CS & IT ENGINEERING

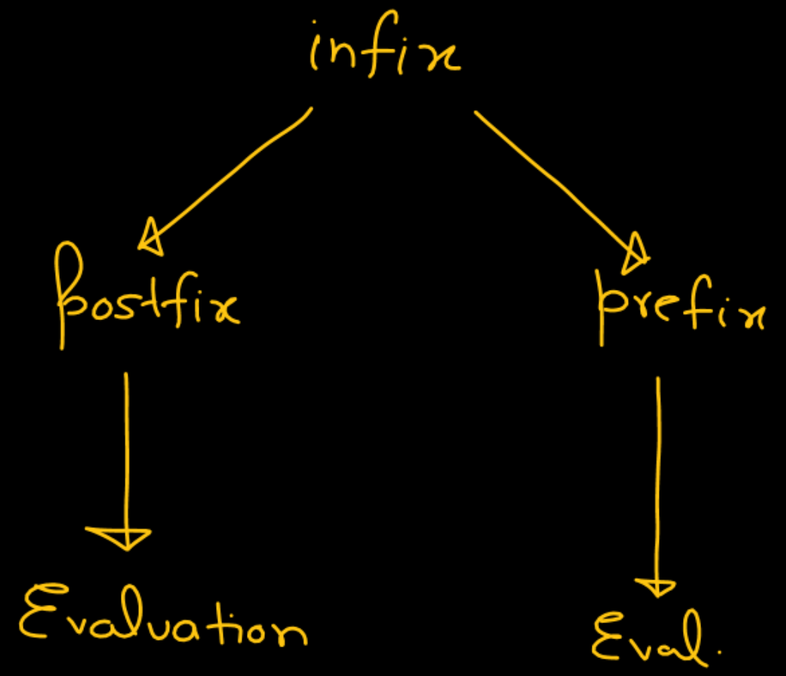
Data Structure & Programming

Stack and Queues

Lec- 03

By- Pankaj Sharma sir

Stack-III


infix
postfix
prefix
Evaluation
Eval.

Postfix Evaluation

infix : 2 + 3 × 5

Postfix : 2 3 5 × +

using stack

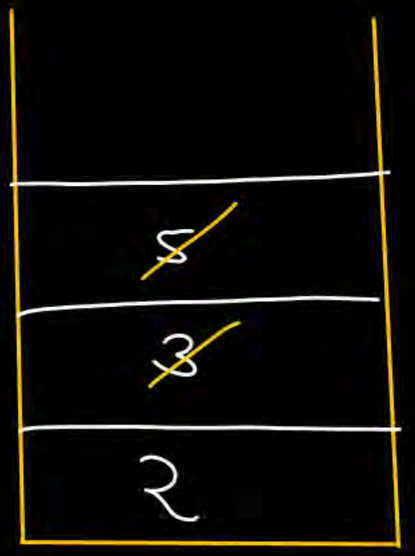
operator ⇒ (*)

a) Pop 1st elem ⇒ A

b) Pop 2nd elem ⇒ B

B (+) A

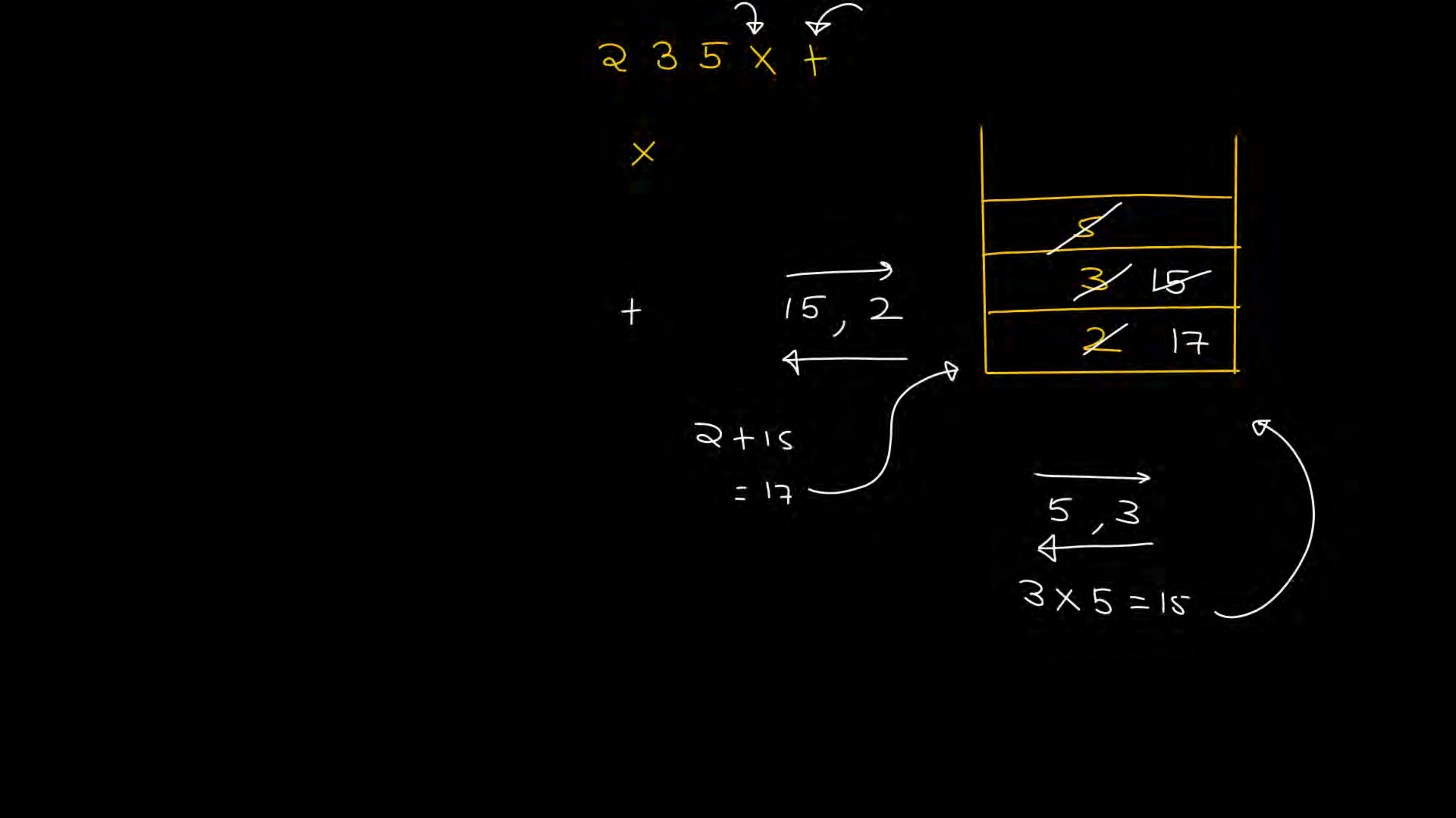
push result onto stack



5 3

3 × 5

= 15

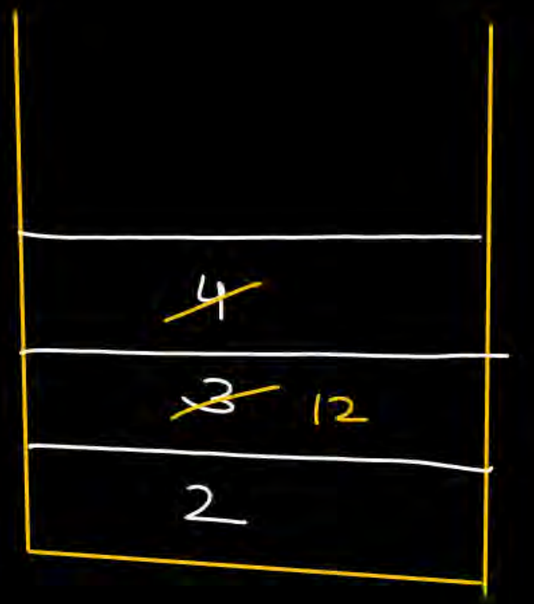

2 3 5 X +
X
+
15 , 2
2 + 15
= 17
5
3 15
2 17
5 , 3
3 X 5 = 15

Ex 2:

