King Saud University College of Computer and Information Sciences Computer Science Department	
CSC 281 Discrete Math for Computer Science	First Semester 1443

Section: 41204 - 66516	Team: 13	Phase: 1-2
Students		
Name		ID
Reema Aldawood		441201050
Raghad Aljuhaimi		441201212
Raghad Aljarbou		441201324

PHASE 1

1.

Compute 3^{80} mod 5 using the provided algorithm without using calculator. (You should never have to keep track of a number larger than 5 except for the exponent itself when you are doing these calculations!) Show all steps.

Divide 80 repeatedly by 2, Subtract 1 if the number is odd

80, 40, 20, 10, 5, 4, 2, 1, 0

***Note:**

a. if the number is odd, we multiply it with the previous result.

b. if the number is even, we square the previous result.

1. b=0, $3^0 mod\ 5 = 1$
2. b=1, $3 \times 1 \text{ mod } 5 = 3$
3. b=2, $3^2 mod\ 5 = 4$
4. b=4, $4^2 mod\ 5 = 1$
5. b=5, $3 \times 1 \text{ mod } 5 = 3$
6. b=10, $3^2 mod\ 5 = 4$
7. b=20, $4^2 \text{mod } 5 = 1$
8. b=40, $1^2 mod\ 5 = 1$
9. b=80, $1^2 mod 5 = 1$

Therefore $3^{80} mod\ 5 = 1$

5.

Implement a version of modExp that computes b^{e}and then, after that computation is complete, takes the result mod n. Compare the speeds of these two algorithms in computing 3^{k}mod 5, for k = 80, k = 800, k = 8000, ..., k = 8, 000, 000. Explain.

Version 1:

3^{80} = 5734450700 nanoseconds.

3^{800} = 3739960801 nanoseconds.

3^{8000} = 4292416499 nanoseconds.

Version 2:

3^{80} = 8649110500 nanoseconds.

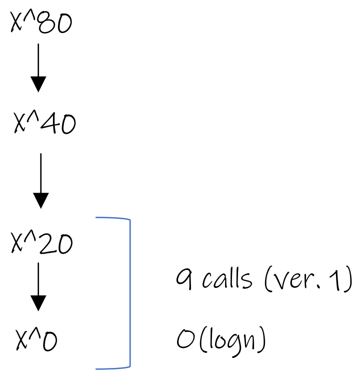
3^{800} = 3933501601 nanoseconds.

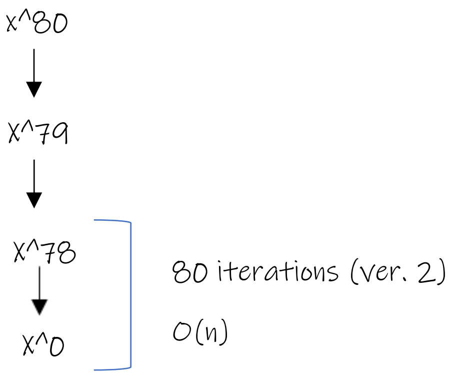
3^{8000}= 5864375700 nanoseconds.

In calculating number larger than 3^{8000} we encounter overflow in version 2 because what we were doing is computing b^{e}and then, after that computation is complete, we take the result mod n but since b^{e} can be very big number we can't compute it and store it in integer variable as it will eventually leads to overflow.

WHY VERSION 1?

Consider calling x^{80} mod 5 using Version 2 of modExp first will calculate $x^{79}$and then it goes on to calculate x^{78} up to x^{0} on the other hand the first version of modExp will make call to x^{40} instead of reducing this to x^{79} then x^{40} recursively makes another call to x^{20} up to x^{0}, what we can observe here that the first version will take less time than the second version (notice the recursion tree below).





Also, from the two algorithms we can see version 1 has O(logn) time complexity, while version 2 has O(n) time complexity and that makes version 1 more efficient and in term of speed it's considered faster than version 2.

Algorithm for version 2:

```
ModExp2 (b, e, n):

input: integers n ≥1, b and e ≥0

output: $b^e \bmod n$


for i=0 → e-1
    result ← result * b
end

return result mod n
```

PHASE 2

The study of prime integers and their characteristics began when Fermat's little theorem appeared and made several contributions to the theory of primes. Fermat's little theorem has surprising consequences for encryption and primality testing.

Primes do not follow a pattern of distribution on the number line; they appear chaotically.

As we mentioned, because prime numbers are not distributed regularly through integers, generating them can be very difficult. A solution to this problem is to test the primality of a number by passing it through one of the available primality tests.

1. Fermat primality test

Inputs*:*

n: a value to test for primality, n>3.

k: a parameter that determines the number of times to test for primality.

Output*: composite if n is composite, otherwise probably prime.*

```
Repeat k times:
Pick a randomly in the range [2, n − 2].
If  $a^{n-1} \not\equiv 1 \pmod{n}$, then return composite.
If composite is never returned: return probably prime.
```

Algorithm: Fermat primality test.

As we can notice in Fermat algorithm If the chosen **a** does not satisfy the equality, then **a** is called a Fermat witness, and n is composite.

Fermat pseudoprime numbers are composite integers that are always declared by the Fermat test as primes. Namely, a pseudoprime number always passes the condition of the Fermat test for any base integer **a**. There is an infinite quantity of Pseudoprime numbers which may raise the failure probability of this test.

2. Miller–Rabin primality test

This test is used in many practical cryptography systems because of its efficiency. The output of this algorithm is a proof that the number is a composite, or the number is probable prime.

Composite numbers that pass Miller-Rabin test are called strong-pseudoprimes and they are much fewer than the pseudoprimes. Furthermore, all Fermat pseudoprime numbers are catched and declared as composite.

PROS & CONS

Miller- Rabin pseudoprimes are called strong pseudoprimes

primality Test	Pros	Cons
Miller-Rabin	- Fast & efficient. - Significantly more accurate than the Fermat Primality Test.	- Strong pseudoprimes can pass the test.
Fermat	- Very simple to implement. - Base for many tests.	- Failure probability may reach 1. - Pseudoprime can pass the test.

STUDENT PEER EVALOUATION

Teamwork		
Raghad Aljarbou	**Raghad Aljuhaimi**	**Reema Aldawood**
Works on searching for useful resources and writes the report, revising and editing the source code.	Implements the algorithm and compare it to Reema's, helps in the final revision.	Implements the algorithm and compare it to Raghad's, helps in finding resources.

Part 1: Teamwork			
Criteria	Student 1	Student 2	Student 3
Work division: Contributed equally to the work			
Peer evaluation: Level of commitments (Interactivity with other team members), and professional behavior towards team & TA			
Project Discussion: Accurate answers, understanding of the presented work, good listeners to questions			
Time management: Attending on time, being ready to start the demo, good time management in discussion and demo			
Total/3			