**Assignment 1**

Advanced Algorithms 1 (7081)

**Group 13**

Jianfeng Zeng (zengjg)

Xin Li (li4x7)

Alex Stewart (stewaaw)

Nathan Daughety (daughenl)

Written Part [50pts]

1. *[10pts] Page 22. Exercise 1.1 (a), (d).*

Compute binary representation of 123 obtaining 1111011.

Compute binary representation of 711 obtaining 1011000111.

1. *[10pts] Page 22, Exercise 1.8.*
2. (24, 108)
3. (23, 108)
4. (89, 144)
5. (1953, 1937)
6. *[10pts] Page 22, Exercise 1.9. Prove that the formula you have given holds.*

**Proof:** let ,

for some x and for some y.

is divisible by a and b by the definition of lcm.

Since x and y are coprime by the definition of gcd;

therefore, we have .

1. *[10pts] Page 24, Exercise 1.19.*

Instead of evaluating a polynomial, say using Horner’s method directly, we can divide into one sub polynomial that consists of indeterminates with only even degree, and one sub polynomial that consists of indeterminates with only odd degree, . Then we have,

We can then evaluate even polynomials and odd polynomials separately using Horner’s method with multiplications and additions. We need to do multiplications because we multiply the coefficients with instead of . To evaluate , we simply add the results from and . On the other hand, to evaluate , we simply subtract from , since -v can only affect the indeterminates with odd degree. Thus, we simply need to do one more addition to evaluate .

1. *[10pts] Design an efficient algorithm for computing the majority element in a list L[0:n – 1] if it exists or determining that it doesn’t exist. An element is a majority element if it is repeated more than n/2 times. Don’t settle for straightforward algorithm. Try to design a linear-time algorithm. Analyze the algorithm you have designed in a), i.e., determine B(n) and W(n).*

**function** findMajorityElement(list[0:n])

**input:** list[0:n] (an array of elements)

**output:** -1 (not found) or i (element index)

HashTable {}

Max 0 (used to track the current max value in the table)

**for** i 0 **to** n - 1 **do**

**if** list[i] **not** in table **then**

HashTable [list[i]] 1

**else**

HashTable [list[i]] +1

**endif**

**if** HashTable [list[i]] > Max **then**

Max HashTable [list[i]]

**if** Max > (n - 1) / 2 **then**

**return** i

**endif**

**if** ((n - 1) / 2) > ((n – 1) - i + Max) **then**

**return** -1

**endif**

**endif**

**endfor**

**return** -1

**end** findMajorityElement

**Analysis:** The algorithm iterates through the input list and uses a look up table to keep tracking the number of times an element has encountered. The table stores an element into the table initialized beforehand with value set for 1 if the element is not found in the table or increments the value by 1 otherwise. A variable Max is also initialized for 0 at the start of the algorithm to store the current max value in the table. The iteration stops and returns the element index when the max value is larger than one half of the list length. This max value is used to`a stop the iteration when one half of the list length is larger than the list length minus the current index plus the max value. In another words, the iteration stops and returns -1 when there is no way the current max value will exceed one half of the list length after iterating through the rest of the list. The algorithm at the end returns -1 if no return statement is hit inside the list iteration. The time complexity for this algorithm is **O(n)** as it iterates through the entire list. The space complexity for this algorithm is also **O(n)** as it creates a key-value relation in the look up table for each element encountered in the input list. The best case, **B(n)** for this algorithm is because it stops whenever the current max value is larger than one half of the list. On the other hand, the worst case, **W(n)** for this algorithm is because the majority element could be at the very end of the list.

Programming Project [50pts]

This programming project consists of total 3 files: main.cpp, RSA.hpp, and Makefile

**main.cpp**

file path: ./src/main.cpp

/\*  
 \*  
 \* Course Name: Advanced Algorithm I  
 \* Group Assignment 1  
 \* Group: 13  
 \* Group Members:  
 \* Jianfeng Zeng (zengjg)  
 \* Xin Li (li4x7)  
 \* Alex Stewart (stewaaw)  
 \* Nathan Daughety (daughenl)  
 \*  
 \*/  
  
#include "./RSA.hpp"  
  
using namespace std;  
  
#define BASE 27  
  
int main(){  
 // 1) Get two prime numbers  
 RSA rsa = RSA();  
 rsa.set\_p(2000); // select a range to get random prime number  
 rsa.set\_q(3000); // select a range to get random prime number  
 rsa.set\_phi();  
  
 // 2) Get public key e  
 rsa.set\_publicKey();  
  