infix : 2 + 3 x 4 - 6/2

Postfix : 2 3 4 x + 6 2 / -

(i) x



4, 3

3 x 4

push 12

Ex 2:

infix : 2 + 3 x 4 - 6/2

Postfix : 2 3 4 x + 6 2 / -

(i) x

4, 3

3 x 4

push 12

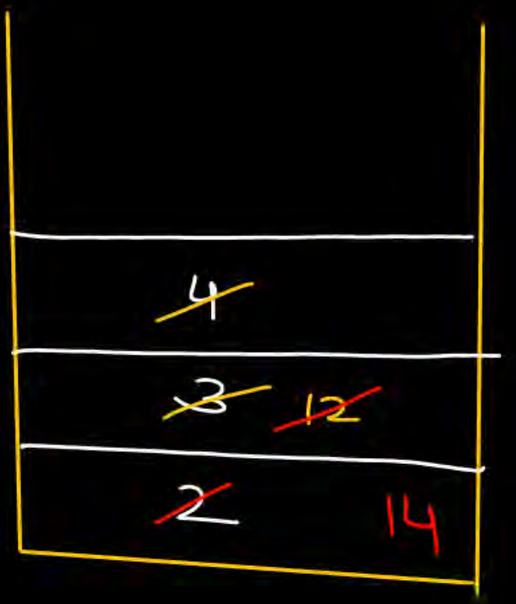
(ii) +

12, 2

2 + 12

14

push 14



Ex 2:

infix : 2 + 3 x 4 - 6/2

Postfix : 2 3 4 x + 6 2 / -

(i) x

4, 3

3 x 4

push 12

(ii) +

12, 2

2 + 12

14

push 14

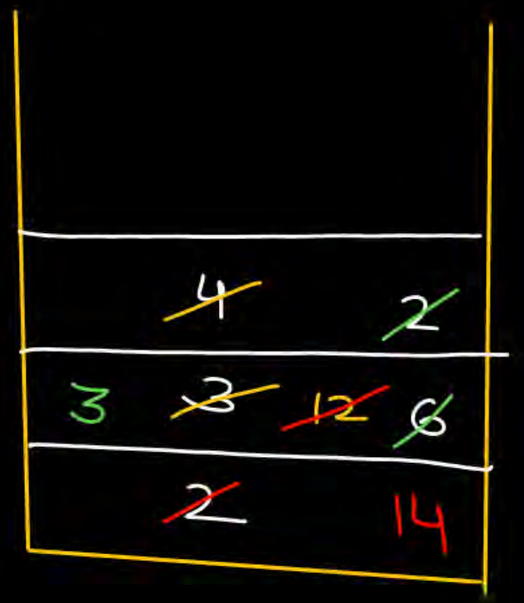
(iii) /

2, 6

6/2

= 3

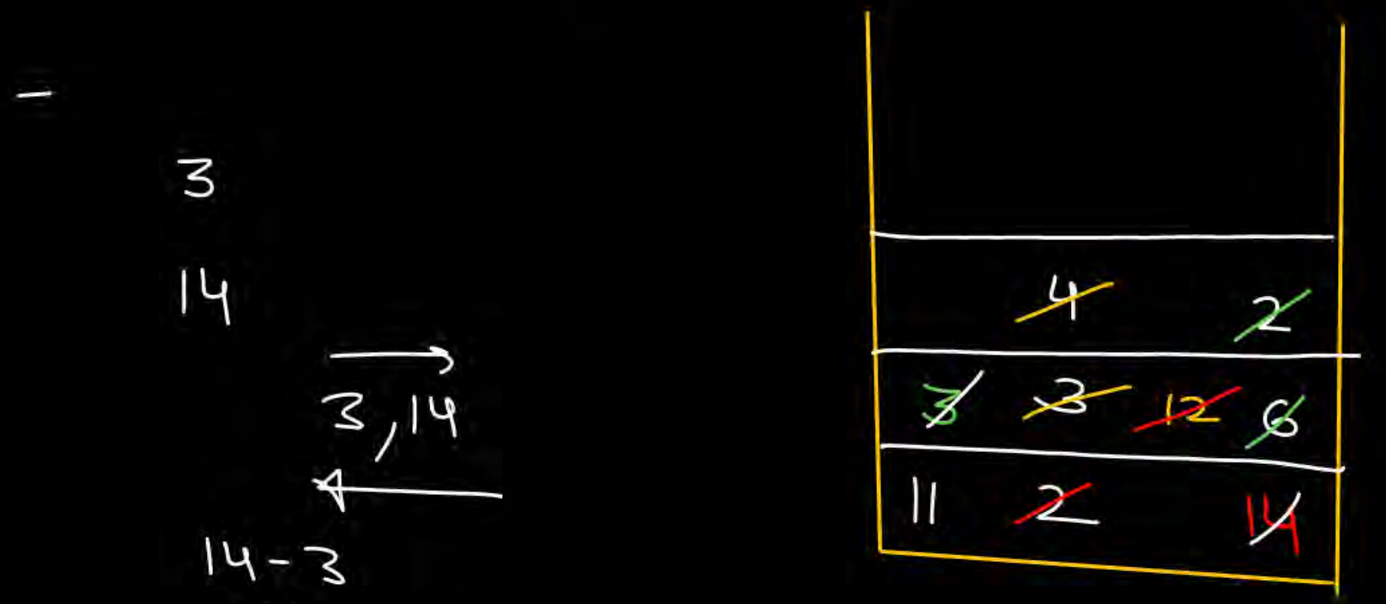
push 3



Ex 2:

