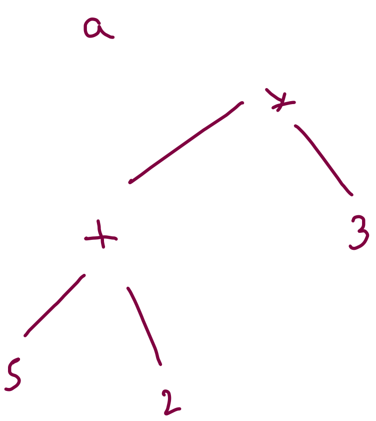
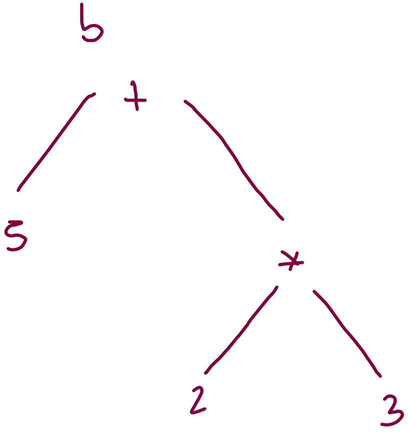
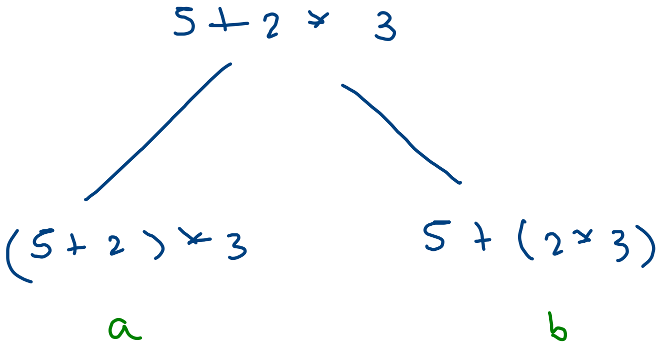
expressions