 // 3) Get message in BEARCATII  
 rsa.set\_msg\_bc();  
  
 // 4) Convert message from list of numbers of base 27 to a single number of decimal  
 rsa.set\_msg\_decimal();  
  
 // 5) Encrypt message  
 rsa.set\_n();  
 rsa.set\_c();  
  
 // 6) Decrypt message (get privateKey)  
 rsa.set\_privateKey();  
 rsa.set\_m();  
  
 // 7) Change base back to 27  
 rsa.set\_msg\_base27();  
  
 // 8) 27 to Eng  
 rsa.set\_msg\_original();  
  
 // 9) Report  
 rsa.report();  
  
 return 0;  
}

**RSA.hpp**

file path: ./src/RSA.hpp

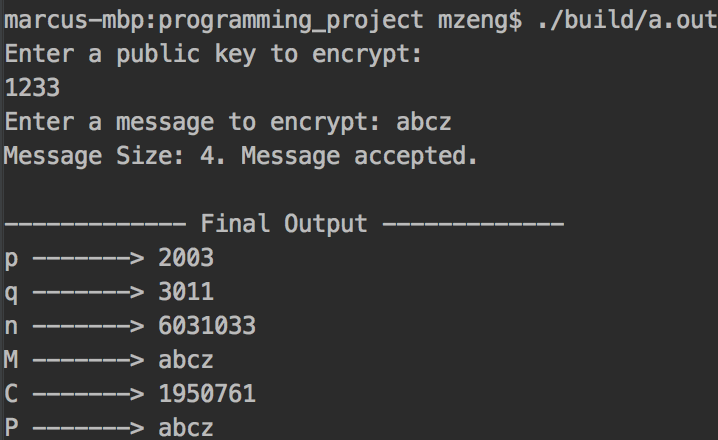
#include <stdio.h>  
#include <stdlib.h>  
#include <vector>  
#include <cmath>  
#include <iostream>  
#include <string>  
#include <stdexcept>  
#include <math.h>  
#include <random>  
  
using namespace std;  
  
class RSA{  
  
public:  
 // Constructor using initialization list  
 RSA(): p(0), q(0){}  
  
 void set\_p(int range){  
 this->p = this->generatePrimeCandidate(range);  
 }  
  
 void set\_q(int range){  
 this->q = this->generatePrimeCandidate(range);  
 }  
  
 void set\_phi(){  
 this->phi = (this->p-1)\*(this->q-1);  
 }  
  
 void set\_publicKey(){  
 this->publicKey = this->getPublicKey();  
 }  
  
 void set\_msg\_bc(){  
 this->msg\_bc = this->getMsg();  
 }  
  
 void set\_msg\_decimal(){  
 this->msg\_decimal = this->polyEval(this->msg\_bc, 27);  
 }  
  
 void set\_n(){  
 this->n = this->p \* this->q;  
 }  
  
 void set\_c(){  
 this->c = this->modExp(this->msg\_decimal, this->publicKey, this->n);  
 }  
  
 void set\_privateKey(){  
 this->privateKey = this->getPrivateKey(this->publicKey, this->phi);  
 }  
  
 void set\_m(){  
 this->m = this->modExp(this->c, this->privateKey, this->n);  
 }  
  
 void set\_msg\_base27(){  
 this->msg\_base27 = this->changeBase(this->m, 27);  
 }  
  
 void set\_msg\_original(){  
 this->msg\_original = this->BCtoEng(this->msg\_base27);  
 }  
  
 void report(){  
 cout << "" << endl;  
 cout << "------------- Final Output -------------" << endl;  
 cout << "p -------> " << this->p << endl;  
 cout << "q -------> " << this->q << endl;  
 cout << "n -------> " << this->n << endl;  
 cout << "M -------> " << this->BCtoEng(msg\_bc) << endl;  
 cout << "C -------> " << this->c << endl;  
 cout << "P -------> " << this->msg\_original << endl;  
 }  
  
private:  
 int p, q; // 2 random select prime number  
 int phi; // phi = (p - 1)(q - 1)  
 unsigned long long publicKey; // user enter a public key  
 vector<int> msg\_bc; // message in BearcatASCII format  
 int msg\_decimal; // message in BearcatASCII -> decimal format  
 int n; // n = pq  
 int c; // encrypted message  
 int privateKey; // private key from calculation  
 int m; // decrypted message  
 vector<int> msg\_base27; // message in BearcatASCII format  
 string msg\_original; // original message (string)  
  