infix : 2 + 3 x 4 - 6/2

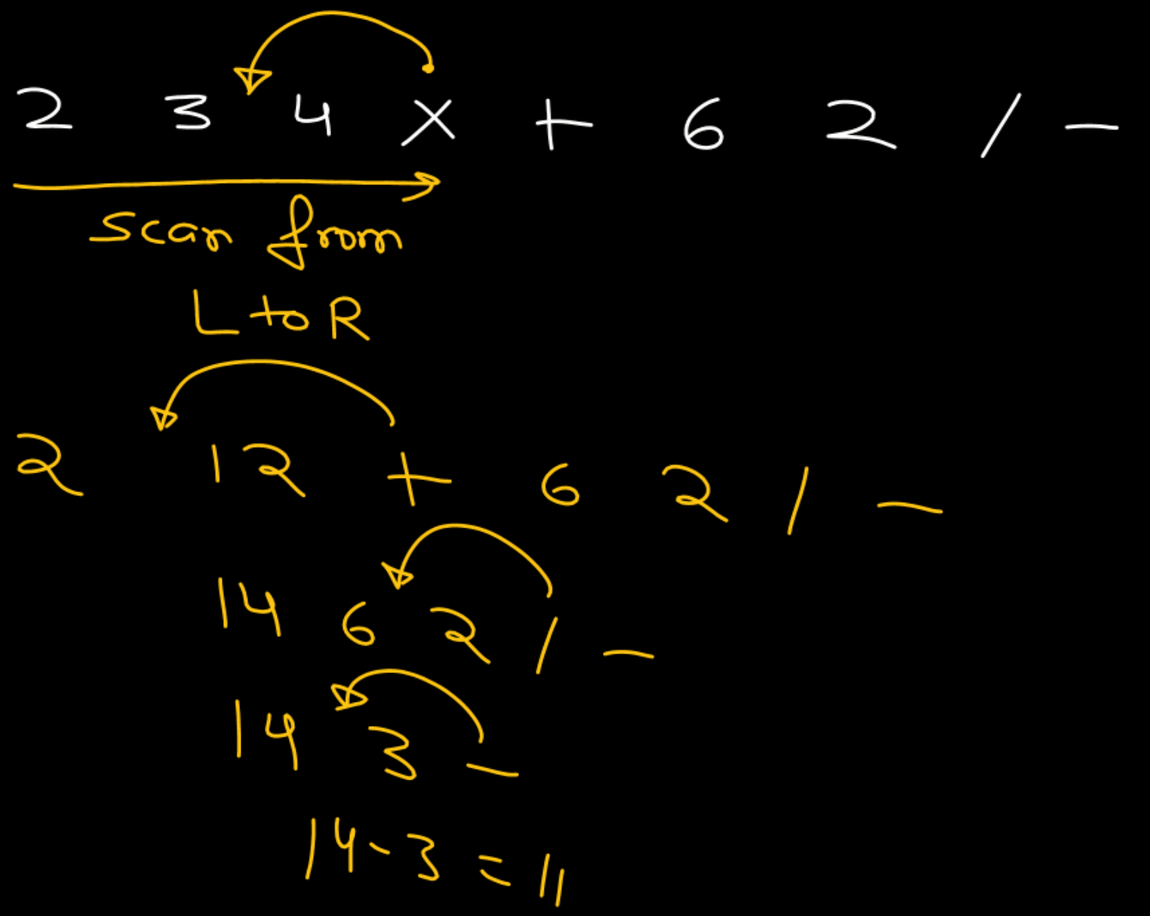
Postfix : 2 3 4 x + 6 2 / -



Ex 2:

infix : 2 + 3 x 4 - 6/2

Postfix : 2 3 4 x + 6 2 / -



A function f defined on stacks of integers satisfies the following properties.

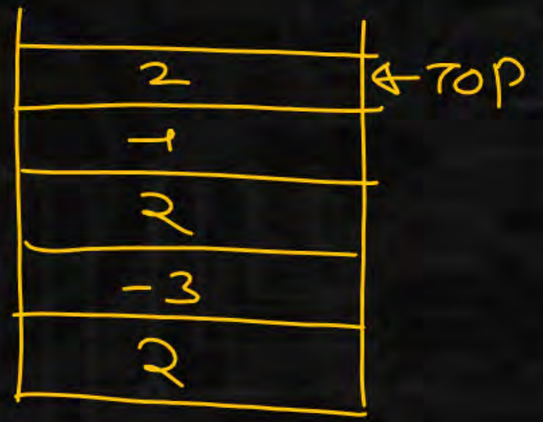
f(∅) = 0 and f(push (S, i)) = max (f(S), 0) + i for all stacks S and integer i.

If a stack S contains the integers 2, - 3, 2, - 1, 2 in order from bottom to top, what is f(S)?

f(stack is Empty) = 0

f(Push(S,2)) = max(f(S),0) + 2

= max(0,0)+2 = 2



A 6

B 4

C 3

D 2

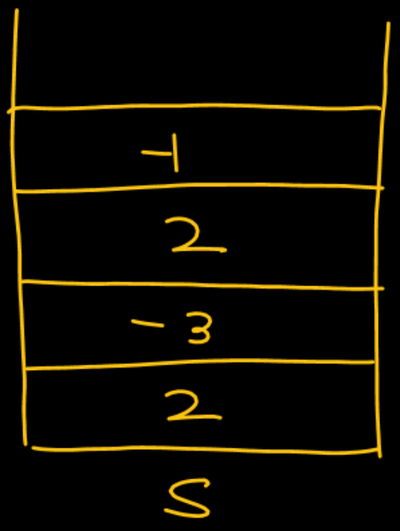
$$f(s) = 2$$

$$f(\text{Push}(s,-3)) = \max(f(s),0) + (-3)$$
$$= \max(2,0) + (-3) = -1$$

$$f(\text{Push}(s,2)) = \max(f(s),0) + 2$$
$$= \max(-1,0) + 2 = 2$$

$$f(\text{Push}(s,-1)) = \max(f(s),0) + (-1) = \max(2,0) + (-1) = 1$$

$$f(\text{Push}(s,2)) = \max(f(s),0) + 2$$
$$= \max(1,0) + 2 = 1 + 2 = 3$$



The result of evaluating the postfix expression 10 5 + 60 6/ * 8 – is

direct ✓
using stack ✗

A 284

B 213

C 142

D 71

10 5 + 60 6 / * 8 –

15 60 6 / * 8 –

15 10 * 8 –

150 8 –

150 – 8

= 142

The best data structure to check whether an arithmetic expression has balanced parentheses is-

A queue

B stack

C tree

D list

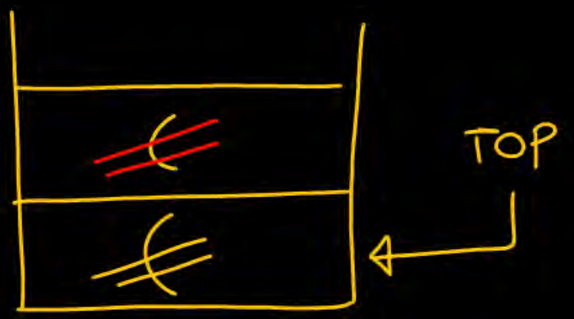
( )

( ) ( )

(( ))

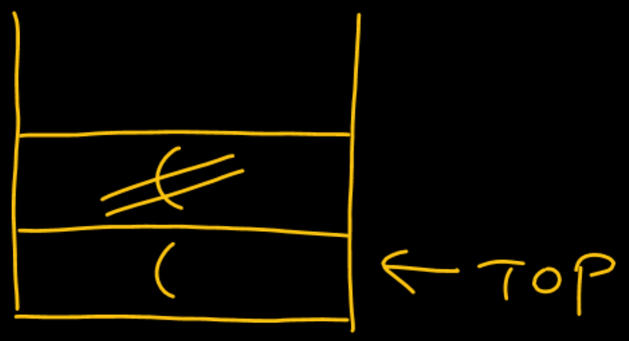
(( )) ( ) ( )


i/p :

