```
for ( String.length){

    if (value)
        if(unary op next)
            do unary
        Valstk.push(string(i))

    else
        if(prec(op, nextop)
            doop();
        Opstk.push(string(i))

}
do OP


doOP()
    while(Valstk≥1 ∧  prec(refOp)≤(prec(Opstk.top()))

        int x = Valstk.pop
        int y = Valstk.pop
        op = opstk.pop
        Valstk push(xopy)

[illegible]
```

b)

```
                      //some iterator that will go through
                        string and parse properly
domath(string)          ↓
    double currentN = nextValue(string)
    string op1 = nextOP(string)
    double currentN2 = nextValue(string)
    string op2 = nextOP(string)

    if(prec Op ≤ prec Op2)
      currentN
      ~~Val.push~~(currentN op currentN2)
        domath(string) //string minus the parts the
                                iterator has seen
    else
        domath(string)
        ~~val.push~~(currentN op currentN2)
        currentN
```

c) stack version: O(n)

n is the size of the strings, since comparing precedence and doOp() are constants, cn+c = O(n)

Recursion Version: O(n)

for linear recursion, it will recurse n times, n being length of String

Scanned by CamScanner