 // ##############################################################  
 // KEYS FUNCTIONS  
 // ##############################################################  
 unsigned long long getPublicKey(){  
 unsigned long long publicKey = 0;  
  
 cout << "Enter a public key to encrypt: " << endl;  
 cin >> publicKey;  
  
 while (publicKey == 0 || euclidGCD(publicKey, this->phi) != 1){  
 cout << "Invalid public key, please enter another one: " << endl;  
 cin.clear();  
 cin.ignore(9, '\n');  
 cin >> publicKey;  
 }  
  
 cin.clear();  
 cin.ignore(9, '\n'); // cin.ignore(nCountOfChar, delim);  
  
 return publicKey;  
 }  
  
 int getPrivateKey(int publicKey, int phi){  
 int x = 0;  
 int y = 0;  
 int g = gcdExtended(publicKey, phi, &x, &y);  
  
 if (x < 0) x = phi + x;  
  
 return x;  
 }  
  
 // ##############################################################  
 // MESSAGE FUNCTIONS  
 // ##############################################################  
 vector<int> getMsg(){  
 char message[256];  
 int msgCount;  
 int j = 0;  
  
 cout << "Enter a message to encrypt: ";  
 cin.getline(message, 256);  
 msgCount = cin.gcount() - 1; //Ignore end of line character  
 cout << "Message Size: " << msgCount << ". ";  
  
 vector<int> BEARCATII(msgCount, -1);  
  
 for(int i = 0; i < msgCount; i++){  
 if(message[i] == 32){ //32 is ASCII for space  
 BEARCATII[j] = 0;  
 j++;  
 }  
 else if((message[i] > 96) && (message[i] < 123)){  
 BEARCATII[j] = int(message[i]) - 96; //Subtract 96 to turn ASCII alphabet into BEARCATII  
 j++;  
 }  
 else{  
 cout << "Invalid characters used, please try again." << endl;  
 BEARCATII.clear(); // Clear vector  
 return BEARCATII; // Return empty vector if error/invalid characters detected  
 }  
 }  
 cout << "Message accepted." << endl;  
 return BEARCATII;  
 }  
  
 string BCtoEng(vector<int> BEARCATII){  
 string raw\_message;  
  
 for(int i = 0; i < int(BEARCATII.size()); i++){ // Iterate through vector converting BEARCATII back to ASCII representation  
 if(BEARCATII[i] == 0){  
 raw\_message += ' '; // 0 represents 'space'  
 }  
 else{  
 raw\_message += char(BEARCATII[i] + 96); // Add 96 to adjust BEARCATII to ASCII  
 }  
 }  
 return raw\_message; // Return string rather than a C-Str for simplicity  
 }  
  
 // ##############################################################  
 // NUMBER CONVERTING FUNCTIONS  
 // ##############################################################  
 int polyEval(vector<int> coef, int base = 27){  
 coef = this->msg\_bc;  
 int l = (int)coef.size();  
 int indexFactor = l - 1;  
 int even = 0;  
 int odd = 0;  
  
 //Corner check for l = 1 or 2  
 if (l == 1) return coef.at(0);  
  
 if (l == 2) return (coef.at(0) + coef.at(0) \* base);  
  
 //If l > 2, determine initial value for even & odd  
 if (l % 2 == 0){  
 odd = coef.at(0);  
 even = coef.at(1);  
 }  
 else{  
 odd = coef.at(1);  
 even = coef.at(0);  
 }  
  
 //Polyn Calculation  
 for(int i = 2; i < l; i++){  
 int index = abs(i - indexFactor);  
 if (index % 2 == 0) even = even \* base \* base + coef.at(i);  
 else odd = odd \* base \* base + coef.at(i);  
 }  
  
 //multiply base for odd after loop  
 odd \*= base;  
 return (even + odd);  
 }  
  
 vector<int> changeBase(int num, int base){  
 vector<int> res;  
  
 while (num > base){  
 int r = num % base;  
 num = (num - r) / base;  
 res.insert (res.begin(), int(r));  
 }  
 res.insert (res.begin(), int(num));  
 return res;  
 }  
  
 // ##############################################################  
 // PRIME TEST FUNCTIONS  
 // ##############################################################  
  
 /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 |Function: modExp(a, b, n)  
 |Input: a (long -- real number), b (+ integer), n (long)  
 |Output: a^b mod n  
 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
 long modExp(long a, unsigned long b, long n){  
 long accumPowers = 1;  
 a = a % n;  
  
 while (b > 0){  
 // if LSB of y is 1 (number is odd)  
 if (b & 1){  
 accumPowers = (accumPowers \* a) % n;  
 }  
  
 // LSB in y is currently 0 (number is even)  
 a = (a \* a) % n;  
  
