Sagar
Date:
Page:

Stack

Stacks are ordered collection of items into which items may be inserted or deleted from same end called TOP of element.

Primitive Operations

Insertion ✓ Push
Deletion ✓ Pop
Traverse ✗
Searching ✗
Sorting ✗

but in Stack we do

Application
1) Editor
2) Windows open in tabs
3) Page on browser tabs we going to other pag.
4) Recursive Func
5) writing for statement
6) Reversing a string
7) Binary Number equal
8) Parenthesis / Bracketed Expression

Dl stack size 10

TOP=-1

```
struct Stack
{ int TOP
  char Item[SIZE];
}
STACKSIZE (10) → Malloc.
    Struct stacks;
```

→ #define SIZE 10
#define TRUE 1
#define FALSE 0

```
ALGORITHM InitialiseStack(STACK S)
BEGIN:
        S.TOP=-1                    O(1)    O(1)
END;

ALGORITHM Empty(STACKS)
BEGIN:
        IF S.TOP==-1 THEN
            RETURN TRUE             O(1)    O(1)
        ELSE
            RETURN FALSE
END;
```

Scanned with CamScanner

```
ALGORITHM PUSH (STACK S, R)
BEGIN:
        IF S.TOP == STACKSIZE-1 THEN
            WRITE("STACK OVERFLOWS")
            EXIT(1)                          O(1)
        S.TOP = S.TOP+1
        S.Item[S.TOP] = R                    O(1)
END;

ALGORITHM POP (STACKS)
BEGIN:
        IF(Empty (S)) THEN
            WRITE ("Stack underflows")
            EXIT(1)                          O(1)
        x = S.Item[S.TOP]
        S.TOP = S.TOP-1                      O(1)
        RETURN x
END;

ALGORITHM StackTop (STACK S)
BEGIN:
        x = S.Item[S.TOP]                    O(1)
        RETURN x
END;                                         O(1)
```

Scanned with CamScanner

2	9	
2	4	1
2	2	0
2	1	0
	0	1

ALGORITHM Decimal To Binary(Decimal)

BEGIN: STACK S
Initialise Stack(S)

WRITE Decimal != 0 DO

r = Decimal % 2

Decimal = Decimal / 2

PUSH (S, r)

WHILE
WRITE ! Empty (S) DO

x = POP(S)

WRRITE (x)

END;

ALGORITHM Reverse (Str [ ])

BEGIN:

~~STACK S~~
~~i=0~~
~~WHILE Str~~

STRING Newstring[ ]
STACK S
Initialize Stack (S)
i=0
WHILE Str[i] != '\0' DO
PUSH (S, Str[i])
i++
i=0;
WHILE ! Empty (S) DO
x = Pop(S)
Write(x)
~~WRITE(x)~~ ~~New~~ String [i] = x
i++;
~~New~~ String[i] = '\0'

END;

Sagar
Date:
Page:

An attemp to insert an element in full Data structure leads to condition of overflow

An attemp to delete an element ~~in~~ from empty Data structure leads to condition of underflow

~~String~~ Reverse (str[ ])
BEGIN:

```
STACK S                                   Time     Space
Initialize Stack (S)                  -- O(1)      O(1)
i=0
WHILE str[i] != '\0' DO
        PUSH (S, str[i])              - O(1)       O(1)
        i++
i=0
WHILE !Empty (S) DO                   - O(1)       O(1)
        x = POP(S)                    - O(1)       O(1)
        str[i] = x
        i++
str[i] = '\0'
```

END;

i.e, (65+x-10)

ALGORITHM Pallendrome (str[ ])
BEGIN;

```
STACK S
Initialize Stack(S)
i=0
WHILE str[i] != '\0' DO
    PUSH(S, str[i])
    i++
i=0
WHILE(!empty( ) !
~~WHILE str[i] != '\0' DO~~
IF(str[i] == STACKTOP(S))
        POP(S)
        i++
ELSE
    BREAK
```

