**RSA Project**

1. **Source Code:**

**#BigInteger Class :**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace RSAProject

{

class BigInteger

{

public string myString;

public static KeyValuePair<BigInteger, BigInteger> DivResult;

public BigInteger()

{

this.myString = " ";

}

public BigInteger(string a)

{

this.myString = a;

}

public void addZeros(BigInteger B)

{

int len1 = myString.Length, len2 = B.myString.Length;

if (len1 > len2)

for (int i = 0; i < (len1 - len2); i++)

B.myString = B.myString.Insert(0, "0");

else if (len1 < len2)

for (int i = 0; i < (len2 - len1); i++)

myString = myString.Insert(0, "0");

}

public BigInteger Power(int length)

{

int len = myString.Length;

for (int i = 0; i < length; i++)

myString = myString.Insert(len, "0");

for (int i = 0; i < len; i++)

{

if (myString[0] != 0)

break;

myString = myString.Remove(0);

len--;

}

return this;

}

private string Reverse(string num)

{

int j = 0;

int len = num.Length;

char[] result = new char[len];

for (int i = len - 1; i >= 0; i--)

{

result[j] = num[i];

j++;

}

return new string(result);

}

public static bool operator <(BigInteger A, BigInteger B)

{

A.addZeros(B);

if (A.myString.CompareTo(B.myString) == -1)

return true;

else

return false;

}

public static bool operator >(BigInteger A, BigInteger B)

{

A.addZeros(B);

if (B.myString.CompareTo(A.myString) == -1)

return true;

else

return false;

}

public BigInteger addition(BigInteger B)

{

addZeros(B);

string Str1 = Reverse(myString);

string Str2 = Reverse(B.myString);

int len1 = Str1.Length;

int len2 = Str2.Length;

int maxLength = len1 > len2 ? len1 : len2;

string result = "";

int carry = 0;

int j = 0;

for (j = 0; j < maxLength; j++)

{

int num1 = Str1[j] - 48;

int num2 = Str2[j] - 48;

int summation = num1 + num2 + carry;

if (summation >= 10)

{

summation -= 10;

carry = 1;

}

else

carry = 0;

result += summation;

}

if (carry == 1)

result += "1";

result = Reverse(result);

if (result == "")

result = "0";

return new BigInteger(result);

}

public BigInteger subtaction(BigInteger B)

{

int num3 = 0;

addZeros(B);

bool bow = false;

if (myString.CompareTo(B.myString) == -1)

{

string tmp = myString;

myString = B.myString;

B.myString = tmp;

}

string Str1 = Reverse(myString); ;

string Str2 = Reverse(B.myString);

int len1 = Str1.Length;

int len2 = Str2.Length;

int maxLength = len1 > len2 ? len1 : len2;

string result = "";

for (int j = 0; j < maxLength; j++)

{

int num1 = Str1[j];

int num2 = Str2[j];

if (bow == true)

{

num1 = num1 - 1;

bow = false;

}

if (num1 >= num2)

num3 = num1 - num2;

else

{

num1 = num1 + 10;

num3 = num1 - num2;

bow = true;

}

result += num3;

}

result = Reverse(result);

int resLen = result.Length;

while (resLen != 0 && result[0] == '0')

{

result = result.Remove(0, 1);

resLen--;

}

if (result == "")

result = "0";

return new BigInteger(result);

}

public KeyValuePair<BigInteger, BigInteger> division(BigInteger B)

{

BigInteger tmpB = new BigInteger();

tmpB.myString = (string)B.myString.Clone();

if (this < B)

return new KeyValuePair<BigInteger, BigInteger>(new BigInteger("0"), this);

DivResult = division((B.addition(tmpB)));

BigInteger G = new BigInteger();

G = DivResult.Key.addition(DivResult.Key);

DivResult.Key.myString = G.myString;

if (DivResult.Value < B)

{

int divLen = DivResult.Value.myString.Length;

while (divLen != 0 && DivResult.Value.myString[0] == '0')

{

DivResult.Value.myString =

DivResult.Value.myString.Remove(0, 1);

divLen--;

}

if (DivResult.Value.myString == "")

DivResult.Value.myString = "0";

return DivResult;

}

else

return new KeyValuePair<BigInteger, BigInteger>

(DivResult.Key.addition(new BigInteger("1")),

DivResult.Value.subtaction(B));

}

public bool checkOdd(string val)

