WEEK 9 - Lecture 1

INTRODUCTION:

In this lecture we studied what about randomized Algorithms are and also discussed how we can check if a number is prime or not using the Miller-Rabin Primality Testing Algorithm

Randomized Algorithm's :

They are the algorithms where we use random numbers to decide what to do next in the logic of the algorithm.

Many times, the solution we have is very time consuming and inefficient and at those instances , it is better for us to use randomzied algorithms.

There are both pros and cons of using a randomized algorithm.

Pros:

- We can get the answer in a very good complexity since randomized algorithms are usually very fast.
- It is hard to break a randomized algorithm since we pick random numbers , so there is no uniform pattern and it is difficult to predict which numbers have been chosen. Hence it is difficult to break a randomzied algorithm .

Cons:

- They are not 100 percent accurate and there can be instances where randomized algorithm prints the wrong output.
- Generation of random numbers(truly random) is still an unsolved problem in computer science. Hence if someone gets to know our algorithm , then he can easily design an input with the motive to break our algorithm.

Miller-Rabin Algorithm:

This randomized algorithm gurantees a number is prime if it is actually prime . However for few composite numbers , it might say that the number is prime when in reality it is composite.

Still if we set the parameter k (which we shall introduce soon) , the probability that he algorithm will break would be very low.

The algorithm goes as follows :

- Say we are given an input n and we have to check if it is prime of not , then first we consider some base cases like if the number is say 2 , we can directly return true meaning number is prime or it if is even and not 2 then also we can directly return false.
- Now we actually start our algorithm , we pick k numbers say $a_1, a_2, ...a_k$ from the set of whole numbers till n-1 randomly.
- Now for every i from 1 to k , we do the following :
    - We find the value of $a^{n-1} mod n$ and check if it is equal to 1 or not . We know from Fermat's Theorem which we discussed in last class that if n is prime , then $a^{n-1} mod n == 1$. Hence if the value of the check variable is not 1 , we can return false since we would be sure that the number is composite .
    - Now since n is odd, n-1 would be even and hence we factorize n-1 as $n - 1 = a^{2^s.d}$ where s will be some odd number and hence we would be able to compute the sequence $a_i^{s.2^0}, a_i^{s.2^1}, a_i^{s.2^2}, ..., a_i^{s.2^k}$ mod n.
    - We start checking from right to left and find the position of the first element that is not equal to 1 . If the number at that position where value is not 1 is not -1 , that is neither 1 nor -1 , we return false claiming that the number n is composite.
- After doing this for k steps, we just return true saying that number n is prime , since if it would not have been prime according to us,then we would have already returned false in some earlier step.

The code for this algorithm would look as follows :

```
input=n;
if(n==2) return true;
if(n%2==0) return false;
Loop: repeat k times:
    pick a random integer a in the range [2, n − 1]
    if(a^(n-1)modn!=1) return false;
    write n-1 as (2^s).d where d is odd
    int x=a^s mod n;
    if(x==1 || x==n-1) continue;
    repeat s − 1 times:
        x ← x^2 mod n
        if x = n − 1 then
            continue Loop
    return false
return true
```

Proof that algorithm works correctly for prime number input:

We have handle the base cases . Let us look at the case where the input is odd .

This algorithm is based on Fermat's theorem and the proof for correctness lies at the depth of it that Fermat's theorem always returns mod value 1 if the number n is prime although it might output 1 also when the number n is prime in few cases .

So now we have to proof that the first number we witness from right to left that is not 1 has to be -1 for the number to qualify as a prime :

Let us proof this using contradiction . Let the first term that is not 1 be x.

So x is not equal to 1. Now we know every consecutive number in that sequence is the previous number raised to power of 2 .

Hence $x^2 mod n == 1 -> x^2 - 1 mod n == 0$. This would imply that the number $x^2 - 1$ is divisible by n .

This number can be factorized as follows :

$x^2 - 1 = (x-1)(x+1) = 0 mod n$

This means that at least one of these two factors must be divisible by n since we know mod value is zero.

But according to our assumption that b is not equal to 1 , $(x-1)$ cannot be divisible by n .

Hence $(x+1)$ would be divisible by n meaning that $(x-1)$ would not be divisible by n and so $x mod p == -1$.

So if the numbers mod value is -1 , it would be certain that n is indeed a prime number.

If input is odd composite :

We can show that the set of equations we considered above can be true for a composite number also in which case we would get a contradiction.

However we can show that the probability the algorithm gives incorrect output would be $<= 2^{-k}$ where k are the number of iterations .

We can prove this using some manipulation and from the Chinese remainder theorem .

Hence if the value of k is huge , then the probabililty that the output would be wrong would be negligible and hence if the input is random and not tampered, we can assume our algorithm to work correctly given we have chosen a reasonable value of k.