Homework 6

October 26, 2021

[1]:
```
########## PREAMBLE:
def fastPowerSmall(g,A,N):
    a = g
    b = 1
    while A>0:
        if A % 2 == 1:
            b = b * a % N
        A = A//2
        a = a*a % N
    return b

def extendedEuclideanAlgorithm(a,b):
    u = 1
    g = a
    x = 0
    y = b
    while true:
        if y == 0:
            v = (g-a*u)/b
            return [g,u,v]
        t = g%y
        q = (g-t)/y
        s = u-q*x
        u = x
        g = y
        x = s
        y = t

def findInverse(a,p):
    inverse = extendedEuclideanAlgorithm(a,p)[1] % p
    return inverse

def millerRabin(a,n):

    #first throw out the obvious cases
    if n%2 == 0 or extendedEuclideanAlgorithm(a,n)[0]!=1:
        return True
```

```
    #Next factor n-1 as 2^k m
    m = n-1
    k = 0
    while m%2 == 0 and m != 0:
        m = m//2
        k = k+1
    #Now do the test:
    a = fastPowerSmall(a,m,n)
    if a == 1:
        return False

    for i in range(0,k):
        if (a + 1) % n == 0:
            return False
        a = (a*a) % n

    #If we got this far a is not a witness
    return True

def probablyPrime(p):
    for i in range(0,20):
        a = ZZ.random_element(2,p-1)
        if millerRabin(a,p):
            return False
    return True

def findPrime(lowerBound,upperBound):
    while True:
        candidate = ZZ.random_element(lowerBound,upperBound)
        if probablyPrime(candidate):
            return candidate
```

[27]:
```
def PollardFactor(N, a=2, n=-1):
    i = 1
    while true:
        #print(i)
        p = extendedEuclideanAlgorithm(a-1,N)[0]
        if p == N and a!=2:
            print("TEST FAILED: Found GCD of N, try another value of a")
            return -1
        elif p !=1 and a!=2:
            q = N//p
            return [p,q,i]
        elif i==n:
            print("TEST FAILED: Reached upper bound without finding factors")
            return -1
```

```
        a = fastPowerSmall(a,i,N)
        i = i+1

N = 13927189
print("Factoring",N,"...")
factors = PollardFactor(N)
print(N,"=",factors[0],"*",factors[1],".  This factored in",factors[2],"steps.")

N = 168441398857
print("Factoring",N,"...")
factors = PollardFactor(N)
print(N,"=",factors[0],"*",factors[1],".  This factored in",factors[2],"steps.")

N = 47317162267924657513
print("Factoring",N,"...")
factors = PollardFactor(N)
print(N,"=",factors[0],"*",factors[1],".  This factored in",factors[2],"steps.")

N = 52309777505558628714334338842 91
print("Factoring",N,"...")
factors = PollardFactor(N)
print(N,"=",factors[0],"*",factors[1],".  This factored in",factors[2],"steps.")

N = 515459117588889238503625135159
factors = PollardFactor(N,2,200000)
```

```
Factoring 13927189 …
13927189 = 3823 * 3643 .  This factored in 15 steps.
Factoring 168441398857 …
168441398857 = 350437 * 480661 .  This factored in 54 steps.
Factoring 47317162267924657513 …
47317162267924657513 = 9740740109 * 4857655757 .  This factored in 828 steps.
Factoring 523097775055586287143343388 4291 …
5230977750555862871433433884291 = 835667525772397 * 625963985584303 .  This
factored in 9398 steps.
TEST FAILED: Reached upper bound without finding factors
```

[8]:
```
def pi(n):
    howMany = 2
    for i in range(2,(n+1)//2):
        if probablyPrime(2*i+1):
            howMany = howMany + 1
    return(howMany)

def primeNumberTheorem(n):
    return n/ln(n)
```

```python
for i in range(1,6):
    n = 10**i
    n1 = primeNumberTheorem(n)
    n2 = pi(n)
    print("Number of primes <",n,"=",n2)
    print("Prime number theorem predicts around",float(n1))
    print("ratio is",float(n2/n1))

def pi1(n):
    howMany = 0
    i = 5
    while i<=n:
        if probablyPrime(i):
            howMany = howMany + 1
        i = i + 4
    return howMany

def pi3(n):
    howMany = 1
    i = 7
    while i<=n:
        if probablyPrime(i):
            howMany = howMany + 1
        i = i+4
    return howMany

for i in range(0,6):
    n = 10**i
    print("Primes <",n,"congruent to 1 mod 4:",pi1(n))
    print("Primes <",n,"congruent to 3 mod 4:",pi3(n))
    print("Ratio is:",float(pi1(n)/pi3(n)))
```

```
Number of primes < 10 = 4
Prime number theorem predicts around 4.3429448190325175
ratio is 0.9210340371976184
Number of primes < 100 = 25
Prime number theorem predicts around 21.714724095162588
ratio is 1.151292546497023
Number of primes < 1000 = 168
Prime number theorem predicts around 144.76482730108393
ratio is 1.160502886868999
Number of primes < 10000 = 1229
Prime number theorem predicts around 1085.7362047581294
ratio is 1.1319508317158729
Number of primes < 100000 = 9592
Prime number theorem predicts around 8685.889638065035
ratio is 1.1043198105999446
```

```
Primes < 1 congruent to 1 mod 4: 0
Primes < 1 congruent to 3 mod 4: 1
Ratio is: 0.0
Primes < 10 congruent to 1 mod 4: 1
Primes < 10 congruent to 3 mod 4: 2
Ratio is: 0.5
Primes < 100 congruent to 1 mod 4: 11
Primes < 100 congruent to 3 mod 4: 13
Ratio is: 0.8461538461538461
Primes < 1000 congruent to 1 mod 4: 80
Primes < 1000 congruent to 3 mod 4: 87
Ratio is: 0.9195402298850575
Primes < 10000 congruent to 1 mod 4: 609
Primes < 10000 congruent to 3 mod 4: 619
Ratio is: 0.9838449111470113
Primes < 100000 congruent to 1 mod 4: 4783
Primes < 100000 congruent to 3 mod 4: 4808
Ratio is: 0.9948003327787022
```

[0]:

```
```

[0]:

```
```