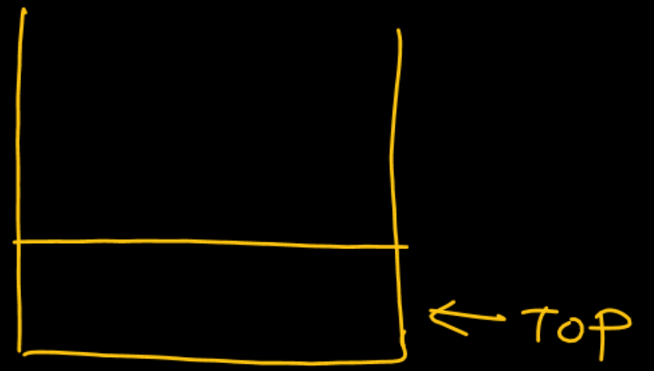

TOP


most recently
occured paranthesis

i/p : (( )



i/p :) ((



Prefix Evaluation

infix : 2 + 3 x 5

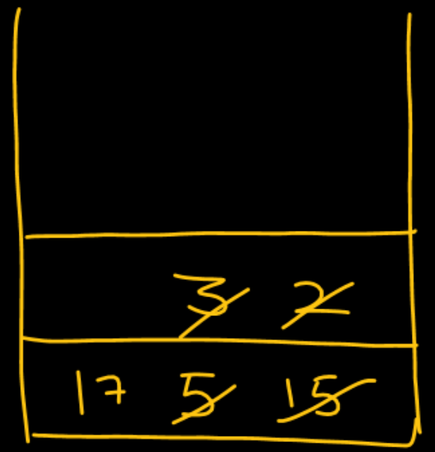
Prefix : + 2 x 3 5

Reverse Prefix : 5 3 x 2 +

x → 3, 5

3 x 5

15



+

2, 15 →

2 + 15

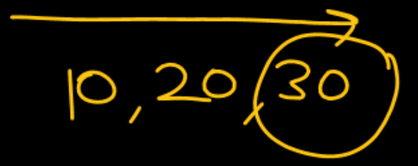
= 17

Stack

Push →

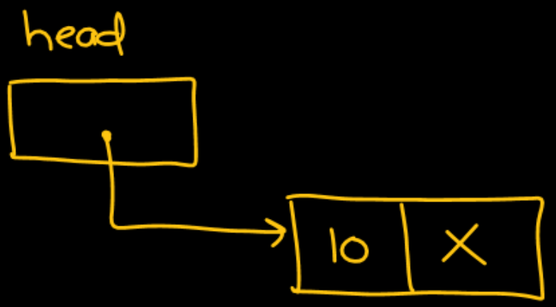
Pop →

using l.l.



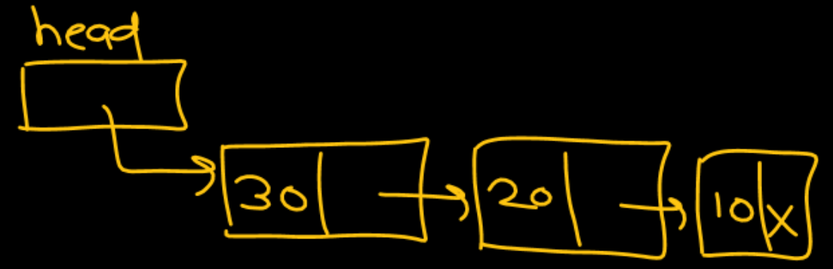
Push(10)

(i)



Push(20) (ii)

(iii) Push(30)



Pop() ⇒ deletion from beg.

Push() ⇒ Insert at begin

Recursion

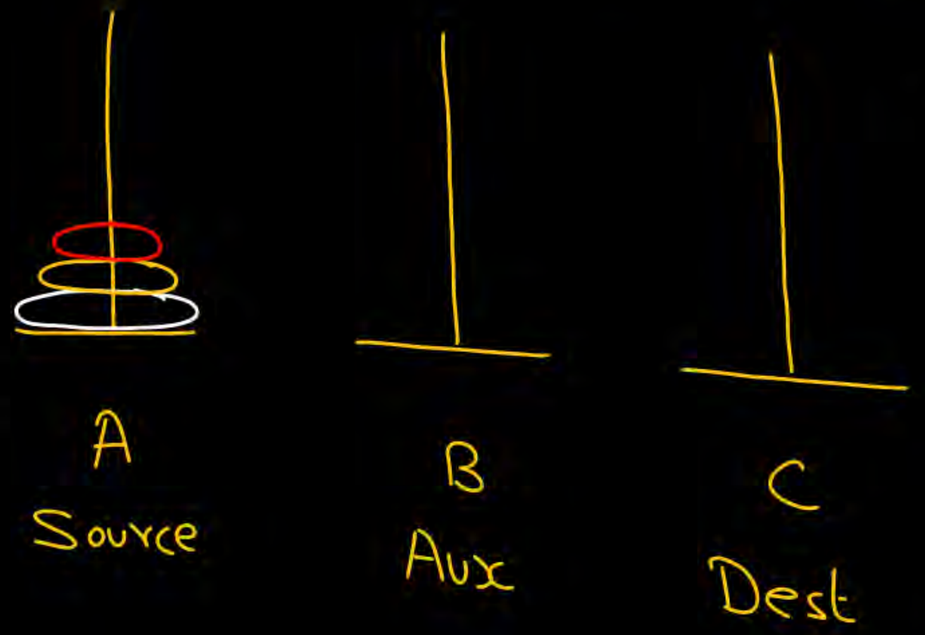
Tower of Hanoi Problem:

Given 3 pegs and n disc each of distinct size.

$n = 3$

(i) We can not put a large size plate above smaller size plate.

(ii) Move only 1 disk at a time

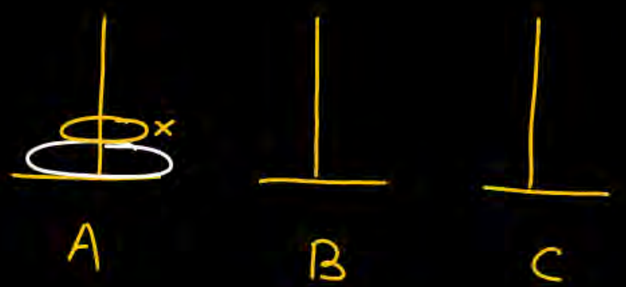


Target:
Move all disc from Source peg to Dest. peg.