- infix $a+b$
- prefix $+ab$
- postfix $ab+$







inorder : $5 + 2 * 3$

preorder : $* + 5 2 3$

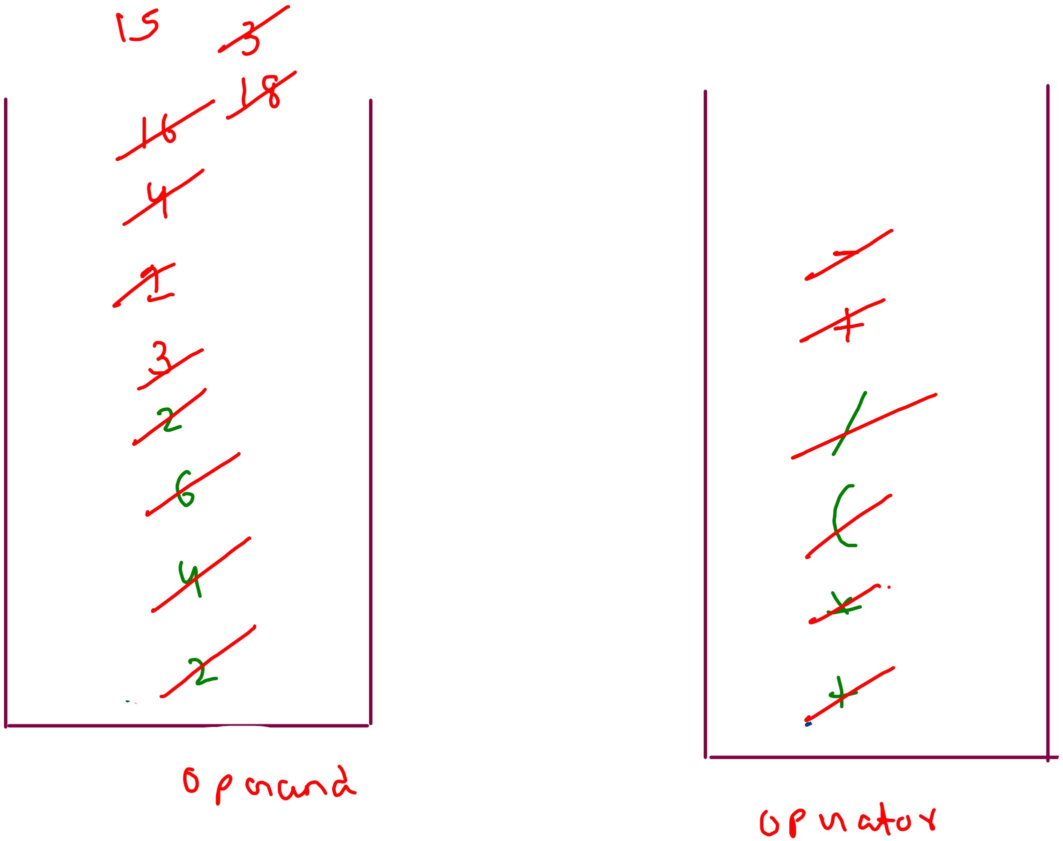
postorder $5 2 + 3 *$

$5 + 2 * 3$

$+ 5 * 2 3$

$5 2 3 * +$

$2 + 4 * (6 / 2 + 1) - 3$.



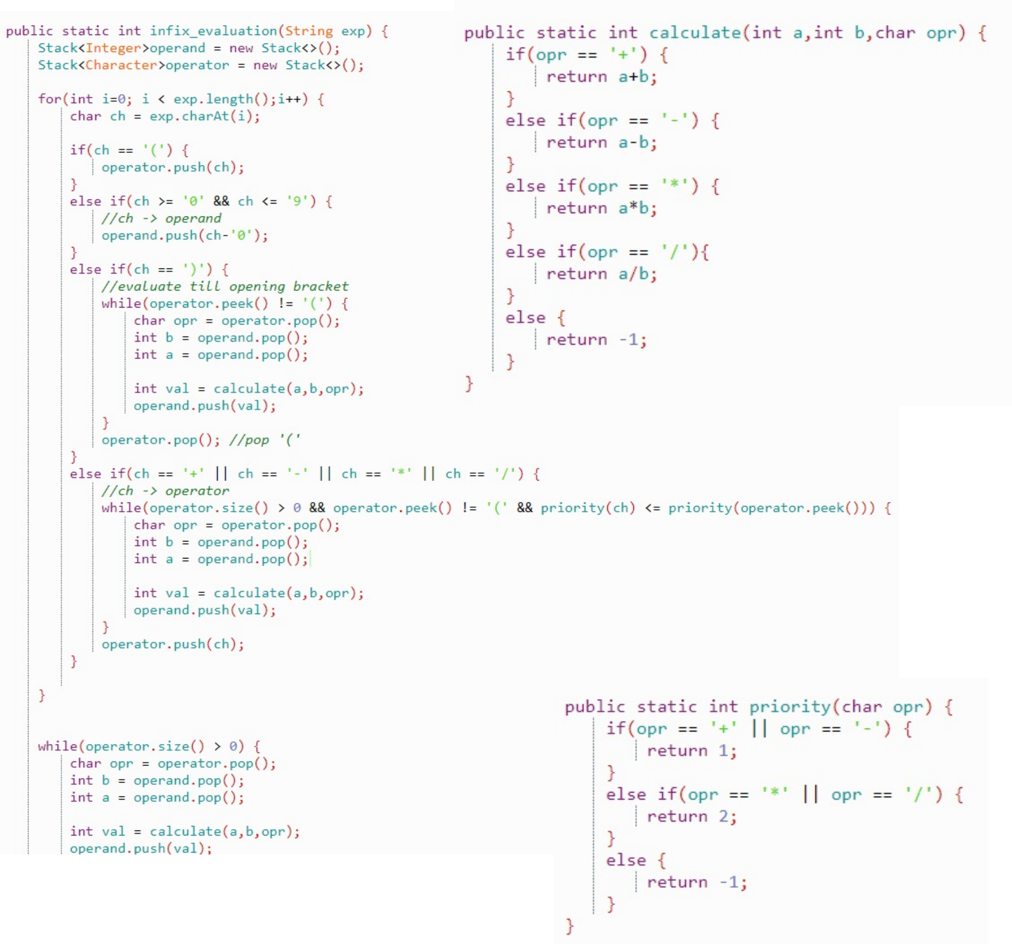
(i) operand -> push it in operand stack.

(ii) operator -> evaluate high or equal operator first, then push this current operator.

(iii) (-> push it in operator stack.

(iv)) -> evaluate till opening.

