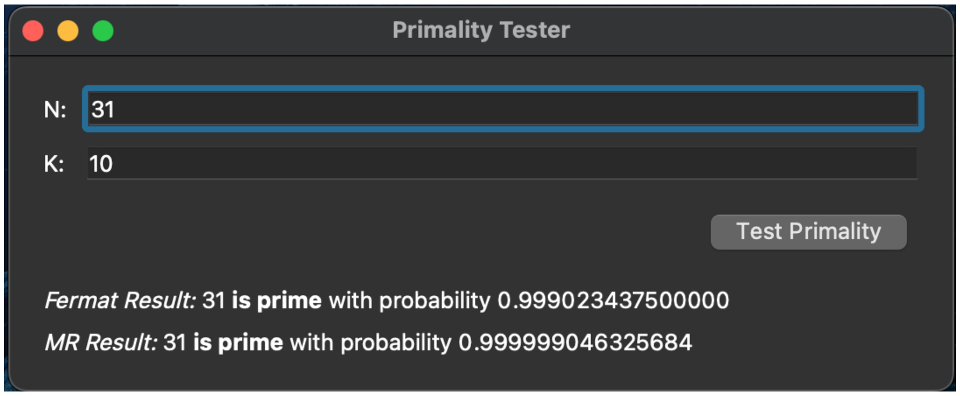
Project 1 - Primality Tests

Austin McKamey
CS312
Sep 8, 2022

1.



N: 31

K: 10

Fermat Result: 31 **is prime** with probability 0.999023437500000

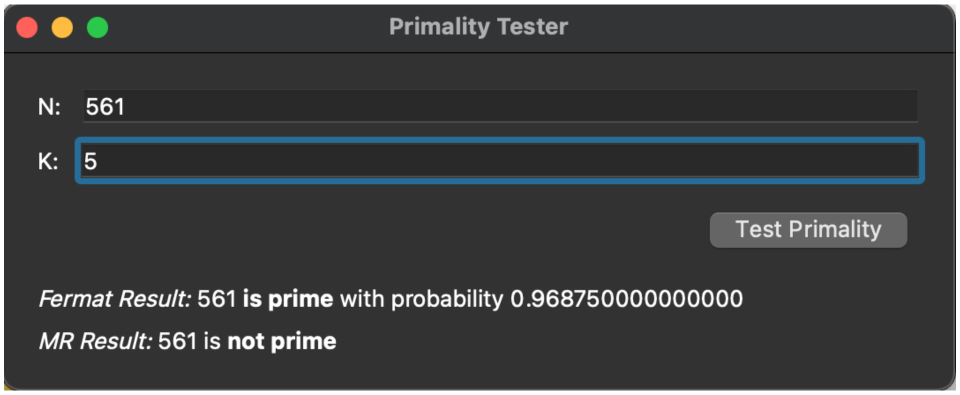
MR Result: 31 **is prime** with probability 0.999999046325684

2.

```
def mod_exp(x, y, N):
    # This function conducts modular exponentiation, and returns x^y mod N
    # Space complexity - O(nlog(n))
    # Time complexity - O(n^3)
    if y == 0:                  # O(n)     # O(n)
        return 1
    z = mod_exp(x, y//2, N)     # O(n^2)
    if y % 2 == 0:              # c
        return (z ** 2) % N     # O(n^2)
    else:
        return (x * z ** 2) % N # O(n^2)


def fprobability(k):
    # This function determines the probability that N is correctly classified
    # as prime by the Fermat method based off of the iterations k
    # Space complexity - O(n)
    # Time complexity - O(n^2)
    return 1.0 - 1 / (2 ** k)   # O(n^2)


def mprobability(k):
    # This function determines the probability that N is correctly classified
    # as prime by the Miller-Rabin method based off of the iterations k
    # Space complexity - O(n)
    # Time complexity - O(n^2)
    return 1.0 - 1 / (4 ** k)   # O(n^2)
```

```python
def fermat(N, k):
    # This function conducts the Fermat primality test k times, using the mod_exp()
    # method to determine whether N is prime or composite
    # Space complexity - O(n^2log(n))
    # Time complexity - O(n^3log(n))
    for _ in range(k):                 # O(n)
        a = random.randint(2, N - 1) # O(log(n))
        x = mod_exp(a, N-1, N)       # O(n^3)
        if x != 1:                   # O(n)
            return 'composite'
    return 'prime'


def miller_rabin(N, k):
    # This function conducts the Miller-Rabin primality test k times, using the
    # mod_exp() method to determine whether N is prime or composite
    # Space complexity - O(n^2log(n))
    # Time complexity - O(n^3log(n))
    d = N - 1                          # O(1)
    while d % 2 == 0:                  # O(log(n))
        d //= 2
    for _ in range(k):                 # O(n)
        a = random.randint(2, N - 1) # O(log(n))
        x = mod_exp(a, d, N)         # O(n^3)
        if x == 1 or x == N - 1:     # O(n)
            return 'prime'
        while d != N - 1:            # O(n^2log(n))
            x = (x * x) % N
            d *= 2
            if x == 1:               # O(n)
                return 'composite'
            if x == N - 1:           # O(n)
                return 'prime'
        return 'composite'
```

2d. In trying different numbers, it was very rare for the two tests to disagree, especially if the value ‘K’ was higher than 5. However, there were two numbers I found that consistently caused disagreement within fifteen clicks of the test. The first of these numbers was unsurprisingly 561, the first Carmichael number. The second was 221, which happens to have only four factors. For both of these numbers, the disagreement came from the Fermat test incorrectly identifying them as prime, due to the small amount of values it was able to test and some unlucky guesses.



N: 561

K: 5

Fermat Result: 561 **is prime** with probability 0.968750000000000

MR Result: 561 is **not prime**

3. Time complexity:
    a. Conditional return statements - O(n)
    b. Multiplication, division, exponents - O(n^2)
    c. Random integer generator - O(log(n))
    d. While loops halving value - O(log(n))
    e. While loops squaring value - O(n^2log(n))
    f. Recursive call - O(n^2)
    g. Mod_exp - O(n^3)
        i. mod_ exp - (a + 2b + f) * a = O(n^3)
        ii. fprobability - b = O(n^2)
        iii. mprobability - b = O(n^2)
        iv. fermat - 2a + c + g = O(n^3log(n))
        v. miller_rabin - 4a + c + d + e + g = O(n^3log(n))

   Space complexity:
    a. Input variables/return value - O(n)
    b. Recursive call - O(nlog(n))
    c. For loop - O(n)
        i. mod_exp - 5a + b = O(nlog(n))
        ii. fprobability - 2a = O(n)
        iii. mprobability - 2a = O(n)
        iv. fermat - 3a + c = O(n^2log(n))
        v. miller_rabin - 4a + c = O(n^2log(n))

   The Fermat and Miller-Rabin tests add O(nlog(n)) because they both call mod_exp
4. Of the two equations I used, the Fermat was given in the textbook. It states that if N is prime, then at most half the values of a<N will result in a^(N-1) ≡ 1 mod N, as a function of k. This allows us to calculate the probability of correctness by subtracting 1/(2^k), which is the probability for error, from 1. The second equation for the Miller-Rabin test is similar, and states that if N is composite, at least three quarters of the possible choices for ‘a’ will fail the Fermat test or result in something other than -1 after a sequence of 1’s. This gives us the ability to calculate the probability by subtracting 1/(4^k) from 1.