King Saud University
College of Computer and Information Sciences
Computer Science Department
(CSC 281)

<table>
<tr><th colspan="3">Group #5</th></tr>
<tr><th>Name</th><th>ID</th><th>Section</th></tr>
<tr><td>(Leader) Jomanah Almuhanna</td><td>442200133</td><td rowspan="4">41201</td></tr>
<tr><td>Sarah Alajlan</td><td>442202314</td></tr>
<tr><td>Leena Almeshal</td><td>442200408</td></tr>
<tr><td>Majd Alqahtani</td><td>442202282</td></tr>
</table>

Methods implementation and Sample run

Method1: Trial division

•Pseudocode:

Algorithm *TrialDevision(n)*

Input: ***n*** an integer to test for primality

Output: boolean value

limit ← ceil the sqrt of n

isPrime ← true

loop for *i* ← 2 **to** limit **do**

  if *n* mod i = 0 **then**

    isPrime ← false

  end if

end loop

if isPrime = false **then**

  loop for i ← 1 to *n* **do**

    if n mod i = 0

     print i

    end if

  end loop

 return isPrime

end if

else return isPrime

•Java Implementation:

```
public static boolean TrialDevision(int n) {

int limit = ((int)Math.ceil(Math.sqrt(n)));

boolean isprime = true;

for(int i = 2 ; i <= limit ; i++)
if(n%i == 0)
isprime = false;
```

```
if(isprime == false) {

        for (int i = 1; i <= n; i++)
        {
         if(n%i == 0)
             System.out.print(i+ ",");

        }

    return isprime;}

        else
            return isprime;

}
```

•Explanation:

We set the limit which is the square root of n and initialize a Boolean attribute ‘isPrime’,then a loop starting from 2 to the limit we calculated, we make sure it is not an even number because no even number is prime.

•Sample run:

Enter a value to test for primality:

341

Factors: 1,11,31,341,

Composite

- - - - - - - - - - - - - - - - - - - -

Enter a value to test for primality:

561

Factors: 1,3,11,17,33,51,187,561,

Composite

- - - - - - - - - - - - - - - - - - - -

Enter a value to test for primality:

1729

Factors: 1,7,13,19,91,133,247,1729,

Composite

- - - - - - - - - - - - - - - - - - - -

Enter a value to test for primality:

19973

probably prime

- - - - - - - - - - - - - - - - - - - -

Enter a value to test for primality:

20051

probably prime

- - - - - - - - - - - - - - - - - - -

Enter a value to test for primality:

49597

probably prime

- - - - - - - - - - - - - - - - - - -

Enter a value to test for primality:

52721

probably prime

- - - - - - - - - - - - - - - - - - -

Enter a value to test for primality:

86069

probably prime

- - - - - - - - - - - - - - - - - - -

Method2: Fermat Method

•Pseudocode:

Algorithm *FermatMethod(k,n)*

Input: ***n*** an integer to test for primality, ***k*** a parameter that determines the accuracy of the test.

output: Boolean value.

loop while *k* > 0

    a ← random number between 2 and n-2

    x ← a^n-1

    if gcd(a,n) ≠ 1 **then**

        return false

    end if

    If x mod n ≠ 1 **then**

        return false

    end if

    k ← k-1

end loop

return true

•Java Implementation:

```
public static boolean FermatMethod(long k , long n){

while(k > 0) {

long a = 2+(int)Math.random()*(n-4);
long x = modPow(a, n-1, n);

if(gcd(a , n)!= 1)
return false;

if(x%n != 1)
return false;

k--;
}

return true;

}
```

•Explanation:

First we have a loop for the number of times we want to test a number for primality, inside of it we generate a random number ‘a’ with range [2.. n-2],then we need to calculate a^n-1 (mod n) we calculated a^n-1 by calling our implemented method ‘modpow’and stored it in ‘x’, also we need to make sure there aren’t any common devisors between ‘a’ and ‘n’ if it is proven false then we can take x mod n (which is a^n-1 (mod n))and check the result if it is equal to 1 or not.

•Sample run:

Enter the number of times to test for primality:

10

Enter a value to test for primality:

341

probably prime

- - - - - - - - - - - - - - - - - - -

Enter the number of times to test for primality:

11

Enter a value to test for primality:

561

probably prime

- - - - - - - - - - - - - - - - - - -

Enter the number of times to test for primality:

15
Enter a value to test for primality:
1729
probably prime
- - - - - - - - - - - - - - - - - - -
Enter the number of times to test for primality:
10
Enter a value to test for primality:
19973
probably prime
- - - - - - - - - - - - - - - - - - -
Enter the number of times to test for primality:
10
Enter a value to test for primality:
20051
probably Prime
- - - - - - - - - - - - - - - - - - -
Enter the number of times to test for primality:
5
Enter a value to test for primality:
49597
probably prime
- - - - - - - - - - - - - - - - - - -
Enter the number of times to test for primality:
5
Enter a value to test for primality:
52721
probably prime
- - - - - - - - - - - - - - - - - - -
Enter the number of times to test for primality:
20
Enter a value to test for primality:
86069
probably prime
- - - - - - - - - - - - - - - - - - -

Method3: Miller-Rabin

•Pseudocode:

Algorithm millerRabinIsPrime*(n,k)*

Input: ***n*** an integer to test for primality, ***k*** a parameter that determines the accuracy of the test.

