Practical-3

AIM: Implement Randomized Primality Testing algorithm using Fermat's Theorem.

Fermat's Little Theorem:

If n is a prime number, then for every a, 1 < a < n-1, $\mathbf{a^{n-1} \equiv 1 \pmod{n}}$ i.e. in other words $\mathbf{a^{n-1} \% n = 1}$

Example:

1. Since 5 is prime, $2^4 \equiv 1 \pmod 5$ [or $2^4 \%5 = 1$],
   $3^4 \equiv 1 \pmod 5$ and $4^4 \equiv 1 \pmod 5$
2. Since 7 is prime, $2^6 \equiv 1 \pmod 7$,
   $3^6 \equiv 1 \pmod 7$, $4^6 \equiv 1 \pmod 7$
   $5^6 \equiv 1 \pmod 7$ and $6^6 \equiv 1 \pmod 7$

This method is a probabilistic method and is based on Fermat's Little Theorem. If a given number is prime, then this method always returns true. If the given number is composite (or non-prime), then it may return true or false, but the probability of producing incorrect results for composite is low and can be reduced by doing more iterations.

Algorithm:

// Higher value of k indicates probability of correct
// results for composite inputs become higher. For prime
// inputs, result is always correct
1) Repeat following k times:
   a) Pick a randomly in the range [2, n - 2]
   b) If gcd(a, n) ≠ 1, then return false
   c) If a^{n-1} mod n ≠ 1 (mod n), then return false
2) Return true [probably prime].

Algorithm:GCD(a,b)

```
int GCD(int a,int b)
{
   if(a<b)
      return GCD(b,a)
  else if (a%b= =0)
     return b;
  else
    return GCD(b,a%b);
}
```

Algorithm: Modular Exponentiation (a,m,n)

//This will compute a^m mod n

1. res=1 /*To store the result of a^m mod n */
2. Convert Exponent 'm' into Binary and store its bits in Array X. Least significant bit is say at 0th Index and Most significant bit is say at (n-1)th Index.
3. count=X.length //j indicates number of bits required to represent exponent m
4. for (i=count-1;i>=0;i- -)

```
{
   res=(res*res)%n; //Square the result
   if(X[i]= =1) /*If ith bit is set then Multiply else skip
   {
       res=(res*a)%n; //Multiply
   }
}
```

5. return res;

To compute a^m mod n, $\log_2(m)$ multiplications are required, so to compute a^{n-1} mod n, $\log_2(n)$ multiplications are required.

Thus we can use Multiply and Square Algorithm to find out the Modular Exponentiation i.e. to compute a^m mod n and Euclidean's algorithm to find out GCD (Greatest Common Divisor) of two numbers.

As both can be computed in Logarithmic Time , overall running time will be upper bounded by O(klog(n)).

Exercise

1. What is the need of Randomized algorithm for Primality Testing?
2. Justify: Fermat's algorithm to test primality is the Monte-Carlo type of algorithm.
3. Which kind of error can this algorithm make? How to reduce probability of Error?
4. Trace Modular Exponentiation on following by showing all the steps: **145^47 MOD 48**