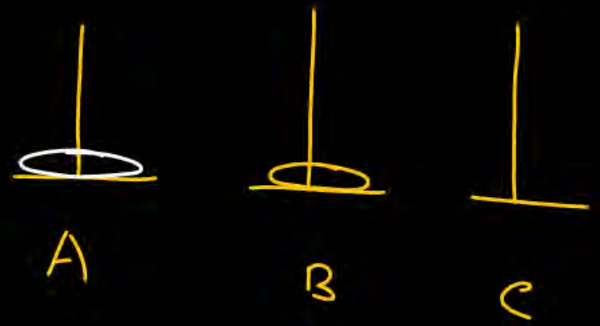
i/p :

o/p :

n = 2



A → B



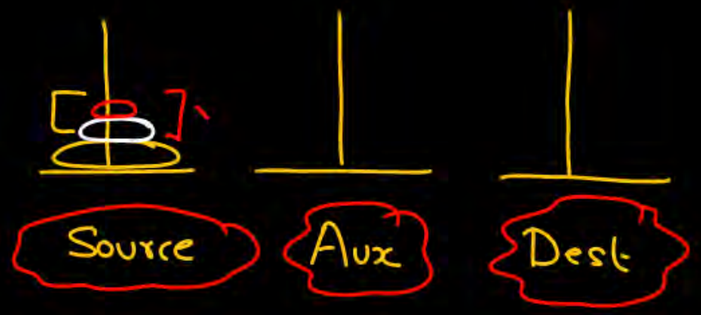
A → C

B → C

TOH(n , Source, Dest, Aux)

no. of plate

n=3



(i)

(ii)

(iii)

TOH(n, Source, Dest, Aux)

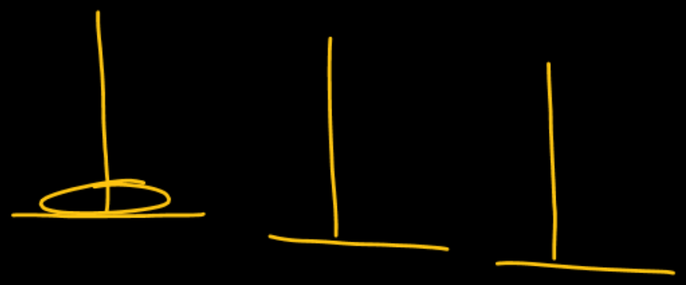
1.) TOH(n-1, Source, Aux, Dest)

2.) Move Source ⟶ Dest

3.) TOH(n-1, Aux, Dest, Source)

TOH

n ⇒ small

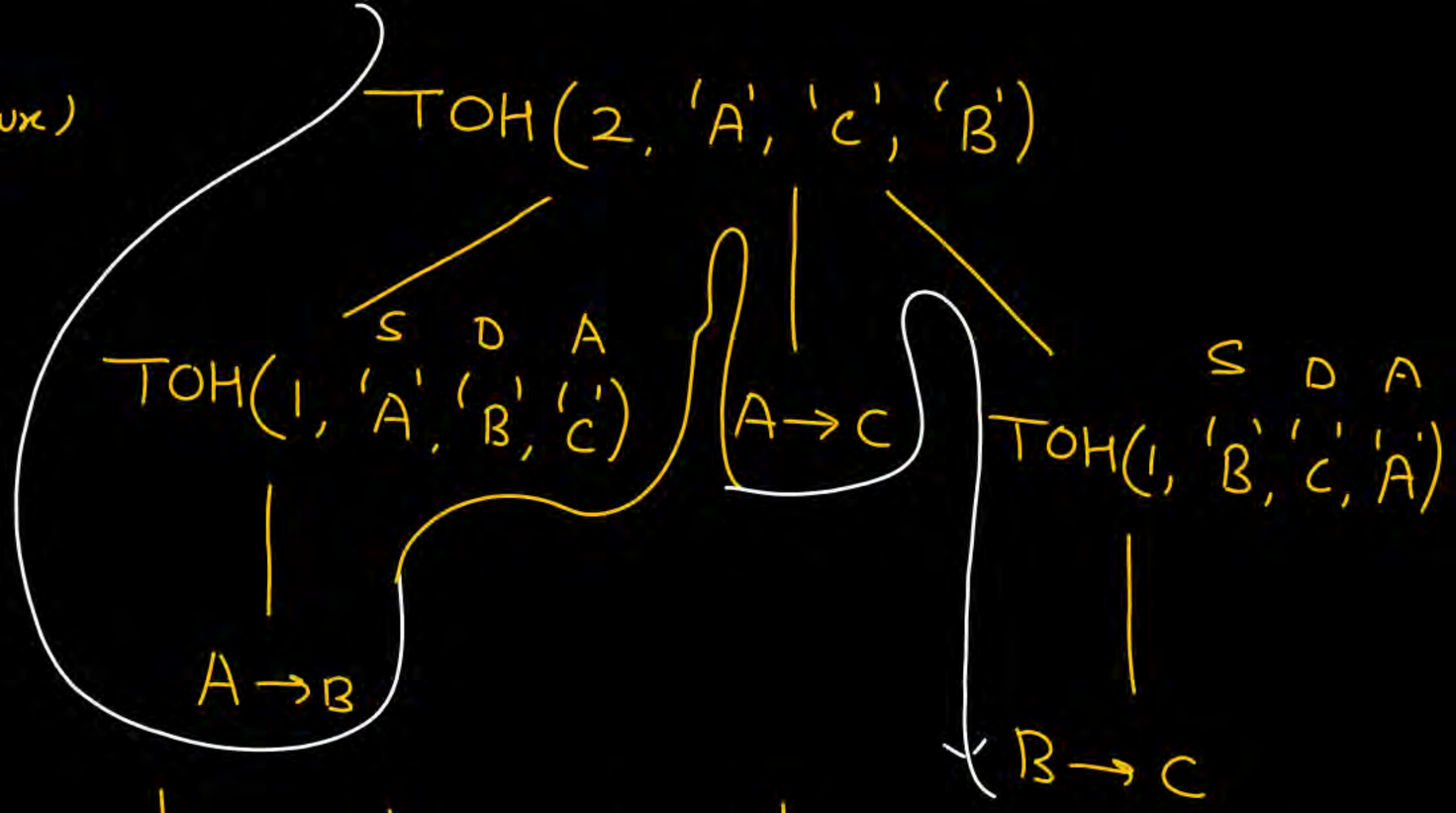


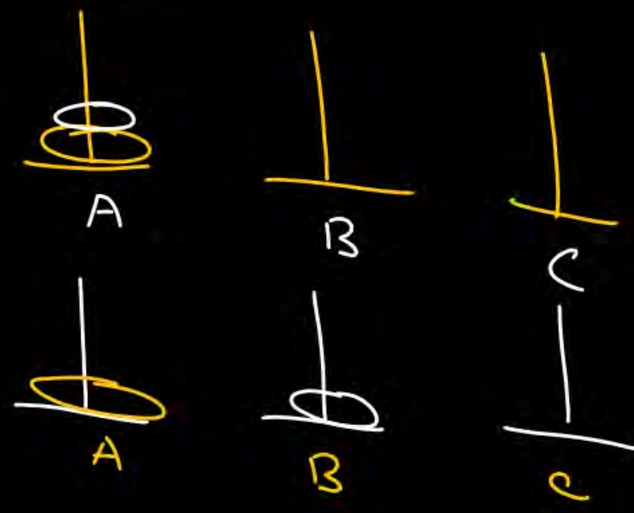
```
if ( n is small)
  {
   Easy case
   No rec. is req.
   Can be answered directly
  }
else {
     n is large
     Not Easy
     Rec. is needed
  }
```