{

int num = Convert.ToInt32(val[val.Length - 1].ToString());

if (num % 2 != 0)

return true;

else

return false;

}

public BigInteger Multiplication(BigInteger S)

{

BigInteger result = new BigInteger();

addZeros(S);

int len1 = myString.Length, len2 = S.myString.Length;

int resultLen = result.myString.Length;

if (len1 != 1 && len1 % 2 != 0)

{

myString = myString.Insert(0, "0");

len1++;

S.myString = S.myString.Insert(0, "0");

len2++;

}

else if (len1 == 1)

{

int mul = int.Parse(myString) \* int.Parse(S.myString);

result.myString = (mul.ToString());

return result;

}

BigInteger A = new BigInteger(myString.Substring(0,len1/2));

BigInteger B = new BigInteger(myString.Substring(len1/2));

BigInteger C = new BigInteger(S.myString.Substring(0,len2/2));

BigInteger D = new BigInteger(S.myString.Substring(len2/2));

BigInteger AC = A.Multiplication(C);

BigInteger BD = B.Multiplication(D);

BigInteger Z = (A.addition(B)).Multiplication((C.addition(D)));

BigInteger ACBD = (AC.addition(BD));

BigInteger NZ = Z.subtaction(ACBD);

result = AC.Power(len1).addition(NZ.Power(len1/2).addition(BD));

while (result.myString.Length != 0 && result.myString[0] == '0')

result.myString = result.myString.Remove(0, 1);

if (result.myString == "")

result.myString = "0";

return result;

}

public BigInteger incryption(BigInteger pubKey, BigInteger n)

{

BigInteger decrept = new BigInteger();

BigInteger power = new BigInteger();

BigInteger powerTmp = new BigInteger();

BigInteger tmp = new BigInteger();

bool odd = false;

KeyValuePair<BigInteger, BigInteger> result;

if (pubKey.myString == "0")

{

decrept.myString = "1";

return decrept;

}

tmp.myString = "2";

result = pubKey.division(tmp);

power = incryption(result.Key, n);

result = power.division(n);

power = result.Value;

odd = checkOdd(pubKey.myString);

powerTmp.myString = (string)power.myString.Clone();

tmp = power.Multiplication(powerTmp);

if (odd == false)

{

result = tmp.division(n);

decrept = result.Value;

return decrept;

}

else

{

result = division(n);

decrept = (result.Value).Multiplication(tmp);

result = decrept.division(n);

decrept = result.Value;

return decrept;

}

}

public BigInteger decryption(BigInteger prvKey, BigInteger n)

{

BigInteger incrept = new BigInteger();

BigInteger power = new BigInteger();

BigInteger powerTmp = new BigInteger();

BigInteger tmp = new BigInteger();

bool even = false;

KeyValuePair<BigInteger, BigInteger> result;

if (prvKey.myString == "0")

{

incrept.myString = "1";

return incrept;

}

tmp.myString = "2";

result = prvKey.division(tmp);

power = decryption(result.Key, n);

result = power.division(n);

power = result.Value;

even = checkOdd(prvKey.myString);

powerTmp.myString = (string)power.myString.Clone();

tmp = power.Multiplication(powerTmp);

if (even == false)

{

result = tmp.division(n);

incrept = result.Value;

return incrept;

}

else

{

result = division(n);

incrept = (result.Value).Multiplication(tmp);

result = incrept.division(n);

incrept = result.Value;

return incrept;

}

}

}

}

**#Main :**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.IO;

namespace RSAProject

{

class Program

{

static void Main(string[] args)

{

BigInteger result = new BigInteger();

BigInteger message = new BigInteger();

BigInteger key = new BigInteger();

BigInteger n = new BigInteger();

string choice;

FileStream fs = new FileStream("d://TestRSA.txt", FileMode.Open);

StreamReader sr = new StreamReader(fs);

FileStream f;

StreamWriter sw;

int num = int.Parse(sr.ReadLine());

for (int i = 0; i < num; i++)

{

f = new FileStream("d://result.txt", FileMode.Append);

sw = new StreamWriter(f);

n.myString = sr.ReadLine();

key.myString = sr.ReadLine();

message.myString = sr.ReadLine();

choice = sr.ReadLine();

if (choice == "0")

{

float time = System.Environment.TickCount;

result = message.incryption(key, n);

Console.WriteLine("the incrypted message is " + result.myString);