Scanned with CamScanner

Sagar
Date:
Page:

```
#include <stdio.h>
Struct student
{
   int roll no;
   char name[20];
   Char OdmNo[12];
};

Void main()
{
   Struct student S={ 98203, "Satwik", "2018CS092"};
   printf("%d", S.roll no);
   printf("\n%s", S.name);
   printf("\n%s", S.admNo);
}
```

```
IF Empty(S) THEN
      WRITE("String is Palindrome")
ELSE
      WRITE("String is not a Palindrome
END;
```

Scanned with CamScanner

```
# include <std lib.h>
# include<stdio.h>
# define SIZE 10
# define TRUE 1
# define FALSE 0

Struct Stack
{
  int TOP;
  int Item [SIZE];
};

Struct stack S;
```

```
# include <stdio.h>
~~main~~
ALGORITHM BRACKETS (str[])
BEGIN:
        STACK S
        Initialize stack(S)
        i=0
        WHILE str[i] != '\0' DO
        IF(str[i] == '(')
            PUSH(S, str[i])
            i++
j=0     WHILE str[i] != '\0'
        ~~WHILE str[i] != '\0' DO~~
        WHILE
        IF(str[i] == ')')
            j++
            x = POP(~~str[i]~~)
        ELSE
            underflow
```

```
STACK S
Initialize Stack(S)
i=0
WHILE str[i] != '\0' DO
  IF(str[i] == '(')
      PUSH(S, str[i])
      i++
  IF(j <= i)
  ~~WHILE str[i]~~
  IF(str[j] == ')')
      j++;
      x = POP(~~str[i]~~S)
IF(str[0] == ')')
UNDERFLOW
```

Scanned with CamScanner

Sagar

Date:
Page:

4+3*7-8↑2+6/2-3↑2

~~4+2-8~~

~~4+(3)(7)*-8(2)↑~~

~~4+21~~

= 4+3*7-64+6/2-9

= 4+21-64+3-9

= 28-73

= -45

9
4

Q 378*5+821-53*+-

*	+	↑ 2	-	* 3
8	5	8	64	5
7	56	61	61	-3
3	3	3	3	3

*
5
840
47
3

+	-	=	
15	12		-9
-3	3		
3			

	↑		*	
	2	-	3	+
8 5 8	64	5		15
856	61	-3		12
3				

*5	2
856	61
7	
3	

= -9

61
×61
166
366
3726

Sagar
Date:
Page:

Symbol	Operand Stack	Opr-1	Opr-2	Value
3	3			
7	3,7			
8	3,7,8			
*	3,56	7	8	56
5	3,56,5			
+	3,61	56	5	61
8	3,61,8			
2	3,61,8,2			
↑	~~3,61,8,2~~ 3,61,64	8	2	64
-	3,-3	61	64	-3
5	3,-3,5			
3	3,-3,5,3			
*	3,-3,15	5	3	15
+	3,12	-3	15	12
-	-9	3	12	-9
~~AS~~				

6924-+*3/

Symbol	Operand Stack	Opr-1	Opr-2	Value
6	6			
9	6,9			
2	6,9,2			
4	6,9,2,4			
-	6,9,-2	2	4	-2
+	6,7	9	-2	7
*	42	6	7	42
3	42,3			
/	14	42	3	14
5				

Scanned with CamScanner

Sagar
Date:
Page:

ALGORITHM Post fix Evaluation (Postfix Expression)

BEGIN
STACK Operand Stack
Initialize (Operand Stack)
WHILE not end of symbols from Postfix Exp DO
Symbol = Next Symbol from Postfix Expression
IF symbol is an Operand THEN
Push (Operand stack, symbol)
ELSE
Opd 2 = Pop (Operand stack)
Opd 1 = POP (Operand Stack)
Value = Result of applying symbol to Opd1 Opd2
Push (operand stack, value)

X = POP (Operand Stack)
RETURN X
END;

Operand identify
Operator identify
Prcd func
Operator stack
Postfix Exp

Scanned with CamScanner

27725

Date:
Page:

