

Problem 4

All the edge cases and their code snippet

a)edge cases in problem 1:

1)Empty postfix expressions:

```
if len(input) == 0 or input == ['']: # empty input edge case [edge case]
    raise ValueError("Empty expression")
```

2)Malformed postfix expressions (insufficient operands)

```
try:
    sign_node = TreeNode(i)
    sign_node.right = temp_stack.pop()
    sign_node.left = temp_stack.pop() # if not enough operands, pop
will raise IndexError [edge case]
except IndexError:
    raise ValueError("Malformed expression - insufficient operands for
operator") #[edge case]
```

3)Malformed postfix expressions(too many operands):

```
if len(temp_stack) != 1: # more than one node left means too many operands [edge case]
    raise ValueError("Malformed expression - too many operands") #[edge case]
```

b)edge cases in problem 3

1)Empty postfix expressions:

If the list is empty, I have kept a check at the top of the function

```
if len(lst) == 0: #empty postfix expression edge case(if the input is empty)
[edge case]
    raise ValueError("Empty postfix expression")
```

2)Malformed postfix expressions (insufficient operands, too many operands):

If we have fewer numbers to compute and finally, when we have many numbers left(fewer signs)
I have kept a try and except block to catch the pop failure

```
try:
    right = self.pop() #negative values also handled [edge case]
    left = self.pop()#negative values also handled [edge case]
except IndexError: #if the stack doesn't have enough variables to pop
[edge case]
    raise ValueError("Malformed postfix expression")
```

```
if self.count != 1: #if the stack has more numbers or signs left than what can be
handled [edge case]
```

```
raise ValueError("Malformed postfix expression")
```

3)Division by zero:

I use the if condition to check if the right value or the denominator is 0 , then raise a ZeroDivisionError

```
case "/":  
    if right == 0:#dividing by zero will result is infinity , so  
raise an error [edge case]  
    raise ZeroDivisionError("The denominator is 0 and will  
result in infinity")
```

4)Invalid tokens (non-numeric operands, unsupported operators):

If we have a value like 1df4, it raises an error

Since I have a switch case, my code doesn't handle any other operators except +,-,*,/

```
try:  
    self.push(int(i)) #negative values also handled [edge case]  
except ValueError: #if the value is invalid like "1df4" [edge case]  
    raise ValueError(f"Invalid value = {i}")
```

```
match i: #switch case to do coorect operation based on the sign  
    case "+":  
        self.push(left+right)  
    case "-":  
        self.push(left-right)  
    case "*":  
        self.push(left*right)  
    case "/":  
        if right == 0:#dividing by zero will result is infinity , so  
raise an error [edge case]  
        raise ZeroDivisionError("The denominator is 0 and will  
result in infinity")  
    num = int(left/right)#negative values also handled [edge case]  
    self.push(num)
```

5)Very large numbers or results

I'm pretty sure Python can already handle very large numbers, in java and C, int is only 8 bytes or 64 bits, but in Python, it can grow if the number is bigger than 8 bytes

6)Negative numbers in the expression:

Since I'm type casting it using int(), it is treated as any negative number.

Input for each of the edge cases(test cases for edge cases)

a)edge case inputs for problem 1

1)Empty postfix expressions:

Input: " "

2)Malformed postfix expressions

i)(insufficient operands)

Input: "5,+"

ii)(too many operands)

Input: "1,2, 3,+"

b)edge case inputs for problem 3

1)Empty postfix expressions:

Input: " "

2)Malformed postfix expressions

i)(insufficient operands)

Input: "5 +"

ii)(too many operands)

Input: "1 2 3 +"

3)Division by zero

Input: "100 10 * 0 /"

4)Invalid tokens (non-numeric operands, unsupported operators)

Input: "10 1df4 +"

5)Very large numbers or results

Input: "9999999999999999 1 +"

Output: 1000000000000000000000000

6)Negative numbers in the expression:

Input: "-3 -2 *"

Output: 6

AI usage/external usage statement

1) I have referred to this material from [w3schools.com](https://www.w3schools.com) to get my concepts brushed up on stacks:
https://www.w3schools.com/python/python_dsa_stacks.asp

2) I have not used any AI for this assignment