A)

```
Algorithm Calculator:
    Set fileName to "input.txt"
    Set line to empty string
    Try
        Open fileName for reading as bufferedReader
        Open "Output.txt" for writing as writer
        Create instance of Calculator
        While bufferedReader can read into line
            If line is empty
                Continue
            Set result to evaluateExpression(line)
            Write line + " = " + result to writer
    Catch FileNotFoundException
        Print "Unable to open file: " + fileName
    Catch IOException
        Print "Error reading or writing file"
End
```

```
Method evaluateExpression(str) returns double
    Initialize space as empty list
    Set removeSpace to "(" + remove all whitespace from str + ")"
    Set num to empty string
    Set count to 0
    For each character c in removeSpace
        If c is digit or '.'
            Append c to num
            Set count to 0
            Continue
        If count is 0
            If num is not empty
                Add num to space
                Set num to empty string
            Set count to 1


        If c is not digit
            If two-character operator found
```

```
            Set op to two-character operator
            Add op to space
            Skip next character
            Continue
        Add c as string to space
For each string s in space
    If s is ")"
        While top of operation is not "("
            Call operate()
        Pop operation
    Else if s is operator
        While precedence of s <= precedence of top of operation
            Call operate()
        Push s to operation
    Else if s is "("
        Push s to operation
    Else
        Push parsed double value of s to number
```

```
    While operation is not empty
        Call operate()
    Return top value from number
End
Method operate()
    Set operator to top of operation
    Set b to top of number
    Set a to top of number
    Switch operator
        Case "+"
            Set result to a + b
        Case "-"
            Set result to a - b
        Case "^"
            Set result to power(a, b)
        Case "*"
            Set result to a * b
        Case "/"
```

```
            Set result to a / b
        Case ">"
            Set result to (a > b) ? 1 : 0
        Case ">="
            Set result to (a >= b) ? 1 : 0
        Case "<"
            Set result to (a < b) ? 1 : 0
        Case "<="
            Set result to (a <= b) ? 1 : 0
        Case "=="
            Set result to (a == b) ? 1 : 0
        Case "!="
            Set result to (a != b) ? 1 : 0
    Push result to number
End
Method isNumber(c) returns boolean
    Return c is digit or '.'
End
```

```
Method isOperator(s) returns boolean
    Return s is "+" or "-" or "*" or "/" or "^" or ">" or ">=" or "<" or "<=" or "==" or "!="
End
Method stringOperator(operator) returns integer
    Switch operator
        Case "+" or "-"
            Return 1
        Case "*" or "/"
            Return 2
        Case "^"
            Return 3
        Case ">" or ">=" or "<" or "<="
            Return 4
        Case "==" or "!="
            Return 5
        Default
            Return 0
End
```

B)

The time complexity of my algorithm is O(n). The method evaluateExpression loops through every character in the input string and into the numbers as well as operations, and this takes O(n) times where n is the length of the input string. The elements are then pushed and popped from the stack many times but since elements are parsed maximum once, it takes O(n) times. The space complexity of my algorithm is also O(n) since the size of my list is O(n), where the list is made of parsed elements.