Primality Test

All programs are attached in the end of the report.

Question 1:

We implemented the trial division algorithm and use it to find primes in the given ranges. In $[1900, 1999]$ the first prime is 1901. There is no prime in the range $[1294268500, 1294268700]$.

Modular Multiplication with Large Numbers:

Before proceeding to other questions, it's necessary to tackle the issue of integer overflow when we consider the expression of the form $ab \bmod N$, where a, b and N are limited to 10^{10}. Here we show one way to circumvent the issue by decomposing a and b.

To begin with, we use the standard division algorithm to decompose
$a = 10^7 a_1 + a_2$ and $b = 10^7 b_1 + b_2$, where a_2 and b_2 are small than 10^7. Moreover by the upper bound of a and b we also have a_1 and b_1 are smaller than 10^3.

Now we do the modular multiplication with standard properties of modular arithmetics. We compute

$$\begin{aligned}
ab \bmod N &= (10^7 a_1 + a_2)(10^7 b_1 + b_2) \bmod N \\
&= 10^{14} a_1 b_1 + 10^7 b_2 a_1 + 10^7 a_2 b1 + a_2 b_2 \bmod N \\
&= ((10^{14} \bmod N) a_1 \bmod N) b_1 \bmod N \qquad (1) \\
&+ (10^7 b_2 \bmod N) a_1 \bmod N \\
&+ (10^7 a_2 \bmod N) b_1 \bmod N \\
&+ a_2 b_2 \bmod N \qquad (2)
\end{aligned}$$

Note that we completely avoid the issue of integer overflow in this way because, for example, in 1, $10^{14} \bmod N$ evaluates to a number smaller 10^{10}, and hence $((10^{14} \bmod N)a_1 \bmod N)$ can be done with the exact arithmetic, which multiplied with b_1 can also be done exactly modular N.

Note also that the complexity of this algorithm is $O(1)$.

Question 2:

To carry out the Fermat test, we implement an algorithm for calculating

$$a^b \bmod N$$

First we convert b into its binary representation, namely $b = b_0 ... + b_n 2^n$, where b_i's are zeros and ones uniquely determined by the standard procedure of binary expansion. Then

$$a^b \bmod N = a^{b_0 + b_1 2 + b_2 2^2 ... + b_n 2^n} \bmod N = a^{b_0} a^{b_1 2} ... a^{b_n 2^n} \bmod N$$

Since b_i's are zeros and ones, it suffices to calculate a^{2^i} for i in $[1, n]$, which can be done via the multiplication algorithm 2 and repeated squaring.

Now we analyze the time complexity of the Fermat test. We take $b = N - 1$:
1. Binary Expansion takes $O(\log N)$ operations.
2. Calculating a^{2^i} for i in $[1, n]$ via repeated squaring takes $O(\log N)$ operations.
3. $a^{b_0} a^{b_1 2} ... a^{b_n 2^n} \bmod N$ also takes $O(\log N)$ operations.

Thus the overall complexity is $O(\log N)$.

Question 3:

We use the modular exponentiation algorithm to conduct the Fermat test. In particular, the first Fermat pesudoprime base 2 and the first absolute Fermat pesudoprime are 341 and 561 respectively. Note that we may speed up the process of checking the absolute Fermat pesudoprimes by factorizing the Fermat pesudoprimes base 2 and perform the Fermat test on factors of these pesudoprimes. This method is valid in light of the Chinese Remainder Theorem.

We need $a = 2, 3, 5, 7, 11$ to determine the primality of an integer N.

Question 4:

We modify our previous program to carry out the Euler test. To compute the Jacobi symbol, we exploit properties of the Jacobi symbol recursively. The program for computing the Jacobi symbol produces the following:

$(\frac{123}{42441}) = 0$ and $(\frac{123}{42443}) = 1$

We implemented the Euler test and apply it to integers up to 10^6 to find the Euler pesudoprimes base 2 and the absolute Euler pesudprimes. The first three Euler pesudoprimes base 2 are 561, 1105 and 1729, and there is no absolute Euler pesudoprime in $[1, 10^6]$. Based on this evidence, we may conjecture that there does not exist any absolute Euler pesudoprime at all.

We need $a = 2, 3, 5, 7, 11, 13$ to determine the primality of a non-absolute Euler pesudoprime(hence any integer) in this range.

Note that, in terms of time complexity, due to our implementation, the modular exponentiation part of the Euler test takes almost the same amount of time as the Fermat test. However, the additional cost of determining the Jacobi symbol causes this method to be less computationally favorable compared to the Fermat test.

Question 5:

We make use of our modular exponentiation algorithm again for the Miller-Rabin test. The first three strong pesudoprime base 2 are $2047, 3277, 4033$, and again there is no absolute strong pesudoprime in the given range.