8695*+3/+2-

	Operand Stack	Opr 1	Opr 2	Value
8	8			
6	8,6			
9	8,6,9			
5	8,6,9,5			
*	8,6,45	9	5	45
+	8,51	6	45	51
3	8,51,3			
/	8,17	51	3	17
+	25	8	17	25
2	25,2			
-	23	25	2	23

Prefix

+*34-84

Reverse the string 48-43*+$

Symbol	Operand Stack	Opd 1	Opd-2	Value
4	4			
8	4,8			
-	4	8	4	4
4	4,4			
3	4,4,3			
*	4,12	3	4	12
+	16	12	4	16
$	16			

Scanned with CamScanner

PREFIX NOTATION - POLISH NOTATION

Sagar

EXPRESSION

$-*\uparrow 2\,4\,3+8\,4\,2$

$2\,4\,8-+\,3\,4\,2\uparrow *-$

Symbol	Operand stack	Opd 1	Opd 2	Value
2	2			
4	2,4			
8	2,4,8			
-	2,4	8	4	4
+	6	4	2	6
3	6,3			
4	6,3,4			
2	6,3,4,2			
↑	6,3,16	2	4	16
*	6,48	16	3	48
-	42	48	-6	42
$	42			

ALGORITHM Prefix Evaluation (Prefix Expression)

BEGIN:

STACK Operand Stack

Initialize (Operand stack)

~~STACK Stack~~ Reverse (Prefix Expression)

WHILE not end of Input from ~~Prefix~~ Postfix DO

Symbol = Next symbol from ~~Prefix~~ Postfix Expression

IF symbol is an operand THEN

Push (operand stack, symbol)

ELSE

Opd 1 = POP (opd stack)

Opd 2 = Pop (Operand stack)

Value = Result of applying symbol to Opd 1 & Opd 2

Push (operand stack, value)

X = POP (operand stack)

RETURN X

END

Scanned with CamScanner

POSTFIX NOTIFICATION -
REVERSE POLISH NOTATION EXPRESSION

Sagar
Date:
Page:

1)	( )
2.)	↑
3.)	* / %
4.)	+ -

Right Associative
} Left Associative

Prcd (op1, op2) True
If opd1 > or = Opd2
Then True

Prcd (↑, -) True
Prcd (-, +) True
Prcd (/, -) True
Prcd (+, +) True

Prcd = Prescidence of operator op 1 is more than op2 then TRUE
" op1 = OP 2 THEN TRUE if Left Associative
ELSE FALSE
but OP1 = OP2 but if Right Associative
False

Scanned with CamScanner

Sagar

Date:
Page:

Solve postfix Exp

ABCDE↑↑*+FG/-

A=29
B=14
C=3
D=2
E=2
F=36
G=3

Symbol	Operand Stack	Opd1	Opd2	Value
29	29			
14	29,14			
3	29,14,3			
2	29,14,3,2			
2	29,14,3,2,2			
↑	29,14,3,4	2	2	4
↑	29,14,81	3	4	81
*	29,1134	14	81	1134
+	1163	29	1134	1163
36	1163,36			
3	1163,36,3			
/	1163,12	36	3	12
-	1151	1163	12	1151
$				

$$\begin{array}{r} 81 \\ \times\ 14 \\ \hline 324 \\ 81\times \\ \hline 1134 \end{array}$$

Scanned with CamScanner

Infix → Postfix

Q A*B+C↑D↑E

Symbol	Postfix Exp	Operator Stack	Procd
A	A		
*		*	
B	AB	*	→ * Pop
+	AB*	+	(*, +) True
C	AB*C	+	
↑		+, ↑	(+, ↑) False → Then Push
D	AB*CD		
↑		+, ↑, ↑	(↑, ↑) False
E	AB*CDE	+ ↑ ↑	
$	AB*CDE↑↑+		

Q A↑B↑C * D/E+F

Symbol	Postfix Ex	Operator stack	Procd
A	A		
↑		↑	
B	AB	↑	
↑		↑ ↑	(↑, ↑) = F → Push
C	ABC		
*	ABC↑↑	*	→ Pop (↑, *) = T → Pop (↑, *) = T
D	ABC↑↑D	*	
/	ABC↑↑D*	/	→ Pop *, / = T
E	ABC↑↑D*E		
+	ABC↑↑D*E/	+	→ Pop /, + = T
F	ABC↑↑D*E/F	+	
$	ABC↑↑D*E/F+		

