Guillermo Rivera
Project 1
CS 2223

For our initial lab, we were asked to test out three algorithms and compare them. These algorithms were the Euclid method, Consecutive Integer Checking method, and the Middle school procedure method. Using python 3.7.0a2 and the text editor IDLE, the algorithms were implemented and tested. For both the Euclid and CIC method, the algorithms were able to run themselves without any helper functions. When it came to the MSP, which requires first finding the prime factors of each input, a helper function which calculates the prime factors was used to reduce the clutter in the definition of MSP. After testing the algorithms ten times with different values, their GCD and time to execute were recorded. With this project I learned (never used python, mostly just arduino, c# and java) that python is an interpretive language, meaning that when you run code, it does not need to be compiled for it to function. Another thing I learned is that python gets affected easily if you use spaces or tabs, better not to mix these up. When it came to time and space efficiency, the Euclid algorithm would be the best at space and time (O(log (m/(m mod n))(m*n))). When it came to CIC and MSP, these would fluctuate. A pattern I saw when running these is that with larger numbers, MSP would be faster than CIC and vice versa. In space efficiency, CIC had (O(N)) while MSP had (O(M+N+P+T)) due to the loops of the helper function and the function itself, making it the least space efficient.

```
def Euclid(m, n): #input
        while(n != 0):
                r = m % n
                m = n
                n = r
        return m
```

```
# Find GCD with Consecutive Integer Checking
def CIC(m,n):
    t=min(m,n)#step1
    run=1
    step=2
    while run==1:
        if step==2:#step2
            if m % t==0:#step2
                step=3
            else:
                step=4 #go to step4
        if step==3:
            if n % t==0:
                run=0
            else:
                step=4 #go to step4
        if step==4:#step4
            t-=1
            step=2#step2
    return t
```

```
# Prime factorization for Middle School Procedure
def primeFactors(val):
    factors = []
    unit = 2
    while(unit != val and val != 1):
        if(val % unit == 0):
            val = val / unit
            factors.append(unit)
        else:
            unit += 1
    factors.append(int(val))
    return factors

# Find GCD with Middle School Procedure
def MSP(m,n):
    primeofm=primeFactors(m)#call prime factorization
    primeofn=primeFactors(n)#call prime factorization
    primeofmn=[]
    for i in primeofm:
        if i in (primeofm and primeofn):
            primeofmn.append(i)
    final=1
    for f in primeofmn:
        final=f*final
    return final
```

| M Val | N Val | EUC Time | CIC Time | MSP Time | GCD | Time total |
|---|---|---|---|---|---|---|
| 31415 | 14142 | 4.000000000004e-06 | 0.0030769999999999964 | 0.0005589999999999762 | 1 | 0.0092059999999999992 |
| 52665 | 955 | 5.000000000032756e-06 | 0.0002229999999999997322 | 0.0008409999999999999807 | 5 | 0.0069099999999999972 |
| 63243 | 16316 | 4.999999998107711e-06 | 0.0035499999999999972 | 0.0024959999999998317 | 1 | 0.0115150000000000164 |
| 2815 | 10785 | 3.999999996709334e-06 | 0.0006979999999998654 | 0.0002840000000000173 | 5 | 0.0062980000000000137 |
| 4935 | 3191 | 4.999999998107711e-06 | 0.0006899999999996353 | 0.0005299999999999999194 | 1 | 0.0066090000000000087 |
| 38 | 9093 | 4.000000000115023e-06 | 1.3000000000040757e-05 | 0.0001120000000001120 1 | 1 | 0.0055360000000000207 |
| 122 | 988 | 1.999999999835 4667e-06 | 3.0000000000196536e-05 | 3.0000000000196536e-05 | 2 | 0.0054410000000000251 |
| 340 | 43 | 4.999999998107711e-06 | 1.5000000000320313e-05 | 2.3000000000550358e-05 | 1 | 0.0054939999999999666 |
| 27 | 26 | 4.000000000559112e-06 | 1.1000000000064938e-05 | 1.5000000000320313e-05 | 1 | 0.0054569999999999823 |
| 41 | 28 | 2.000000000279556e-06 | 1.0000000000509601e-05 | 2.0000000000131024e-05 | 1 | 0.0052780000000000056 |