In terms of complexity, the worst case complexity of the Miller-Rabin test, based on the highest power of 2 in $N - 1$, is also $O(\log N)$. But in practice it can be much faster than this.

Surprisingly, only two values of a, namely $a = 2$ and 3, are needed for determining the character of an integer N in the given range.

Question 6:

We use the program developed in Question 5 to enumerate the number of the Fermat/Euler/strong pesudoprimes in $[10^k, 10^k + 10^5]$ for $5 \leq k \leq 9$. The enumeration produces the following result: the number of the strong pesudoprimes base 2 and base 3 in $[10^5, 10^5 + 10^5]$ are 3 and 7, and there is no strong pesudoprime both base 2 and base 3 at all in the given range.

We note also that, given a base a, a strong pesudoprime is an Euler pesudoprime, and an euler pesudoprime is a Fermat pesudoprime. For instance, 130561 is a strong/Euler/Fermat pesudoprime at the same time. Hence the number of strong pesudoprimes is less than the number of Euler pesudoprimes in the given range, and this relation also holds for the Euler pesudoprimes and the Fermat pesudoprimes.

Question 7:

In light of previous results, we develop an efficient algorithm for determining the primality of a given integer N.
1. if N is in the range $[0, 10^6]$ or $[10^k, 10^k + 10^5]$ for $6 \leq k \leq 9$, we apply the strong test base 2 and 3 to N, since in these ranges $a = 2$ and 3 completely determine the character of N.
2. Otherwise, we apply the strong test base $2, 3, 5, 7$ and 11 to N. If N passes five rounds of test we use the trial division to determine the character of N.

We implement this algorithm and experiment it on 10000 random integers. On the author's machine, the algorithm takes 0.517 second. For comparison, the trial division algorithm takes 1.177 seconds. Indeed, by taking larger sample sizes, say 100000 and 10000000, we find that, in general, our algorithm saves approximately half amount of time for primality test compared to the trial division.

Question 8:

Let A be the event that N is prime, B be the event that N passes t rounds of the strong test with randomly chosen a. We investigate $\mathbb{P}(A|B)$. The probability that if N passes t rounds of the strong test with randomly chosen a, then N is in fact composite is just $1 - \mathbb{P}(A|B)$.

Let s be the number of primes between 2^{k-1} and 2^k. By the Prime Number theorem

$$\begin{aligned} s &= \frac{2^k}{k\log 2} - \frac{2^{k-1}}{(k-1)\log 2} \\ &= \frac{2^k}{\log 2} * \frac{k-2}{2k(k-1)} \\ &\sim \frac{2^{k-1}}{k\log 2}, \text{ where } \sim \text{ denotes 'asymptotically'.} \end{aligned}$$

Hence $\mathbb{P}(A) \sim \frac{2}{k\log 2}$, since there are 2^{k-2} k-bit even integers.

Now we apply the Bayes' formula.

$$\mathbb{P}(A|B) = \frac{\mathbb{P}(B|A)\mathbb{P}(A)}{\mathbb{P}(B|A)\mathbb{P}(A) + \mathbb{P}(B|A^c)\mathbb{P}(A^c)}$$

We also know $\mathbb{P}(B|A)$ is 1. It remains to find $\mathbb{P}(B|A^c)$. Let C be the event that N passes the strong test with a randomly chosen a. Then $\mathbb{P}(B|A^c) = (\mathbb{P}(C|A^c))^t$. ***We assume that*** $\mathbb{P}(C|A^c)$ ***is independent of*** k.

To find $\mathbb{P}(C|A^c)$, we consider a given composite M. We calculate the ratio of the number of bases for which M passes the test to M. We experiment on a sample of M's and find a general bound on this ratio, which can be expected to be a bound on $\mathbb{P}(C|A^c)$. We choose the sample space of M to be $[1000, 2000]$.

The upper bound we find is 0.2374. For safety we proceed by assuming the upper bound is 0.3. Then

$$\begin{aligned}
\mathbb{P}(A^c|B) &= 1 - \frac{\mathbb{P}(B|A)\mathbb{P}(A)}{\mathbb{P}(B|A)\mathbb{P}(A) + \mathbb{P}(B|A^c)\mathbb{P}(A^c)} \\
&\leq 1 - \frac{\frac{2}{k\log 2}}{\frac{2}{k\log 2} + 0.3^t(1 - \frac{2}{k\log 2})} \\
&= 1 - \frac{2}{2 + 0.3^t(k\log 2 - 2)}
\end{aligned}$$

We see that as t tends to ∞, the desired probability converges to 0.