Scanned with CamScanner

Q A*B*C↑D↑E%F+G%H

Symbol	Postfix Exp	Operand Stack	Prcd
A	A		
*	A	*	
B	AB	*	
*	AB*	*	(*,*) = T
C	AB*C	*	
↑	AB*C	*↑	(*,↑) = F
D	AB*CD	*↑	
↑	AB*CD	*↑↑	(↑,↑) = F
E	AB*CDE	*↑↑	
%	AB*CDE↑↑*	%	(↑,%) = T, (↑,%) = T, (*
F	AB*CDE↑↑*F	%	
+	AB*CDE↑↑*F%	+	(%,+) = T
G	AB*CDE↑↑*F%G	+	
%	AB*CDE↑↑*F%G	+%	(+,%) = F
H	AB*CDE↑↑*F%GH%+		
$			

Scanned with CamScanner

A - B ↑ C + E ↑ F / G ↑ F % G

Symbol	Postfix Exp	Operand stack	Proc
A	A		
-	A	-	
B	AB	-	
↑	AB	-↑	(-↑) = F
C	ABC	-↑	
+	ABC↑-	+	(↑+) = T (-+) = T
E	ABC↑-E	+	
↑	ABC↑-E	+↑	+↑ = F
F	ABC↑-EF	+↑	
/	ABC↑-EF↑	+/	↑,/ = T +,/ = F
G	ABC↑-EF↑G	+/	
↑	ABC↑-EF↑G	+/↑	/↑ = F
F	ABC↑-EF↑GF	+/↑	
%	ABC↑-EF↑GF↑/	+%	(↑%) = T, (/%) = T, (+%)
G	ABC↑-EF↑GF↑/G	+%	
$	ABC↑-EF↑GF↑/G%+		

If 2 operator in presedence table lead to True then pop the first operand until stack is empty or condition True

Else ~~[illegible]~~ False.

Scanned with CamScanner

```
Valid = TRUE
Str[]
i=0
WHILE Str[i] != '\0' DO
    IF Str[i] == '(' THEN
        PUSH(S, Str[i])
    ELSE
        IF Str[i] == ')' THEN
            IF ! Empty(S) THEN
                Pop(S)
            ELSE
                Valid = False
                BREAK
    i++

IF Valid == False THEN
    WRITE("Invalid Exp")
ELSE
    IF Empty(S) THEN
        WRITE("Valid Exp");
    ELSE
        WRITE("Invalid Exp");
```

Scanned with CamScanner

Convert to postfix

b^2-4ac

b*b -4*a*c

Symbol	Postfix Exp	Oper and Stack	Prcd
b	b		
*	b	*	
b	bb	*	
-	b b*	-	(*-) T
4	b b*4	-	
*	bb*4	-*	(-*) F
a	bb*4a	-*	
*	b b*4a*	-*	(* *)=T (-*)=F
c	b b*4a*c	-*	
$	bb*4a*c*-		

b	b		
↑	b	↑	
2	b2	↑	
-	b2↑	-	(↑-) T
4	b2↑4	-	
*	b2↑4	-*	(-*) F
a	b2↑4a	-*	
*	b2↑4a*	-*	** T (-*)F
c	b2↑4a*c		
$	b2↑4a*c*-		

Scanned with CamScanner

Q x^2+y^2-2xy

x↑2+y↑2-2*x*y

x	x		
↑	x	↑	
2	x2	↑	
+	x2↑	+	(↑+)=F
y	x2↑y	+	
↑	x2↑y	+↑	(+↑)F
2	x2↑y2	+↑	
-	x2↑y2↑+	-	(↑-)T, (+-)T
2	x2↑y2↑+2	-	
*	x2↑y2↑+2	-*	(-*)F
x	x2↑y2↑+2x	-*	
*	x2↑y2↑+2x*	-*	(**)T (-*)F
y	x2↑y2↑+2x*y*		

