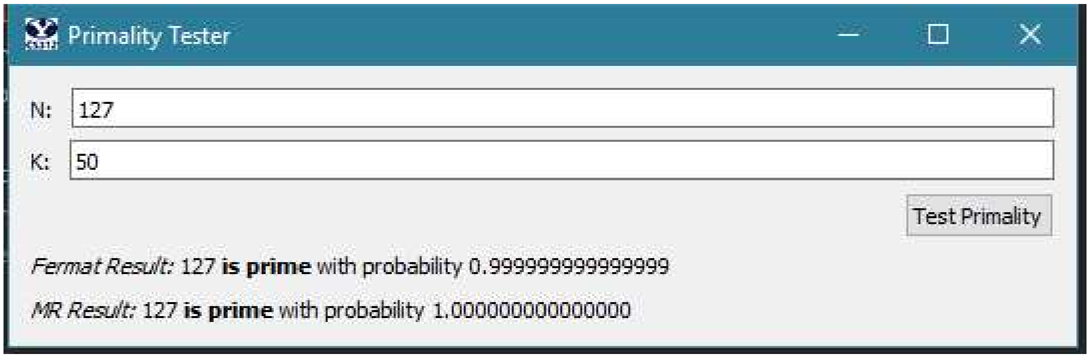
Taylor Whitlock

Project 1



****
****

SOURCE CODE

****
****

```
import random


def prime_test(N, k):

	# This is the main function connected to the Test button. You don't need to touch it.

	return run_fermat(N,k), run_miller_rabin(N,k)




def mod_exp(x, y, N):

	if y == 0:

		return 1

	z = mod_exp(x, y // 2, N)

	return (z ** 2) % N if y % 2 == 0 else (((z ** 2) % N) * x) % N




def fprobability(k):
```

```
    return 1 - (1/(2**k))


def mprobability(k):
    return 1 - (1/(4**k))


def run_fermat(N,k):
    x_vals = []
    for i in range(k):
        x = random.randint(2, N-1)      # O(log(n)) time according to the
internet
        while x in x_vals:                          # check to avoid
duplicate x values
            x = random.randint(2, N-1)
        x_vals.append(x)
        result = mod_exp(x, N-1, N)
        if result != 1:
            return "composite"
    return 'prime'


def run_miller_rabin(N,k):
    x_vals = []
    for i in range(k):
        y = N - 1
        x = random.randint(2, N-1)
        while x in x_vals:
            x = random.randint(2, N-1)
        x_vals.append(x)
        result = mod_exp(x, N-1, N)
        if result != 1:
            return 'composite'
```

```
            while y % 2 == 0:
                  result = mod_exp(x, y, N)
                  if result == 1:
                        y //= 2
                  elif result == N - 1:
                        break
                  else:
                        return 'composite'
      return 'prime'
```

****
****

SOURCE CODE

****
****

Part 3:

Mod_exp: time complexity is O(n^3); multiplication of n-bit numbers is n^2, and there will be at most n recursive calls for a total of n^3.

Fermat: k iterations will add a factor of k to the total time, and mod_exp is O(n^3), bringing the total time complexity to O(k*(n^3)). In the worst case, the check for a valid x_val will add a factor of n to the total time.

Fermat_probability: It’s basically O(1) when compared to the rest of the program, but it would be k multiplications of 2, because it is 1-(1/(2^k))

Miller_rabin: Again, k iterations will add a factor of k, the calls to mod_exp will be O(n^3), and though the two calls would make O(2(n^3)), the constant coefficient can be dropped. The division of the exponent (which starts as n-1) will remove 1 bit each time it divides, adding another factor of n (the -1 can be dropped), so the algorithm ends up being O(k*(n^4)) in total.

Miller_rabin_probability: Similar to the ferma_probability, there will be k multiplications of 4 since it is 1-(1/(4^k)), but it is practically O(1) compared to the rest of the program, since k is pretty much a constant factor added to the total time complexity, so it can be dropped.

Part 4:

The book gives the probability of the fermat test being incorrect as 1/(2^k), so the probability that the fermat test is accurate for a large k value is 1-(1/(2^k)).

The book gives the probability of the miller-rabin test being CORRECT as 3/4 chance, so the inaccuracy is 1/(4^k), and the correctness is 1-(1/(4^k)).