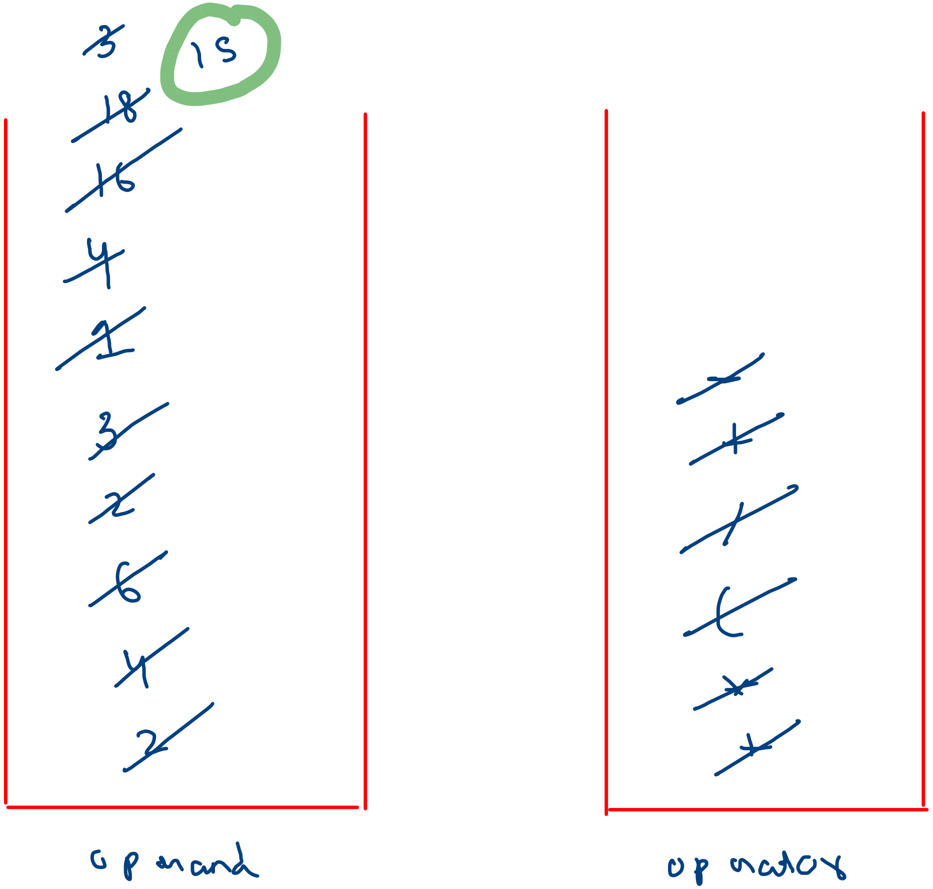
2 + 4 * (6 / 2 + 1) - 3 .



```
public static int infix_evaluation(String exp) {
    Stack<Integer>operand = new Stack<>();
    Stack<Character>operator = new Stack<>();

    for(int i=0; i < exp.length();i++) {
        char ch = exp.charAt(i);

        if(ch == '(') {
            operator.push(ch);
        }
        else if(ch >= '0' && ch <= '9') {
            //ch -> operand
            operand.push(ch-'0');
        }
        else if(ch == ')') {
            //evaluate till opening bracket
            while(operator.peek() != '(') {
                char opr = operator.pop();
                int b = operand.pop();
                int a = operand.pop();

                int val = calculate(a,b,opr);
                operand.push(val);
            }
            operator.pop(); //pop '('
        }
        else if(ch == '+' || ch == '-' || ch == '*' || ch == '/') {
            //ch -> operator
            while(operator.size() > 0 && operator.peek() != '(' && priority(ch) <= priority(operator.peek())) {
                char opr = operator.pop();
                int b = operand.pop();
                int a = operand.pop();

                int val = calculate(a,b,opr);
                operand.push(val);
            }
            operator.push(ch);
        }
    }


    while(operator.size() > 0) {
        char opr = operator.pop();
        int b = operand.pop();
        int a = operand.pop();

        int val = calculate(a,b,opr);
        operand.push(val);
```

```
public static int calculate(int a,int b,char opr) {
    if(opr == '+') {
        return a+b;
    }
    else if(opr == '-') {
        return a-b;
    }
    else if(opr == '*') {
        return a*b;
    }
    else if(opr == '/'){
        return a/b;
    }
    else {
        return -1;
    }
}
```

```
public static int priority(char opr) {
    if(opr == '+' || opr == '-') {
        return 1;
    }
    else if(opr == '*' || opr == '/') {
        return 2;
    }
    else {
        return -1;
    }
}
```