Scanned with CamScanner

```
ALGORITHM Infix To Postfix (Infix Expression)
BEGIN:
        STACK operator stack
        Initialize(operator stack)
        WHILE not end of symbols from Infix Expression Do
                Symbol = Next Symbol from Infix Expression
                IF Symbol is an operand THEN
                        Add symbol to Postfix Expression
                ELSE
                        WHILE !Empty(operator Stack) AND
                            prcd (StackTop(operator Stack, symbol) Do
                                x = POP(operator Stack)
                                Add x to postfix Expression
                        PUSH(operator Stack, symbol)
        WHILE ! Empty (operator Stack) Do
                x = POP(operator stack)
                Add x to Postfix Expression
        RETURN Postfix Expression
END;
```

Prcd ((,)) = False
Prcd (op, ( ) = False
Prcd (op,)) = True
Prcd ((,)) = False special case

Scanned with CamScanner

(		(	
a	a		
↑	a	(↑	((↑)=F
(	a	(↑(	(↑(=F
b	ab	(↑(	
*	ab	(↑(*	((*)=F
c	abc	(↑(*	
+	abc*	(↑(+	(*+)=T, ((+)=F
d	abc*d	(↑(+	
)	abc*d+	(↑	+)=T ()=F
)	abc*d↑		↑)=T ()=F
$	abc*d+↑		

(1) $(b^2-4ac)/2a$

(b↑2 -4*a*c)/(2*a)

(		(	
b	b		
↑	b	(↑	(↑=F
2	b2	(↑	
-	b2↑	(-	(↑-)=T
4	b2↑4	(-	
*	b2↑4	(-*	(-*)=F
a	b2↑4a	(-*	
*	b2↑4a*	(-*	(*,*)T (-*)F
c	b2↑4a*c	(-*	
)	b2↑4a*c*-		(*)T (-)=T ()=F
/	b2↑4a*c*-	/	
(	b2↑4a*c*-	/(	(/,()=F
2	b2↑4a*c*-2	/(	
*	b2↑4a*c*-2	/(*	(*)=F
a	b2↑4a*c*-2a	/(*	
)	b2↑4a*c*-2a*	/	*)=T *()=F

$ | b2↑4a*c*-2a*

Q $(A^2-B^2)/(C^3+D^3/E^2)$

(A↑2 - B↑2)/(C↑3+D↑3%E↑2)

(		(	
A	A		
↑	A	(↑	((↑)=F
2	A2	(↑	
-	A2↑	(-	(↑-)T ((-)=F
B	A2↑B	(-	
↑	A2↑B	(-↑	(-↑)=F
2	A2↑B2		
)	A2↑B2↑-		(↑))=T (-))T ((-)=F)
/	A2↑B2↑-	/	
C	A2↑B2↑-	/(	(/(C)=F
↑C	A2↑B2↑-C		
↑	A2↑B2↑-C	/(↑	(↑)=F
3	A2↑B2↑-C3	/(↑	
+	A2↑B2↑-C3↑	/(+	(↑+)=T ((+)=F
D	A2↑B2↑-C3↑D	/(+	
↑	A2↑B2↑-C3↑D	/(+↑	(+↑)=F
3	A2↑B2↑-C3↑D3	/(+↑	
%	A2↑B2↑-C3↑D3↑	/(+%	(↑%)=T
E	A2↑B2↑-C3↑D3↑E	/(+%	
↑	A2↑B2↑-C3↑D3↑E	/(+%↑	(%↑)=F
2	A2↑B2↑-C3↑D3↑E2	/(+%↑	
)	A2↑B2↑-C3↑D3↑E2↑%+	/	(↑)=T (%↑)=T (+))T ()=F
$	A2↑B2↑-C3↑D3↑E2↑%+/		

Scanned with CamScanner

```
Prod (x, y)

IF x == '↑' OR x == '*' OR x == '/' OR x == '%' THEN
    IF y == '↑' THEN
        RETURN FALSE
    ELSE
        RETURN TRUE
ELSE
    IF x == '+' OR x == '-' THEN
        IF y == '+' OR '-' THEN
            RETURN TRUE
        ELSE
            RETURN FALSE
END;
```

