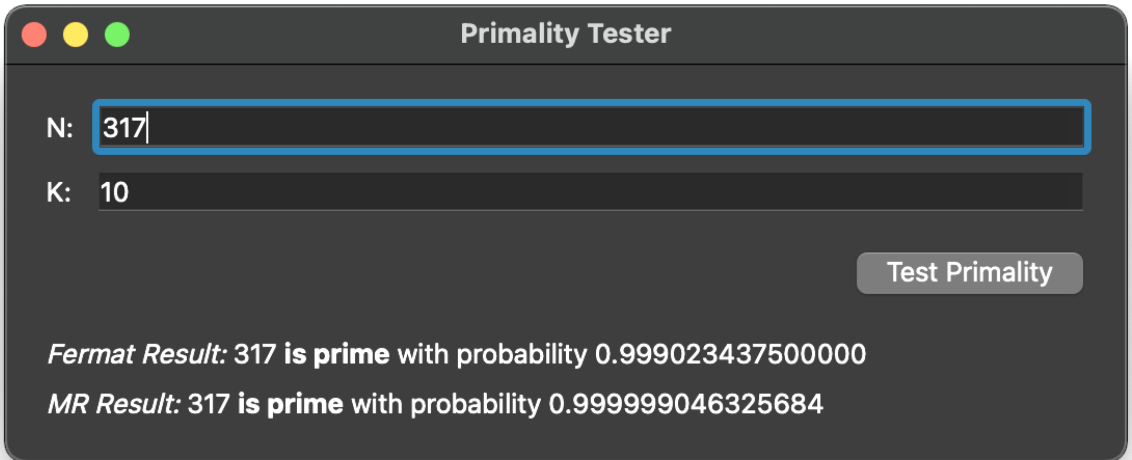
Project 1 - Fermat's Primality Tester

Example of Program Being Run



Mod_Exp Function

```
def mod_exp(x, y, N):                 #                                    SPACE COMPLEXITY O(3n) {Int} x y N
    # TOTAL TIME COMPLEXITY  O(n^3)
    # TOTAL SPACE COMPLEXITY O(5n)
    if y == 0:
        return 1                      #                                    SPACE COMPLEXITY O(1n) {Int} Return value
    z = mod_exp(x, y // 2, N)         # RECURSIVE CALL O(log2(n))          SPACE COMPLEXITY O(1n) {Int} Z
    if y % 2 == 0:
        return (z ** 2) % N           # MULTIPLICATION O(n^2)              SPACE COMPLEXITY O(1n) {Int} Return value
    else:
        return (x * (z ** 2)) % N     # MULTIPLICATION O(2n^2)             SPACE COMPLEXITY O(1n) {Int} Return value
```

Probability Function

```
def f_probability(k):
    # TOTAL TIME COMPLEXITY  O(2n^2 * 2^k)
    # TOTAL SPACE COMPLEXITY O(1)
    return 1 - (1 / (2 ** k))         # EXPONENTIAL, Div, Sub O(2n^2 * 2^k)  SPACE COMPLEXITY O(1) {Int} Return Value


def m_probability(k):
    # TOTAL TIME COMPLEXITY  O(2n * 2^k)
    # TOTAL SPACE COMPLEXITY O(1)
    return 1 - (4 ** -k)              # EXPONENTIAL, Sub O(2n*2^k)           SPACE COMPLEXITY O(1) {Int} Return Value
```

Fermat's

```
def fermat(N, k):                  #                                       SPACE COMPLEXITY O(2) {Int} k N
    # TOTAL TIME COMPLEXITY  O(n^4 + 1)  -> O(n^4)
    # TOTAL SPACE COMPLEXITY O(6n + 4)
    M = N - 1                      #                                       SPACE COMPLEXITY O(1) {Int} M
    for i in range(0, k):          # FOR O(n) where n is k
        y = random.randint(1, M)   # randint O(1)                          SPACE COMPLEXITY O(1n) {int} y
        if mod_exp(y, M, N) != 1:  # Mod_Exp O(n^3)                        SPACE COMPLEXITY O(5n)
            return 'composite'     #                                       SPACE COMPLEXITY O(1) {String} Return Value
    return 'prime'                 #                                       SPACE COMPLEXITY O(1) {String} Return Value
```