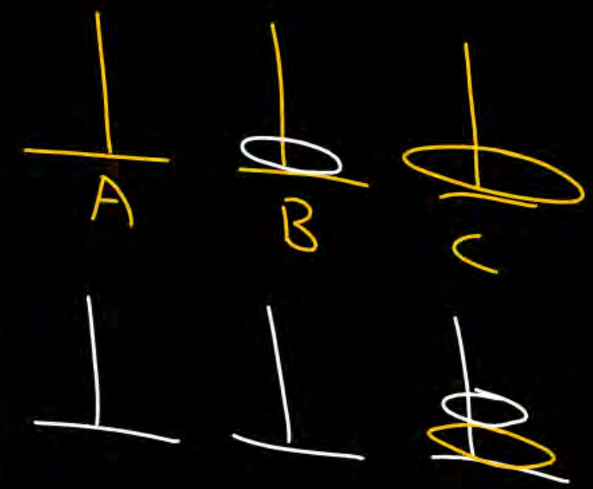
```
void TOH(int n, char Source, char Dest, char Aux)
{
if(n==1)
{
printf("%c ⟶ %c", Source, Dest);
return;
}
TOH(n-1, Source, Aux, Dest);
printf("%c ⟶ %c", Source, Dest);
TOH(n-1, Aux, Dest, Source);
}
```

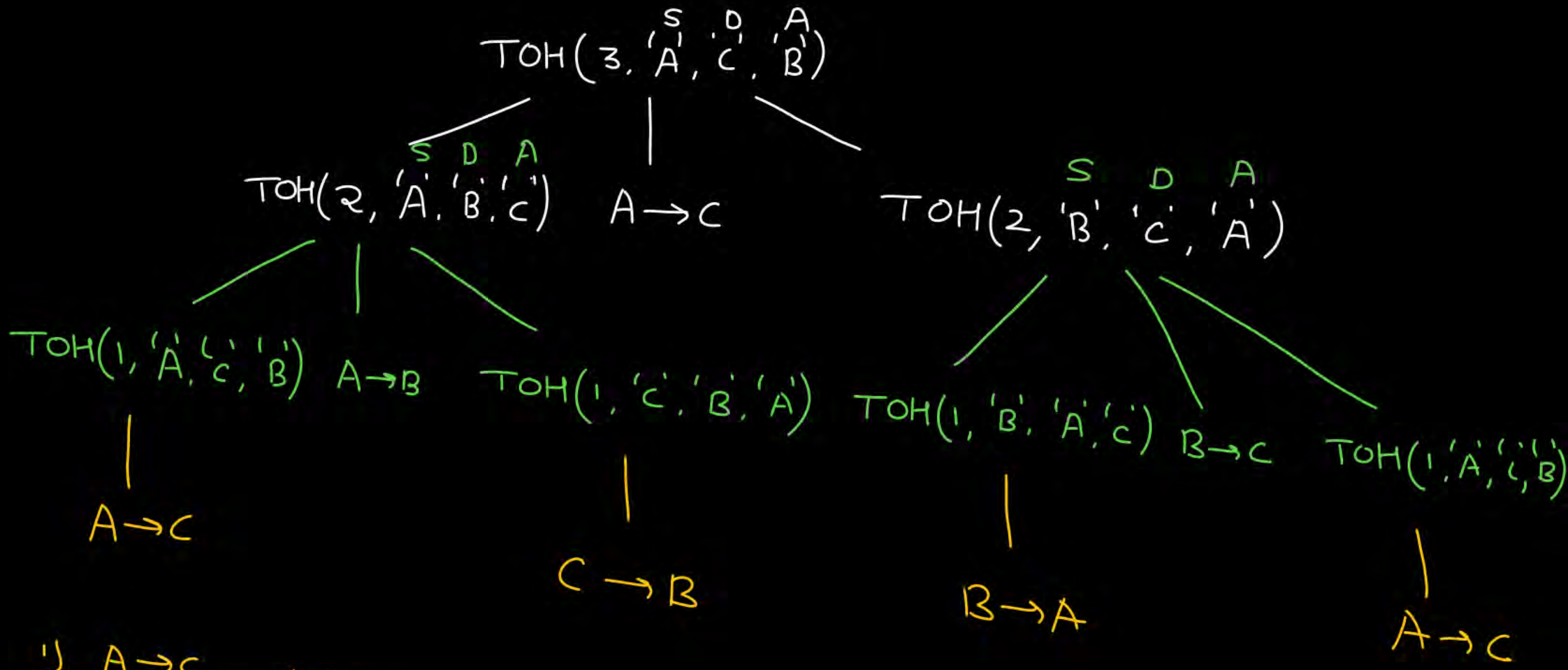
```
void main(){
int n;
pf("Enter the value of n");
sf("%d", &n);
TOH(n, 'A', 'C', 'B');
}
```

```
void TOH(int n, char Source, char Dest, char Aux)
  {
  if(n==1)
   {
    printf("%c → %c", Source, Dest);
    return;
   }
  TOH(n-1, Source, Aux, Dest);
  printf("%c → %c", Source, Dest);
  TOH(n-1, Aux, Dest, Source);
  }
```