a*(b-c+d)/e

abc-d+*e/

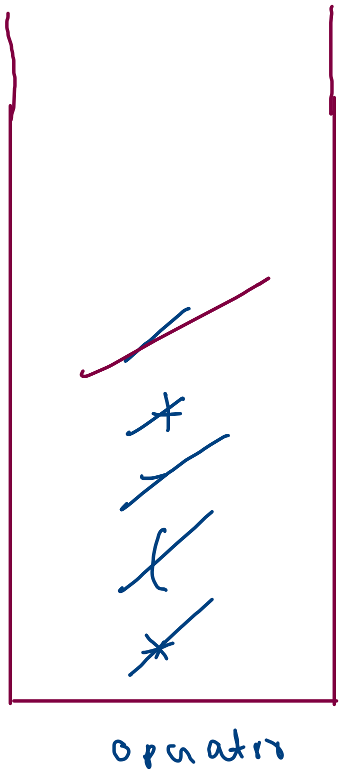
/*a+-bcde

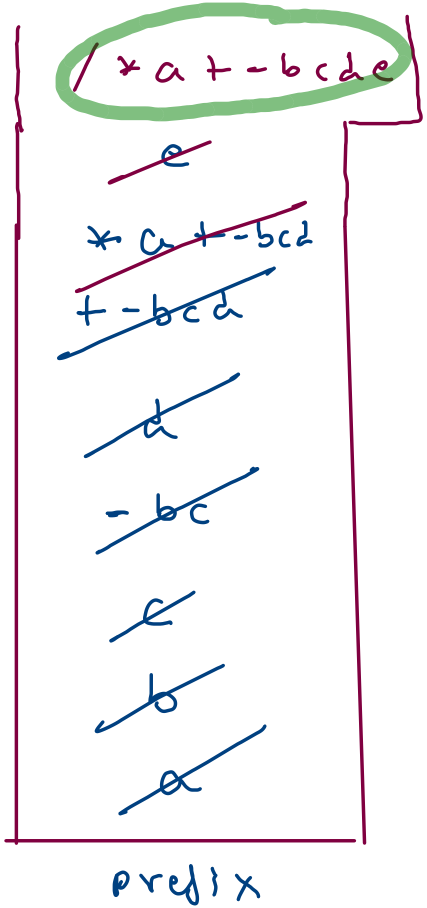
$a * (b - c + d) / e$

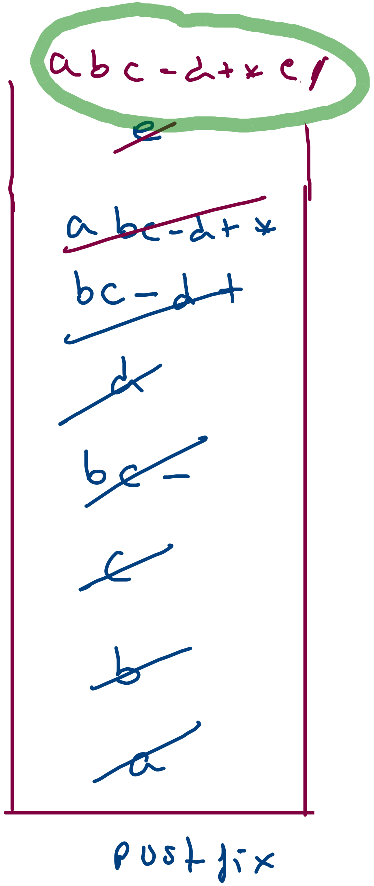
3 1 2 4

pre = opr x y

post = x y opr