Miller-Rabin

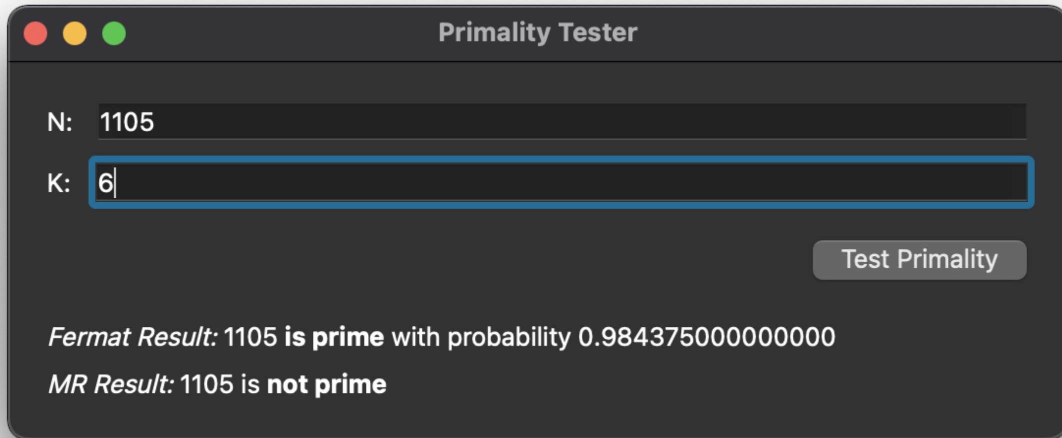
```
def miller_rabin(N, k):                     #                                       SPACE COMPLEXITY O(2) {Int} k N
    # TOTAL TIME COMPLEXITY  O(2n^4 + n^2 + 2N + 2)  -> O(n^4)
    # TOTAL SPACE COMPLEXITY O(6n + 4)
    squares = 0                             #                                       SPACE COMPLEXITY O(1) {Int} squares
    M = N - 1                               #                                       SPACE COMPLEXITY O(1) {Int} M

    # Determine how many times the MR algorithm should loop by squaring N-1 through a bit shift until it is odd
    # TIME COMPLEXITY  O(n+2)
    # SPACE COMPLEXITY N/A
    while M % 2 == 0:                       # While           O(n)
        M >>= 1                             # Bit Shift       O(1)
        squares += 1                        # Basic Operation O(1)

    # MR Algorithm
    # TIME COMPLEXITY  O(2n^3 * n)
    # SPACE COMPLEXITY O(5n+2)
    def test_evaluation(X):
        if mod_exp(X, M, N) == 1:           # Mod_Exp O(n^3)                        SPACE COMPLEXITY O(5n)
            return False                    #                                       SPACE COMPLEXITY O(1) {Bool} Return
        for j in range(0, squares):         # For O(n) where n is squares           SPACE COMPLEXITY O(1) {Int} j
            if mod_exp(X, 2 ** j * M, N) == N - 1:  # Mod_Exp O(n^3)                SPACE COMPLEXITY O(5n)
                return False                #                                       SPACE COMPLEXITY O(1) {Bool} Return
        return True                         #                                       SPACE COMPLEXITY O(1) {Bool} Return

    # Accuracy Loop
    # TIME COMPLEXITY O(2n^4 * n^2 + n)
    # SPACE COMPLEXITY O(6n + 4)
    for i in range(0, k):                   # For O(n) where n is k                 SPACE COMPLEXITY O(1) {Int} i
        a = random.randint(2, N)            # randint O(1)                          SPACE COMPLEXITY O(1) {Int} a
        if test_evaluation(a):              # Test_Evaluation O(2n^3 * n)           SPACE COMPLEXITY O(5n + 2)
            return 'composite'              #                                       SPACE COMPLEXITY O(1) {Str} Return
    return 'prime'                          #                                       SPACE COMPLEXITY O(1) {Str} Return
```

2.) d.) One of the things that I tried in order to find a number was exploit the weakness in the Fermat result by pitting it against a pseudoprime/Carmichael number, although with a high enough K it is unlikely that one of these numbers completely gets past the program it's still possible to introduce a discrepancy in the results.



3.) Although it was described in the code quite explicitly the overall time complexity for both the Fermat and the Miller-Rabin algorithms is O(n^4) *[After selecting the most impactful part of the polynomial -- O(n^4+1) and O(2n^4+n^2+2n+2) respectively]* This is because both rely quite heavily on the Mod_Exp [O(n^3)] function existing within a loop to guarantee an accurate result. Logically the two probability functions are both computed in finite time O(1), because we are calculating with a very fine granularity we get O(2n^2 * 2^k) for the fermat probability and O(2n * 2^k) for the MR function. Both could be calculated to the exact same O(n) however by swapping the division in the fermat calculation for the negative exponent in the MR calculation we can bring the time down to a linear O(n) time.
Space Complexity is largely inconsequential in a program of this size but it is mostly built on the complexity of the mod_exp function as it is recursive and leaves a lot of memory on the stack as it drills down further into the recursive element.

4.) The probability of the fermat problem is calculated with the following formula

$$1 - 1/2^k$$

The probability of the MR problem is calculated by the following formula

$$1 - 4^{-k}$$

Both provide very small numbers which can be subtracted from 1 to determine the chance that we have determined a correct number. Because of its exponential nature the MR is more reliable and faster to determine than the Fermat while k < ~50 but it becomes negligible after that.