Scanned with CamScanner

A+B-C

C-B+A

Symbol	Prefix	Opstack	Prcd
C	C		
-	C	-	
B	CB	-	
+	CB	- +	(+,-) = T ⇒ Push Here reve
A	CBA		
$	CBA+-		
	-+ABC		
	ans		

A↑B*C-D↑E/F

F/E↑D-C*B↑A

Symbol	Prefix	OpStack	Prcd
F	F		
/	F	/	
E	FE	/	
↑	FE	/↑	(↑/) = T → Push
D	FED	/↑	
-	FED↑/	-	(-↑) = F → POP, (-/) = F
C	FED↑/C		
*	FED↑/C	-*	(*-) T
B	FED↑/CB	-*	
↑	FED↑/CB↑	-*↑	(↑*) T
A	FED↑/CBA	-*↑	
$	FED↑/CBA↑*-		

-*↑ABC/↑DEF

Scanned with CamScanner

$A^2 - B^2 + 4CD \% E$

A↑2-B↑2+4*C*D %.E

E %.D*C*4+2↑B-2↑A

Symbol	Prefix	Op Stack	Prcd
E	E		
%	E	%	
D	ED		
*	ED	%*	(*%)=T
C	EDC		
*	EDC	%**	(**)=T
4	EDC4	%**	
+	EDC4** %	+	(+*)=F (+*)=F +(*)=
2	EDC4**%2		
↑	EDC4**%2	+↑	(↑+)T
B	EDC4**%2B		
−	EDC4**%2B↑	+−	(−↑)=F (−*)=T
2	EDC4**%2B↑2	+−	
↑	EDC4**%2B↑2	+−↑	(↑−)=T
A	EDC4**%2B↑2A		
$			
	EDC4**%2B↑2A↑−+		

+−↑A2↑B2↑%**4CDE

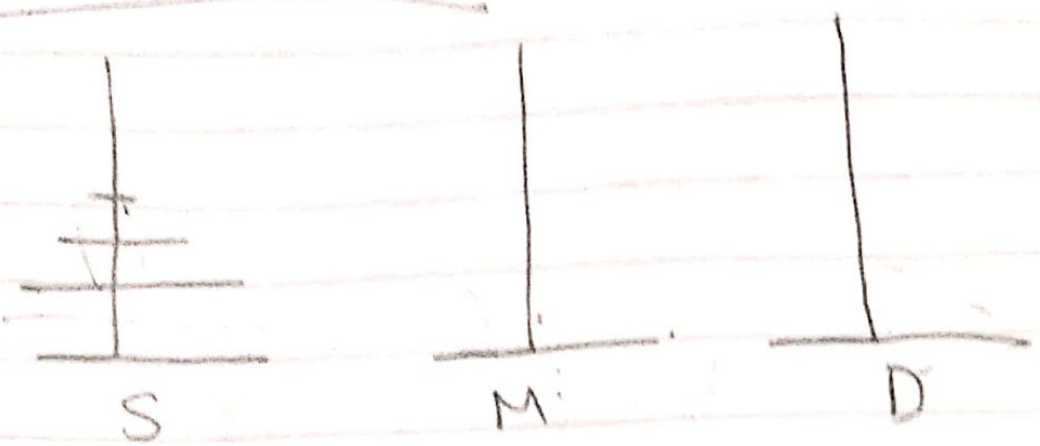
Scanned with CamScanner

Sagar
Date:
Page:

```
ALGORITHM Infix To Prefix (Infix Expression)
BEGIN :
        STACK Operator Stack
        Initialize (Operator stack)
                Reverse (Infix Expression)
        WHILE not end of symbols from Infix Expression Do
            Symbol = Next Symbol from Infix Expression
            If symbol is an operand THEN
                    Add symbol to Prefix Expression
            ELSE
                    WHILE ! Empty (Operator Stack) AND
                      ! pred (StackTop (operatorStack, Symbol
                            x = Pop (operator Stack)
                            Add x to prefix Expression
                    PUSH (operator stack, symbol)
            WHILE ! Empty (Operator Stack) DO
                    x = Pop (operatorStack)
                    Add x to prefix Expression
            Reverse (prefix Expression)
            RETURN Prefix Expression
END
```