 // shift bits in b right 1 so next bit  
 // in line becomes LSB  
 b = b >> 1;  
 }  
 return accumPowers;  
 } // end modExp  
  
 int generatePrimeCandidate(int k){  
 if (k % 2 == 0) k = k + 1;  
  
 while (true){  
 k = k + 2;  
 bool prime = millerRabinWrapper(k, 40);  
 if (prime == true) return k;  
 }  
 }  
  
 /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 \* Function: millerRabin(int)  
 \* step 1: find n - 1 = (2^k) m  
 \* step 2: choose a from 1 < a < n - 1  
 \* step 3: compute b0 = a^m (mod n), bi = (bi-1)^2  
 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
 bool millerRabin(int n){  
 int nMinus1 = n - 1;  
  
 // step 1  
 int m = n - 1;  
 int k = 0;  
  
 while (m % 2 == 0){  
 m = m / 2;  
 k = k + 1;  
 }  
  
 // step 2  
 random\_device rd; // obtain a random number from hardware  
 mt19937 eng(rd()); // seed the generator  
 uniform\_int\_distribution<> distr(1, n-1); // define the range  
  
 int a = distr(eng);  
  
 // step 3  
 int b;  
 b = modExp(a, m, n);  
 if (b == 1 || b == nMinus1) return true;  
  
 // iterate b after b0 up to k - 1 times  
 for (int i = 0; i < k - 1; i ++){  
 b = modExp(b, 2, n);  
 if (b == 1) return false;  
 if (b == nMinus1) return true;  
 }  
 return false;  
 }  
  
 bool millerRabinWrapper(int n, int k){  
 for (int i = 0; i < k; i++){  
 if (!millerRabin(n)){  
 return false;  
 }  
 }  
 return true;  
 }  
  
 // ##############################################################  
 // GCD  
 // ##############################################################  
  
 /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 |Description: This function finds the greatest common divisor  
 | of the inputs a and b.  
 |Function: euclidGCD(a, b)  
 |Input: a, b (both nonnegative integers)  
 |Output: greatest common divisor  
 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
 int euclidGCD(int a, int b){  
 int remainder;  
  
 // while the remainder of a % b != 0 do:  
 while (b != 0){  
 // gcd(a, b) = gcd(a, a % b)  
 remainder = a % b;  
 a = b;  
 b = remainder;  
 }  
 return a;  
 } // end euclidGCD  
  
 /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 |Description: This function finds the greatest common divisor  
 | of the inputs a and b while finding values s and  
 | t such that sa + tb = g where g is the gcd(a, b).  
 |Function: extendEuclidGCD(a, b, g, s, t)  
 |Input: a, b (nonnegative integers)  
 |Output: return g = gcd(a,b) and integers s and t such that  
 |sa + tb = g  
 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
 int gcdExtended(int a, int b, int \*x, int \*y){  
 if (a == 0){  
 \*x = 0; \*y = 1;  
 return b;  
 }  
  
 int x1, y1;  
 int gcd = gcdExtended(b%a, a, &x1, &y1);  
  
 \*x = y1 - (b / a) \* x1;  
 \*y = x1;  
  
 return gcd;  
 }  
  
};

**Makefile**

file path: ./Makefile

CXX ?= g++  
  
TARGET\_EXEC ?= a.out  
  
BUILD\_DIR ?= ./build  
SRC\_DIRS ?= ./src  
  
SRCS := $(shell find $(SRC\_DIRS) -name \*.cpp -or -name \*.c -or -name \*.s)  
OBJS := $(SRCS:%=$(BUILD\_DIR)/%.o)  
DEPS := $(OBJS:.o=.d)  
  
INC\_DIRS := $(shell find $(SRC\_DIRS) -type d)  
INC\_FLAGS := $(addprefix -I,$(INC\_DIRS))  
  
CPPFLAGS ?= $(INC\_FLAGS) -MMD -MP  
  
  
$(BUILD\_DIR)/$(TARGET\_EXEC): $(OBJS)  
 $(CXX) $(OBJS) -o $@ $(LDFLAGS)  
  
# c++ source  
$(BUILD\_DIR)/%.cpp.o: %.cpp  
 $(MKDIR\_P) $(dir $@)  
 $(CXX) $(CPPFLAGS) $(CXXFLAGS) -c $< -o $@  
  
.PHONY: clean  
  
clean:  
 $(RM) -r $(BUILD\_DIR)  
  
-include $(DEPS)  
  
MKDIR\_P ?= mkdir -p

**Sample Run**

****