output: Boolean value.

```
 If n<2 then
   return false
 End if

 if n=2 then
   return true
End if

 If n mod 2 = 0 then
   return false
 End if

m ← 0
e ← 0

loop for i ← 1 to n-1  do
    x ←2^I
    If (n-1) mod x = 0  then
       m ← (n-1)/x
       e ←  i
    End if
   If (n-1) mod x ≠ 0
     Break;
   End if
End loop

ver: loop for i ← 0 to k  do

        a← random number between 2 and n-1
        x ← a^m mod n
       if(x =1 or x = n-1)
          Continue
       End if
       loop for j← 0 to e  do
          x ←x^2mod n
          If x=1 then
            return false
          End if

         If x=n-1 then
            Continue ver
         End if
      End loop
   return false
End loop
```

return true

•Java Implementation:

```
public static boolean millerRabinIsPrime(int n, int k) {

if(n < 2)
return false;
if(n == 2)
return true;
if(n % 2 == 0)
return false;

long m = 0;
long e = 0;
for(int i = 1; i < n-1 ; i++) {
int x = (int)Math.pow(2, i);

if((n-1)%x == 0) {
m = (n-1)/x;
 e = i;
}

if((n-1)%x != 0)
break;
}

ver:     for(int i = 0; i < k ; i++) {

long a = 2+(long)Math.random()*(n-3);
long x = modPow(a , m ,n);

  if(x == 1 || x == n-1)
continue;

for(int j = 0 ; j < e ; j++) {

x = modPow(x , 2 ,n);

if( x == 1)
return false;

if(x == n-1)
continue ver;


}

return false;
}
```

return true;

}

•Explanation:

Method will first handle the base cases if n<2 or if n is even, it will return false.
It will search for an odd number m such that n-1 can be written as m*2^e then it will check the following k times with a for loop: generates an a between 2 and n-1, compute x = a^m (mod n) with our method ‘modPow’ if result is 1 or –1, it will continue the loop else it will do the following e times with a for loop: x = (x*x) % n if the result is 1 return false if –1 it will continue the outer loop if we continue through all the iterations it will return true at the end of the method meaning its prime else if it did not continue It will reach the end of the loop returning false meaning it didn’t satisfy the condition.

•Sample run:

```
Enter the number of times to test for primality:
5
Enter a value to test for primality:
341
composite
   - - - - - - - - - - - - - - - - - - -
Enter the number of times to test for primality:
10
Enter a value to test for primality:
561
composite
   - - - - - - - - - - - - - - - - - - -
Enter the number of times to test for primality:
10
Enter a value to test for primality:
1729
composite
   - - - - - - - - - - - - - - - - - - -
Enter the number of times to test for primality:
10
Enter a value to test for primality:
19973
probably prime
   - - - - - - - - - - - - - - - - - - -
Enter the number of times to test for primality:
10
Enter a value to test for primality:
20051
probably prime
   - - - - - - - - - - - - - - - - - - -
```

Enter the number of times to test for primality:
5
Enter a value to test for primality:
49597
probably prime
- - - - - - - - - - - - - - - - - - -
Enter the number of times to test for primality:
16
Enter a value to test for primality:
52721
probably prime
- - - - - - - - - - - - - - - - - - -
Enter the number of times to test for primality:
7
Enter a value to test for primality:
86069
probably prime

EXTRA METHODS

Algorithm modPow(a , b ,c)
Input a the base , b the exponent and c the modulus
Output a^b mod c

```
res ← 1
Loop for I ← 0 to b-1 do
    res ← res*a
    res ← res mod c
End loop

return res mod c
```

•Java Implementation:

```java
public static long modPow(long a, long b, long c)
   {
      long res = 1;
      for (int i = 0; i < b; i++)
      {
         res *= a;
         res %= c;
      }
      return res % c;
   }
```

Algorithm gcd(a , b)
Input a and b to calculate their gcd
Output gcd of a and b

```
 If a > b
     If b = 0
         return a
     End if

     Else
         return gcd(b , a mod b)
     End else
 End if

 Else
     If a = 0
         Return b
     End if

     Else
       Return gcd(a , b mod a)

     End else
 End else
```

•Java Implementation:

```
if(a > b) {

if(b == 0)
return a;

else return gcd(b , a % b);


}

else {
if(a == 0)
return b;
else return gcd(a , b %a);
}}
```

Discussion

	Strength	Weakness	Accuracy
Trial Division	We can be certain if the number is prime or not Faster than the other two methods	We can’t reach a high range of numbers	The most accurate out of the three methods
Fermat	We can reach higher range of numbers unlike trial divison	Its not that accurate. Pseudoprimes can pass the test. Faster when figuring out composite numbers but slower with primes comparing to the third method	A lot of pseudoprimes pass the test The method isn't reliable since it yields Different results for the same number

Miller-Rabin	Its more accurate than fermat and we can reach higher range of numbers Pseudoprimes are annonced as composites.	Slower when figuring out composite number but faster with primes than the second method	Few psudoprimes pass the test more accurate than fermat

K = 10

Efficiency comparison:

n	**Method 1**	**Method 2**	**Method 3**
341	0.0004283	0.000743	0.0011614
561	0.000527	0.0007541	0.0009658
19973	0.0003745	0.0057174	0.0017754
20051	0.0003028	0.0048482	0.0022307