```
 1 import random
 2
 3
 4 def prime_test(N, k):
 5     return fermat(N,k), miller_rabin(N,k)
 6
 7
 8 # Time complexity 0(n^3)
 9 # Space complexity 0(n^2)
10 # We use a recursive algorithm with intermediate
   computations modulo N to make sure number doesn't
   grow too large
11 def mod_exp(x, y, N):
12     # base case
13     if y == 0:
14         return 1
15
16     # recursive call
17     z = mod_exp(x, y // 2, N)
18
19
20     if y % 2 == 0:
21         #even y's
22         return z**2 % N
23
24     else:
25         # odd y's
26         return x * (z**2) % N
27
28
29 def fprobability(k):
30     # the error probability is 1 /2 **k, so we
   subtract from one to get the success probability
31     return 1 - (1 / 2**k)
32
33
34 def mprobability(k):
35     # the error probability is 1 /4 **k, so we
   subtract from one to get the success probability
36     return 1 - (1 / 4 ** k)
37
```

```
38 # Time complexity 0(n^3 * k)
39 # Space complexity 0(n^2)
40 def fermat(N,k):
41     # we loop through k times with a different base
    and call mod_exp(). We return composite
   immediately if the return value of mod_exp != 1
42     for x in range(k):
43         randBase = random.randint(1, N-1)
44         if mod_exp(randBase, N-1, N) != 1:
45             return 'composite'
46 
47     # if get to the end of the loop, we conclude
   that N is prime
48     return 'prime'
49 
50 # Time complexity 0(n^4 * k)
51 # Space complexity
52 def miller_rabin(N,k):
53     #if its even, return composite
54     if N%2 == 0 and N != 2:
55         return 'composite'
56 
57    # for k times, perform the miller rabin test, if
    a test returns composite, return composite
58     for x in range(k):
59         a = random.randint(1, N-1)
60         solution = miller_rabin_helper(a, N)
61 
62         if solution == 'composite':
63             return 'composite'
64 
65     #return prime if all tests pass
66     return 'prime'
67 
68 # Time complexity 0(n^4)
69 def miller_rabin_helper(a, N):
70     power = N-1
71 
72     # while the power is even, call mod_exp
73     while power % 2 == 0:
74         power_output = mod_exp(a, power, N)
```

```
75
76          # if a ** power mod N is -1, return prime
77          if power_output == N-1:
78              return 'prime'
79
80          # if a ** power mod N is anything but -1
    or 1, return prime
81          elif power_output != 1:
82              return 'composite'
83
84          #update power
85          power = power / 2
86
87      # the final test if the final power is odd and
     we haven't returned yet
88      power_output = mod_exp(a, power, N)
89
90      # if a ** power mod N is is -1 or 1, return
    prime
91      if power_output == 1 or power_output == N-1:
92          return 'prime'
93      else:
94          return 'composite'
95
96
```