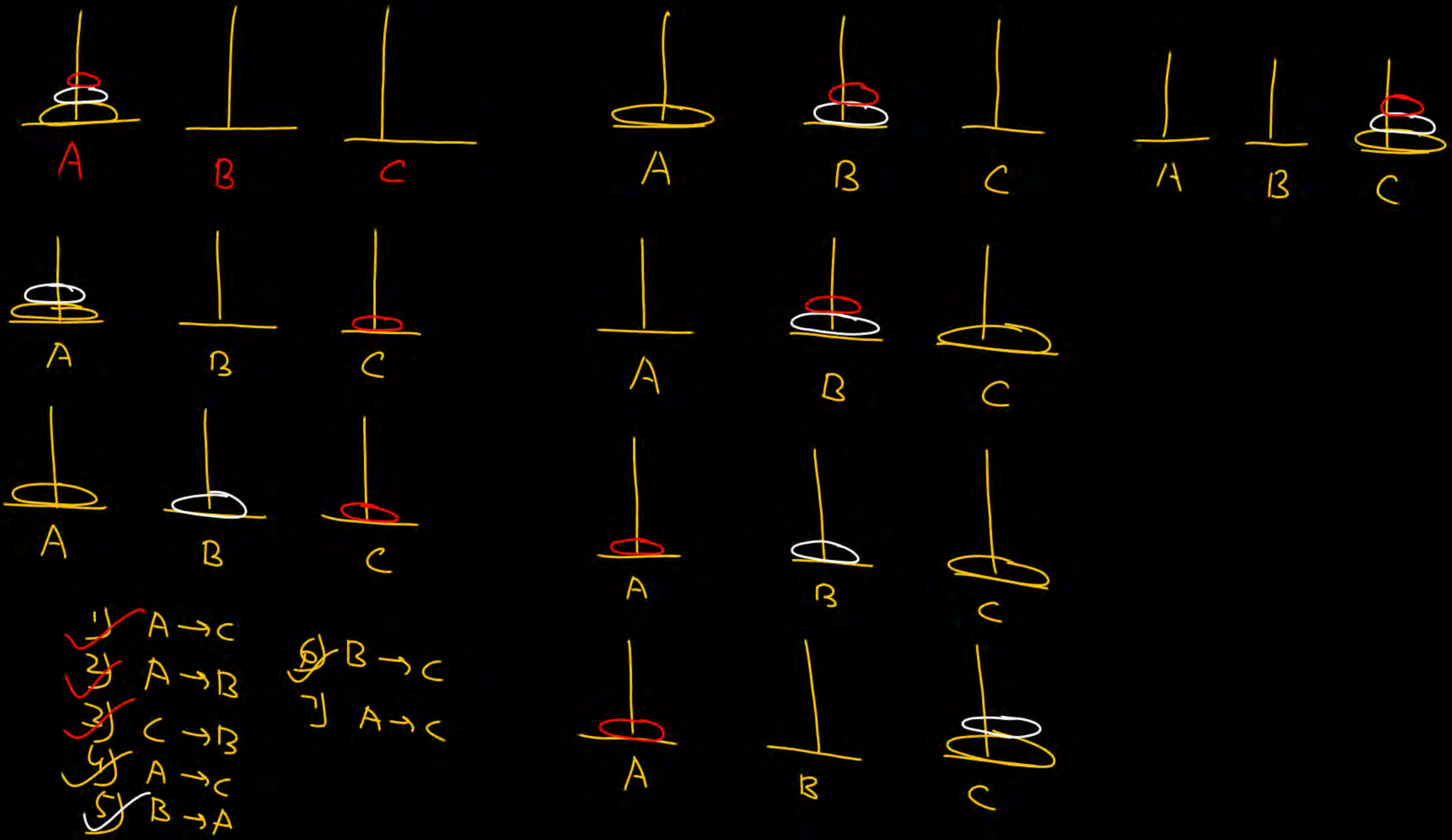




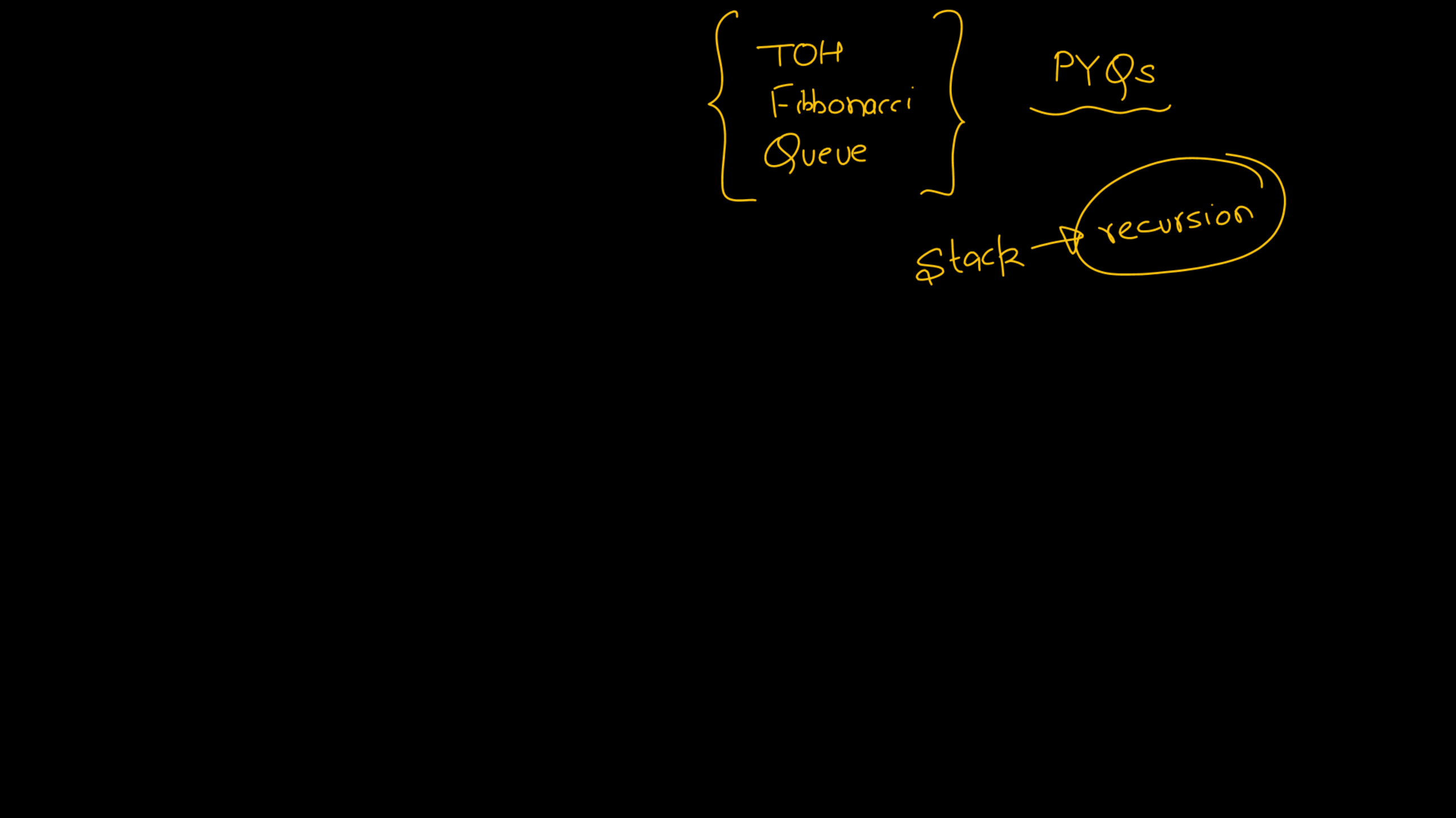




1) A→C
2) A→B
3) C→B
4) A→C
5) B→A
6) B→C
7) A→C



A
B
C
1) A → C
2) A → B
3) C → B
4) A → C
5) B → A
6) B → C
7) A → C


TOH
Fibbonacci
Queue
PYQs
Stack
recursion

Which of the following is essential to convert an infix expression to post-fix expression efficiently?

A. An operator stack

B. An operand stack

C. An operator and an operand stack

D. A parse tree

The following sequence of operations is performed on stack: PUSH(10), PUSH(20), POP, PUSH(10), PUSH(20), POP, POP, POP, PUSH(20), POP. The sequence of the value popped out is-

A 20 10 20 10 20

B 20 20 10 10 20

C 10 20 20 10 20

D 20 20 10 20 10

20, 20, 10, 10, 20

Which of the following permutations can be obtained in the output (in the same order) using a stack assuming that the input sequence is 1, 2, 3, 4, 5 in that order?

A) 3, 4, 5, 1, 2 ×

Push(1), Push(2), Push(3), Pop(), Push(4), Pop(), Push(5), Pop()

3 4

B) 3, 4, 5, 2, 1

C) 1, 5, 2, 3, 4

D) 5, 4, 3, 1, 2

A program attempts to generate as many permutations as possible of the string "abcd" by pushing the character a, b, c, d in the same order onto a stack but it may pop off the top character at any time. Which one of the following strings CANNOT be generated using this program?