Source Code

```
function x = TrialDiv(n)
x = 1;
k = 2;
while k≤sqrt(n)
r = Mod(n,k); % Mod stands for modulo
if r == 0
x = 0; % x is composite with a factor = k
break
end
k = k + 1;
end
```

```
function x = strongTest(a,N)
h = highestPower(2,N-1);
if h == 0
x = 0;
else
s = (N-1)/2^h;

A = [powerMod(a,s,N)];
k = Mod(10^14,N);
for i = 0:h-1
A = [A,productMod(A(length(A)),A(length(A)),N,k)];
end

if length(A) == 1
if A(1) == 1
x = 1;
else
x = 0;
end
else
y = 0;
for i = 1:length(A)
if A(i) == N-1
y = 1;
break
end
end
if A(1) == 1 || y == 1
x = 1;
else
x = 0;
end
end
end
end
```

```
function A = strongPPCount(a,k)
A = 0;
for i = 0:10^5
if strongTest(a,10^k+i) == 1
if TrialDiv(10^k+i) == 0
A = A+1;
end
end
end
end
```

```
function p = productMod(a,b,N,k)% k = 10^14 mod N
[a1,a2] = decompose(a);% n = 10^7 * a + b
[b1,b2] = decompose(b);


f = Mod(k*a1,N);%first term
f = Mod(f*a2,N);

s = Mod(10^7*b2,N);%second term
s = Mod(s*a1,N);


t = Mod(10^7*a2,N);%third term
t = Mod(t*b1,N);

fo = Mod(a2*b2,N);%fourth term

p = Mod(f+s+t+fo,N);
end
```

```
function x = primalityTest(N)
x = 0;%x is initially composite
if ...
    (0≤N≤10^6)||(10^6≤N≤10^6+10^5)||(10^7≤N≤10^7+10^5)||(10^7≤N≤10^7+10^5)||(10^8≤N≤10
if (strongTest(2,N)==1)&&(strongTest(3,N) == 1)
x = 1;
end
else
if (strongTest(2,N) == ...
    1)&&(strongTest(3,N)==1)&&(strongTest(5,N)==1)&&(strongTest(7,N)==1)&&(strongTest(
    == 1)
x = TrialDiv(N);
end
end
```

```
function x = powerMod(a,b,N) %% this method aims to compute ...
    a^b mod N, accelarated by binary operations
```

```
k = Mod(10^14,N);
b2 = base2(b);
x = 1;
A = [Mod(a,N)];
for i = 1:(length(b2)-1)
A = [productMod(A(1),A(1),N,k),A]; %A = [a^2^m,a^2^m-1 ... ...
    a^2, a] mod N
end

for i = 1:length(b2)
if b2(i) == 1
x = productMod(A(i),x,N,k);
else
x = x;
end
end
end
```

```
function r = Mod(n,k) % n mod k
r = (n - k*floor(n/k));
end
```

```
function J = jacobi(M,N)%conpute (M/N)
M = Mod(M,N);
if M == 0
J = 0;
elseif M == 1
J = 1;
elseif gcd(M,N) ≠ 1
J = 0;
elseif Mod(M,2) == 0
h = highestPower(2,M);
J = (-1)^(h*(N^2-1)/8)*jacobi(M/2^h,N);%make M,N coprime
else
J = (-1)^((M-1)*(N-1)/4)*jacobi(N,M);% flip the symbol
end
end
```

```

function k = highestPower(d,n)%for d dividing n, this gives ...
    the largest s s.t. d^s divides n)
for i = 1:n
r = Mod(n,d^i);
if r ≠ 0
k = i-1;
break
end
end
end
```

```
function A = fermatPPCount(a,k)
A = 0;
for i = 0:10^5
if fermatPP(a,10^k+i) == 1
if TrialDiv(10^k+i) == 0
A = A+1;
end
end
end
end
```

```

function x = fermatPP(a,N) % whether N is a fermat number ...
    base a
p = powerMod(a,N-1,N);
if p == 1
x = 1;
else
x = 0;
end
end
```

```
function x = eulerTest(a,N)
t = powerMod(a,(N-1)/2,N);
if t == N - 1
t = -1;
end
if (t == 1 || t == -1)&&(t == jacobi(a,N))
x = 1;
else
x = 0;
end
end
```

```
function A = eulerPPCount(a,k)
A = 0;
for i = 0:10^5
if eulerTest(a,10^k+i) == 1
if TrialDiv(10^k+i) == 0
A = A+1;
end
end
end
end
```

```matlab
function [a,b] = decompose(n) % n = 10^7 * a + b
a = floor(n/10^7);
b = n - 10^7*a;
end
```

```matlab
function L = base2(n)
L = [1];
k = floor(log2(n)); % k is the highest power of 2
r = n - 2^k;
for i = k-1:-1:0
if r≥2^i
L = [L,1];
r = r - 2^i;
else
L = [L,0];
end
end
```