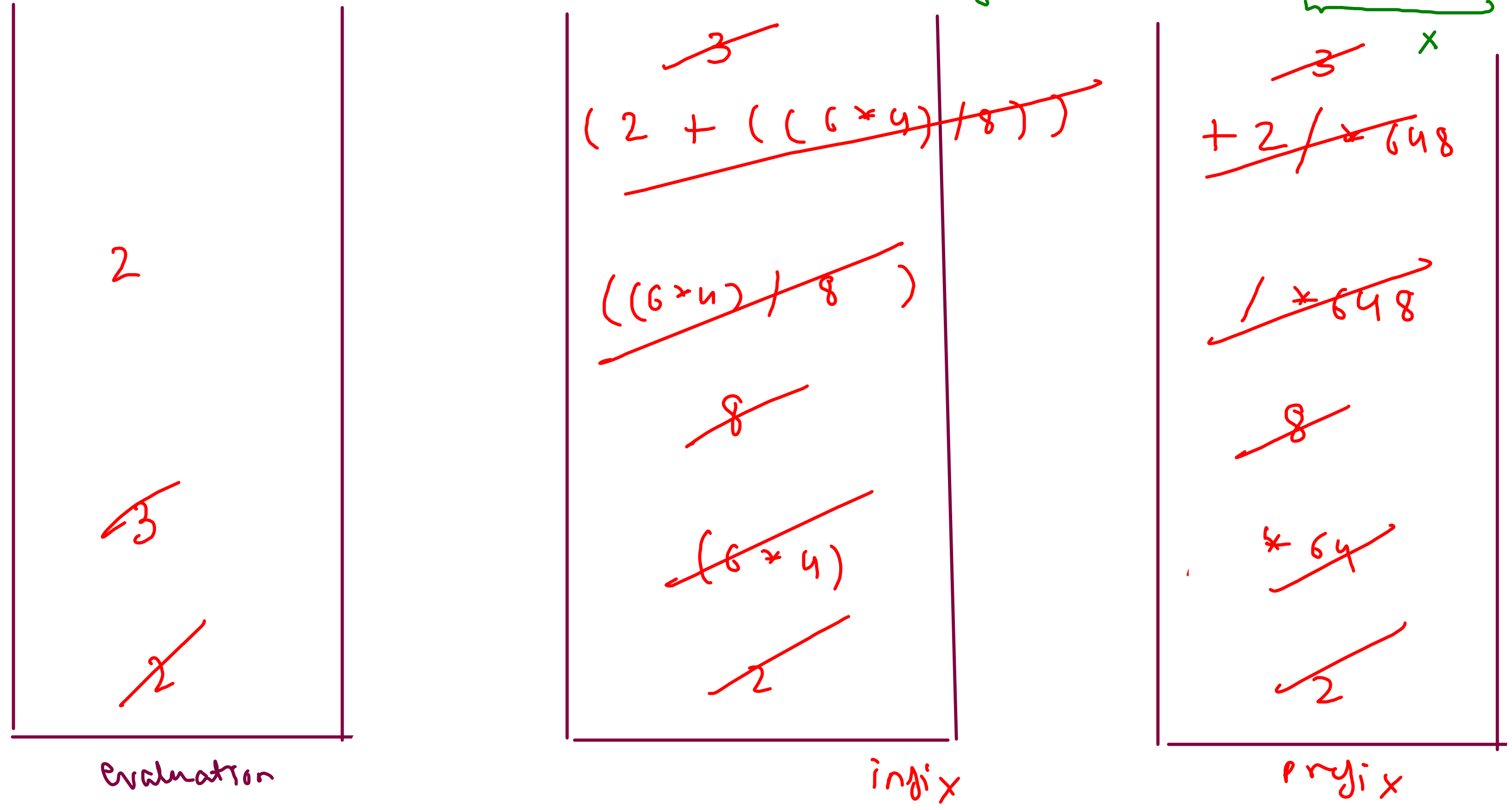


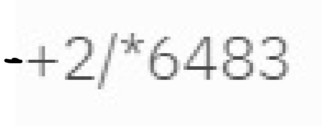
2 6 4 * 8 / + 3 -

2
((2+((6*4)/8))-3)
-+2/*6483

$$\underbrace{-}\ \underbrace{((2+((6*4)/8))}_{x}\ \underbrace{-}_{opr}\ \underbrace{3}_{y}\ \underbrace{)}$$