A. abcd ✓

B. dcba ✓

C. cbad ✓ Push(a), Push(b), Push(c), Pop(), Pop(), Pop(), Push(d), Pop()

D. cabd

Which of the following is essential to convert an infix expression to post-fix expression efficiently?

A. An operator stack

B. An operand stack

C. An operator and an operand stack

D. A parse tree

The postfix expression corresponding to the infix expression a+b*c-d^e^f is-

a+b*c-d^[ef^]

a+b×c-[def^^]

= a+[bcx]-[def^^]

[abcx+]-[def^^]

= abcx+def^^-

^
R to L

d^(e^f)
1

The following postfix expression with single digit operand is evaluated using a stack:

8 2 3 ^ / 2 3 * + 5 1 * -

Note that ^ is the exponentiation operator. The top two elements of the stack after the first * is evaluated are:

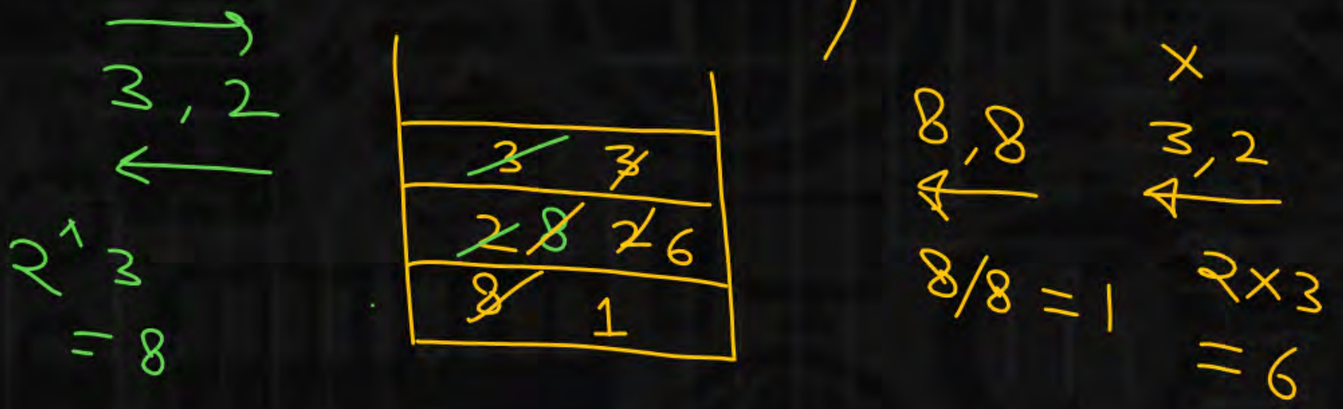
8 2 3 ^ / 2 3 * + 5 1 * -

(A) 6, 1

(B) 5, 7

(C) 3, 2

(D) 1, 5



Consider the following C program:

```
#include <stdio.h>
#define EOF -1
void push (int);
/*push the argument on the stack*/
int pop (void); /*pop the top of the stack */
void flagError();
int main()
{
        int c, m, n, r,
        while((c = getchar())! = EOF){
        if (isdigit(c)  push(c);
        else if ((c = = '+') || (c = = '*')){
                m = pop();
                n = pop();
                r = (c = = '+') ? n + m : n*m;
                push(r);
         }else if (c ! = '')
          flagError( );
        }
        printf(" %c ", pop());
}
```

What is the output of the program for the following?

5 2* 3 3 2 + * +

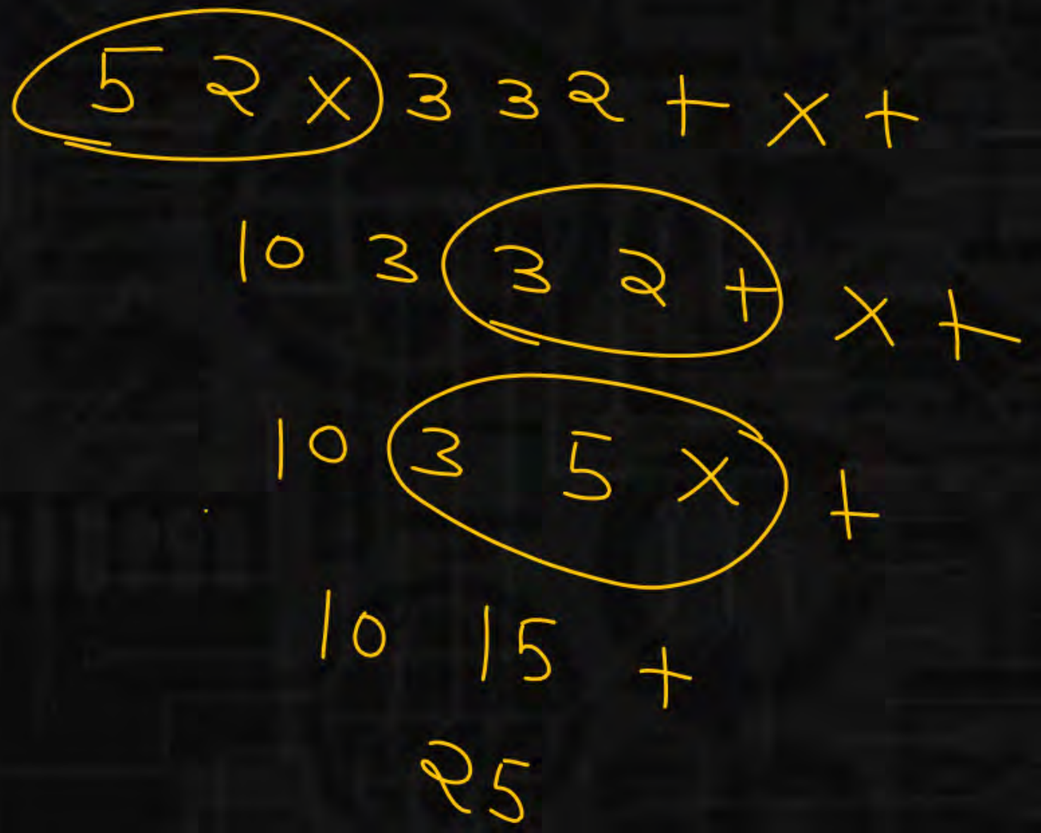
+ +

A 15

B 25

C 30

D 150



Let S be a stack of size n>=1. Starting with the empty stack, suppose we push the first n natural numbers in sequence, and then perform n pop operations. Assume that PUSH and POP operations take X secs each and Y seconds elapse between the end of one such stack operation and the start of the next operation. For m>=1, define the stack life-time of m as the time elapsed from the end f PUSH(m) to the start of POP operation that removes m from S. The average stack-life of an element is-

H.W

A n(X+Y)

B n(X+Y)-X

C 3Y+2X

D Y+2X

The attributes of three arithmetic operators in some programming language are given below.

Operator	Precedence	Associativity	Arity
+	High	Left	Binary
–	Medium	Right	Binary
*	Low	Left	Binary

The value of the expression 2 – 5 + 1– 7 * 3 in this language is 9.

$[2-(-1)]\times 3$

3×3

9

$[2-((5+1)-7)]\times 3$

$(2-(6-7))\times 3$


THANK YOU GW
SOLDIERS !