Scanned with CamScanner

Tower Of Hanoi



$S \to D$
$S - M$
$D \to M$

$7 = 2^3 - 1$

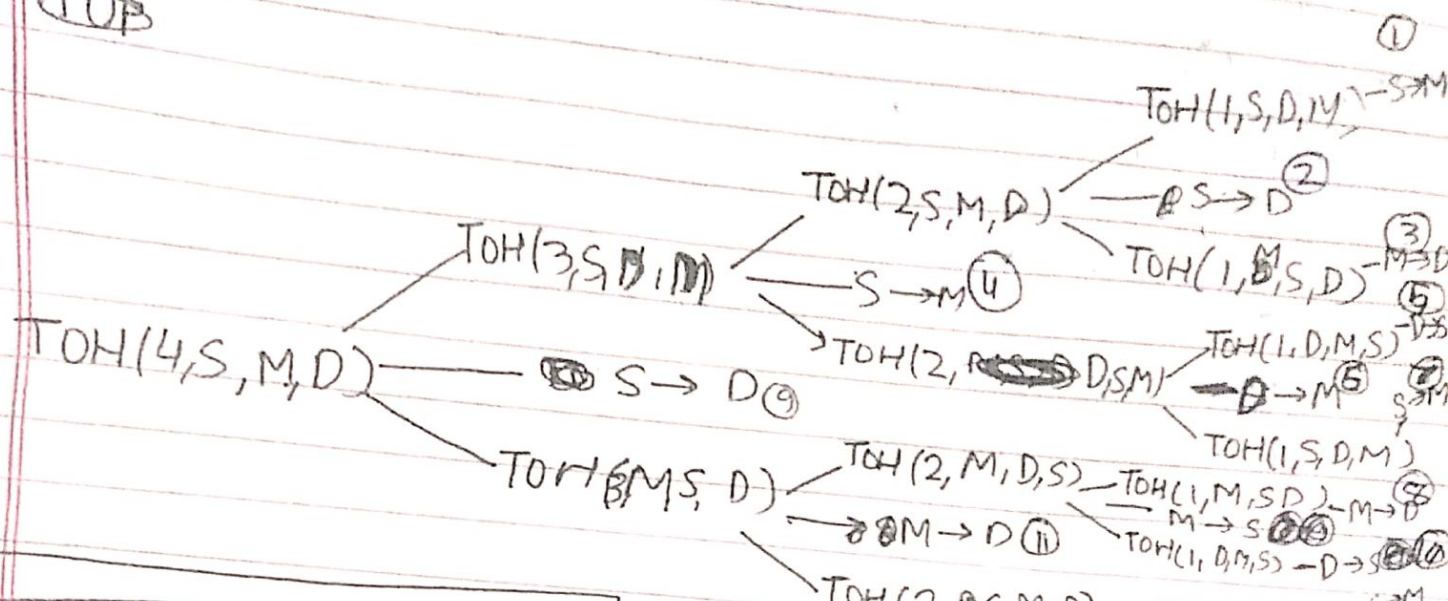
$S \to D$

$M \to S$
$M \to D$
$S \to D$

```
ALGORITHM Towers Of Hanoi (n,S,M,D)
BEGIN:
        IF n==1 THEN
         Transfer Disk from Source to Destination
        ELSE
            Tower Of Hanoi (n-1,S,D,M)
            Transfer 1 disk from Source to Destination
            Tower Of Hanoi (n-1,M,S,D)
END;
```

Scanned with CamScanner

TOH



1	1	S→M
2	2	S→D
3	3	M→D
4	4	S→M
5	5	D→S
6	6	D→M
7	7	S→M
8	8	S→D
9	9	M→D
10	9	M→S
11	10	D→S
12	11	M→D
13	12	S→M
14	13	S→D
15	14	M→D

Scanned with CamScanner