$$\overset{opr}{\underbrace{-}}\ \underbrace{+2/*648}_{x}\ \underbrace{3}_{y}$$

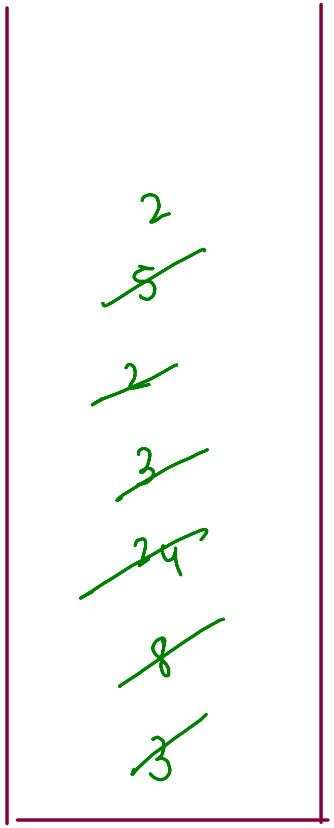




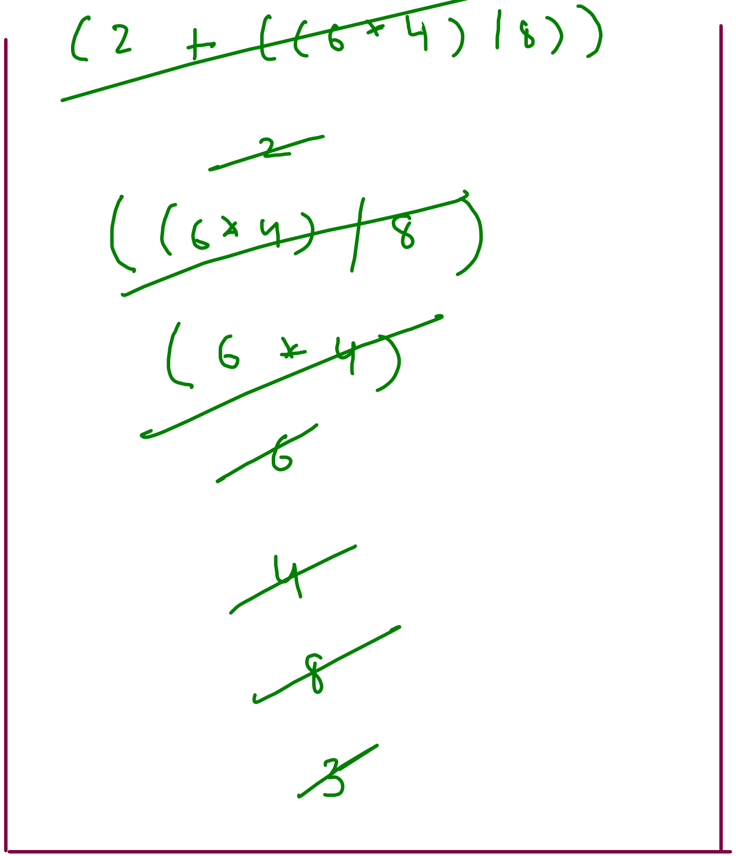
-+2/*6483

((2 + ((6*4)/8)) - 3)

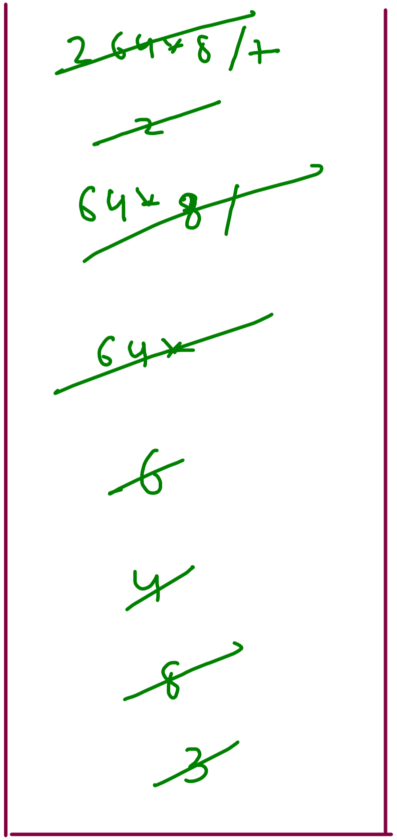
2 6 4 * 8 / + 3 -



eval



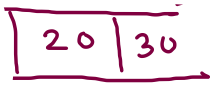
infix



postfix

queue introduction

FIFO



q.add(10)

q.add(20)

q.add(30)

q.remove() → 10

q.peek() → 20

add

remove

peek

size

n = 3

0 1 0

0 1 1

1 0 1

1 1 0

1 1 1

~~0~~	~~1~~	~~01~~	~~10~~	~~11~~	010	011	101	110	111