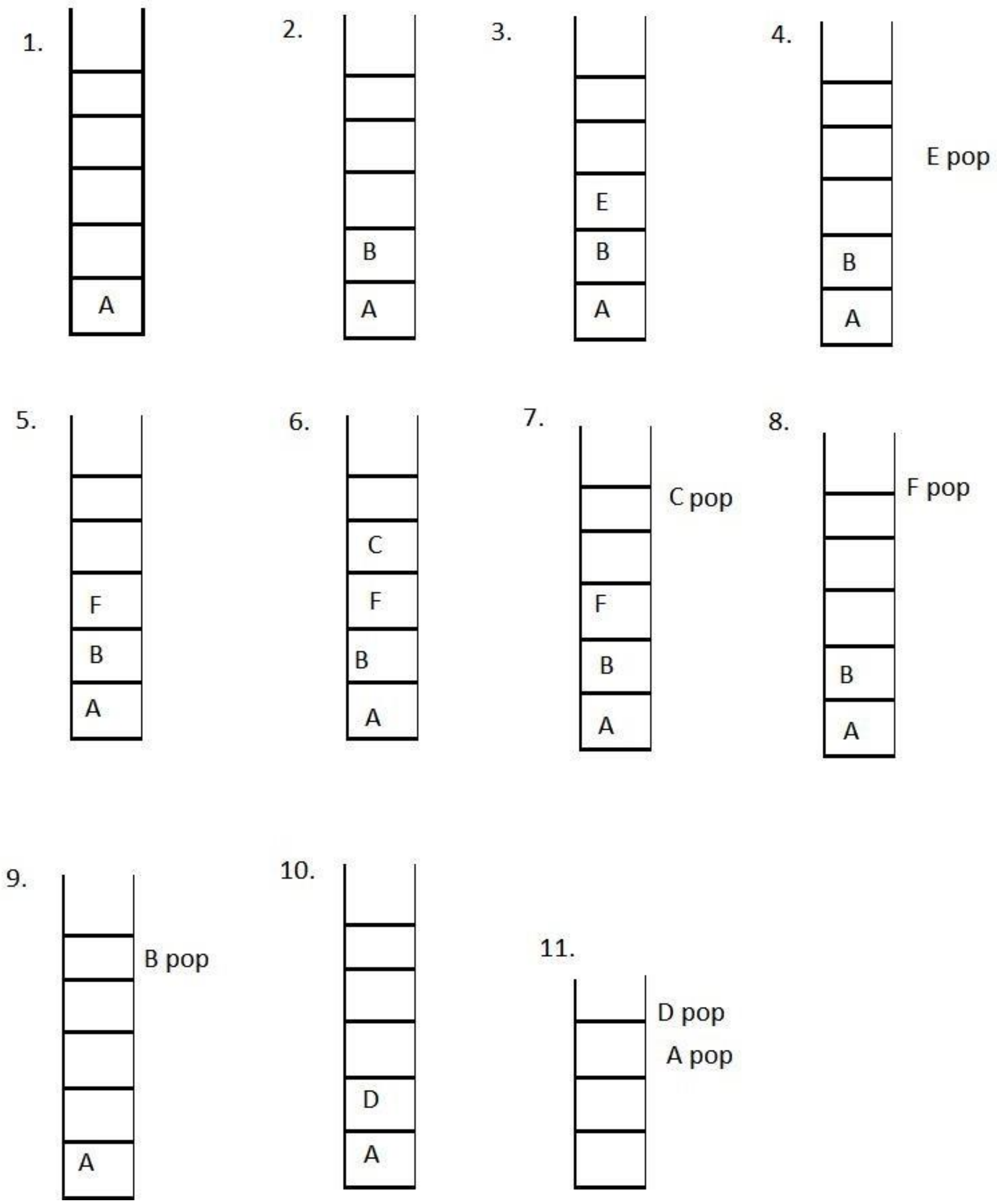
6. Apply DFS and BFS on below graph. Construct adjacency matrix.