Console.WriteLine(" ");

sw.WriteLine(result.myString);

float time1 = System.Environment.TickCount;

float TimeCase1 = (time1 - time) / 60;

Console.WriteLine("the Execution time of the incryption process is " + TimeCase1 + " Seconds");

Console.WriteLine(" ");

}

else if (choice == "1")

{

float time = System.Environment.TickCount;

result = message.decryption(key, n);

Console.WriteLine("the decrypted message is " + result.myString);

Console.WriteLine(" ");

sw.WriteLine(result.myString);

float time1 = System.Environment.TickCount;

float TimeCase2 = (time1 - time) / 60;

Console.WriteLine("the Execution time of the decryption process is " + TimeCase2 + " Seconds");

Console.WriteLine(" ");

}

sw.Close();

}

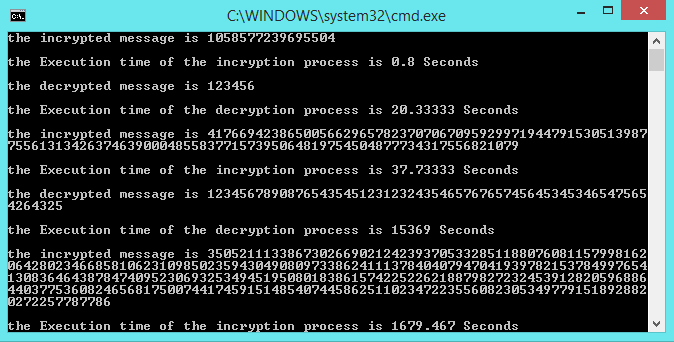
sr.Close();

}

}

}

1. **The Execution Time for first 5 cases is**

****

1. **The Analysis :**

**\* “AddZeros” 🡪 O(N)**

**\* “Power” 🡪 O(N^2)**

**\* “Reverse” 🡪 O(N)**

**\* “<” 🡪 O(N)**

**\* “>” 🡪 O(N)**

**\* “addition” 🡪 O(N)**

**\* “subtraction” 🡪 O(N^2)**

**\* “division” 🡪**

**T(N) = T(2N) + B^2 🡪 1**

**T(2N) = T(2^2 N) + B^2**

**T(N) = T(2^2 N) + 2B^2 🡪 2**

**T(2^2 N) = T(2^3 N) + B^2**

**T(N) = T(2^3 N) + 3B^2 🡪 3**

**The general formula is**

**T(N) = T(2^K N) + KB^2**

**The termination condition at**

**T(0) = 1**

**2^K N = 1 🡪 K = log(1/N)**

**Log(1/N) = log1 – log N**

**K = logN**

**T(N) = 1 – B^2LogN**

**The order is 🡪 O(B^2LogN)**

**\*”checkOdd” 🡪 O(C)**

**\*”multiplication” 🡪**

**T(N) = 3T(N/2) + N^2**

**A=3 , B=2 , F(N)=N^2**

**N^2 VS. N^1.5**

**Case 3 in master method :**

**N^2 = Ω (N^(1.5 + £))**

**N = Ω (N^(0.5 + £))**

**0 < € < 0.5 🡪 TRUE**

**A.F(N/B) <= C.F(N)**

**3(N^2 / 2) <= C.N^2**

**¾ <= C 🡪 C<1 ----> TRUE**

**T(N) = θ(N^2)**

**\*”incryption & decryption” 🡪**

**T(N) = T(N/2) + B^2.LogN 1**

**T(N/2) = T(N/2^2) + B^2.LogN/2**

**T(N) = T(N/2^2) +B^2.LogN + B^2.LogN/2 🡪 2**

**T(N/2^2) = T(N/2^3) + B^2.LogN/2^2**

**T(N) = T(N/2^3) + B^2.LogN + B^2.LogN/2 + B^2.LogN/2^2 🡪 3**

**T(N) = T(N/2^K) + B^2Σi=0 to k (LogN/2^k)**

**T(N) = T(N/2^K) + B^2 Σ (LogN – Log 2^k)**

**T(N) = T(N/2^K) + B^2.LogN – B^2 ΣK**

**The termination condition at**

**T(0) = 1**

**N/2^K = 1 🡪 N = 2^K**

**K = LogN**

**T(N) = 1 + B^2.LogN - B^2 Σ from I=0 to logN (LogN)**

**Σ(LogN) = LogN(LogN+1) /2**

**T(N) = θ(B^2.logN.logN)**