1. Modular Exponentiation

- Recursive calls: Since y is divided by 2 in each recursive call, the recursion depth is O(log y).
- Work per Call: Each step involves at most O(1) multiplications and modulo operations
- Multiplication Cost: If x and N are n-bit numbers, then multiplication of two n-bit numbers using the standard method takes O(n²) time.

Total: O(n^2 * n)= O(n^3)
Space: O(n)(n bit integers)

2. Fermat

- it takes k amount of calls to run the function
- it takes O(n^3) to run per call

Total: k * O(n^3) = O(kn^3)
Space: O(n)(dominated by mode_exp)

3. Miller-Rabin

- **Decomposition:** this step involves repeatedly dividing N-1 by 2 which is O(n)
- **Each iteration of the test:**
    - **Modular exponentiation:** O(n^3)
    - Squaring steps takes O(n^3)
    - Total for one iteration: O(n^3)

Total: k * O(n^3) = O(kn^3)
Space: O(n)

4. Generate_large_prime

- Expected times of call is O(n)
- Work per call: each step takes constant with exception of Miller_Rabin which is O(kn^3)

Total: O(n) * O(kn^3) = O(kn^4)
Space: O(n)(dominated by Miller-Rabin)

5. Extended Euclid

- it take O(n) times of call
- Work per call: division in itself is O(n^2), so it is O(n^2) per call

Total: O(n) * O(n^2) = O(n^3)
Space: O(n)

6. Generate_key_pairs

- **Miller-rabin** is O(kn^4)
- Computing N is multiplication and it is O(n^2)’
- **Finde** runs loop for n times but it involves multiplication to find phi _n so the time complexity is O(n^2)
- **Extended Euclid** ia O(N^3)

Total : O(kn^4) * O(n^2) * O(n^3) = O(kn^4)

Space Complexity

- **Multiplication of N:** it takes two n bit numbers p and q and it is 2nbit number: O(n) + O(n) + O(n^2) = O(n)
- **Miller-Rabin:** it takes O(n) space to store intermediate values
- **Extented Algorithm** runs O(N) space

Total : O(n)

Discussion of probability

1. **Fermat**

   We ust the small theorem of Fermat. We try to find the prime numbers. Inordder to do it we need to test these numbers with the “a” number. However, there is a 50% probability that a will pass the test. That is why we run it k times which makes it: ½^k
2. **Miller-Rabin**

   Miller-Rabon is even stronger. There is 75% that “a” will pass the test. And we also run it k times. That is why the probability is : ¼^k