Solution,

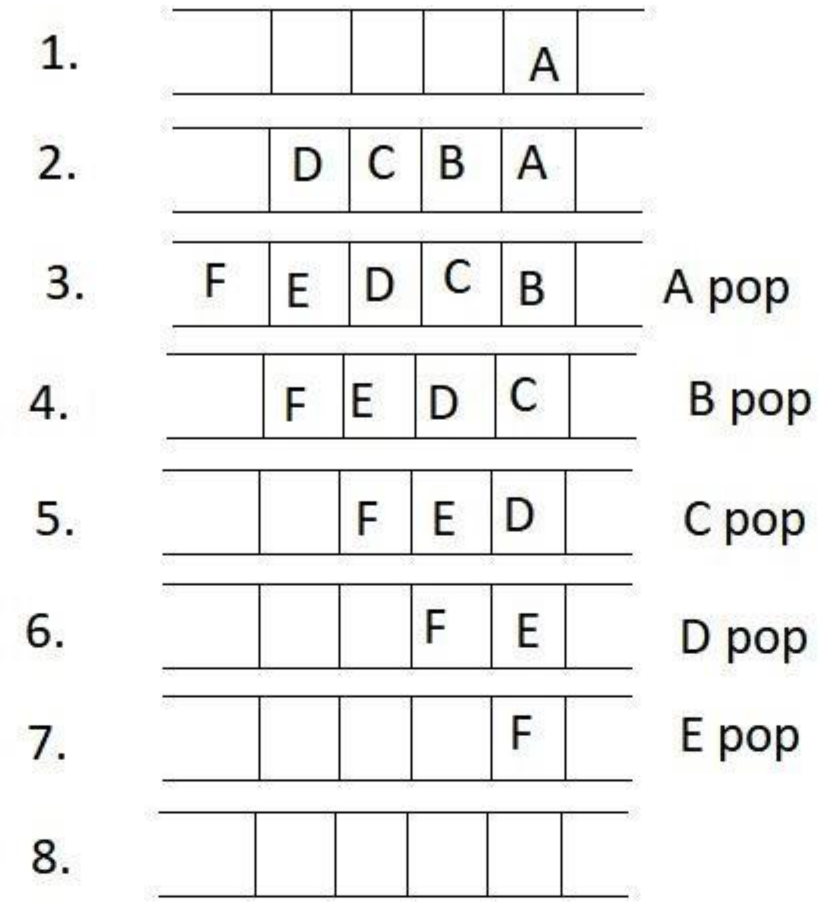
DFS

Output: ABEFCD



BFS

Output:ABCDEF

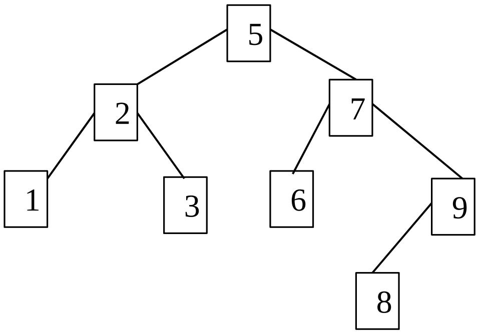


1.					A		
2.		D	C	B	A		
3.	F	E	D	C	B		A pop
4.		F	E	D	C		B pop
5.			F	E	D		C pop
6.				F	E		D pop
7.					F		E pop
8.							

Adjacency matrix is :

	A	B	C	D	E	F
A	0	1	1	1	0	0
B	1	0	0	0	1	1
C	1	0	0	0	0	1
D	1	0	0	0	0	0
E	0	1	0	0	0	0
F	0	1	1	0	0	0

7. Question refer to binary search tree below:

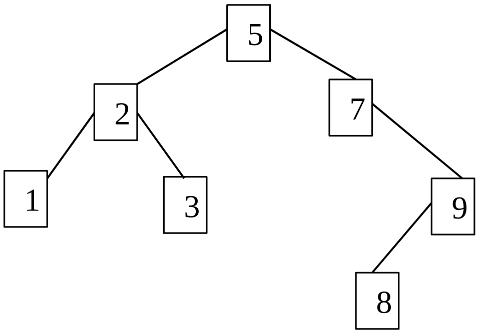


a. Draw the result of deleting 6 then 7.
b. Draw the result of deleting 7 then 6.

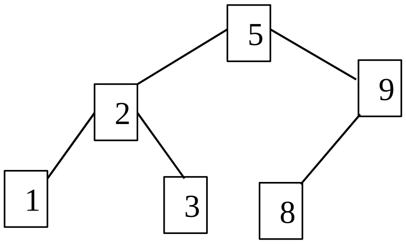
A)

The result of deleting 6 than 7::

After deleting 6:



After deleting 7:



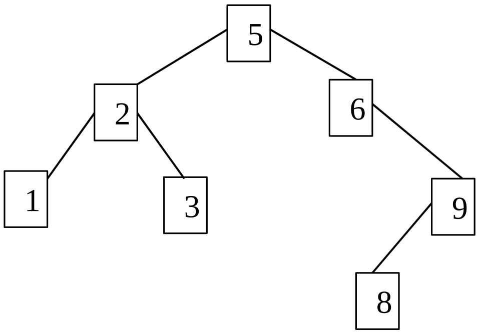
B)

The result of deleting 7 than 6:

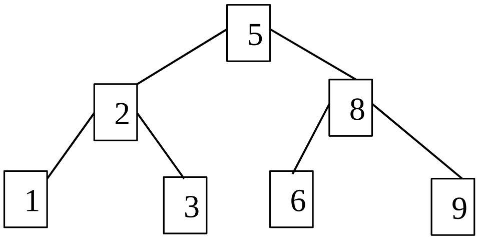
At first we delete 7,

There occur two condition i.e in order success or and in order predecessor.

From in order successor, we get



Form in order processor we get;



After deleting 7, we get;
Now we delete 6.

We get the same result from the in-order predecess or and in-order successor.