# Algorithms Analysis and Design

### Week 9 - Diary


### Ayan Agrawal (2020101034)


## Lecture 14 : Primality Testing

### Problems for the class :

- Randomized Algorithms
- Checking primality of a number by Miller-Rabin Testing Algorithm

## Randomized Algorithms :

Randomised algorithms are algorithms whose outputs are probably correct. The probability of success depends not on the randomness of the input but on the random choices made by the algorithm.

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.

Some *pros* and *cons* of using a randomized algorithms are :

### Pros :

1. There is a high possibility that we can get a good complexity for solving the problem since Randomized algos are pretty quick.
2. It's very difficult to track a randomized algo since it uses random numbers, so we can't easily track the pattern and it is difficult to predict which numbers have been chosen. Hence it is difficult to break a randomzied algorithm.

### Cons :

1. They aren't 100% accurate, we can have erroneous outputs.
2. In computer science, the generation of really random numbers is still an unsolved problem. As a result, if someone learns our method, he can easily build an input with the intent of breaking it.

Randomised algorithms are particularly very useful when we have a bunch of algorithms say $A_1, A_2, A_3, \ldots A_k$ such that for any input at least $2k/3$ values of $i$ are presen such that Algorithm $A_i$ solves the input correctly. If this can be ensured, then by using the majority circuit technique, by taking the majority in every case after forming groups of $3$ and recursively finding the final majority would always ensure we have the answer for that particular input.

# Miller- Rabin Primality Testing 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.

The known deterministic algorithms that we have found find out pretty efficiently, but before these algorithms were known, there was only the algorithm of Miller Rabin, that was randomised algorithm, and it provided the correct result with great accuracy and with great complexity too.

## Algorithm :

For a input $p$ :

1. If $p$ is even, also $p = 2$, then accept it else reject.This implies our algorithm determines the number isn't prime definitely.
2. Select $a_1, a_2, a_3 ..... a_k$ randomly from the Natural numbers.
3. For each $i$ from $1$ to $k$:
    - We can compute $a_i^{(p-1)} mod p$ and reject it if the value is different from $1$, as by Fermat's Little theorem prime numbers always give a value of $1$, thus any $p$ that does not give $1$ has to be composite.
    - Let $p - 1 = s * t$ , This can be done as $p$ has to be odd and thus it will always have an $s$ and $t$ such that $s$ is an odd number and $t$ is a exponent of 2.
    - Let $t = 2^h$, then we will try to compute the sequence, $a_i^{s*2^0}, a_i^{s*2^1}, a_i^{s*2^2} ..... a_i^{s*2^h} mod\ p$.
    - 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.
4. Number is now said to be prime **iff all these tests have passed**.

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 timess:
    pick a random integer a in [2,n-1]
    if(a^(n-1)% n != 1):
        return false
    write n-1 as (2^s)*d where d is odd
    int x = (a^s)%n
    if(x == 1 || x == n-1):
        continue
    repeat s-1 times:
        x <- x^2 % n
        if(x == n-1):
            continue Loop
    return false
return true
```

## Proof that probability of finding a prime is 1 for input to odd:

- In Step 6, Let's take the last number not equal to 1 as some number say $b$. Thus $b$ is not congruent to $1\ mod\ p$.
- Let's assume $p$ is prime and $b$ is not congruent to $-1\ mod\ p$ , which implies $p$ would be rejected and composite thus failing our algorithm, if we can prove this case is never seen, then we can be sure that $p$ as a prime number is always identified correctly.

  We know $b^2 \equiv 1\ mod\ p$ thus $b^2 - 1 \equiv 0\ mod\ p$

  $\implies (b-1)(b+1) \equiv 0\ mod\ p.$

  Clearly, $b-1$ and $b+1$ are greater than $0$ and less that $p$, and $p$ divides them perfectly as well. This would imply either $b-1 \equiv 0\ mod\ p$ or $b+1 \equiv 0\ mod\ p$ . This makes us arrive at contradiction that if $p$ is prime, then $b$ has to be congruent to either $1$ or $-1\ mod\ p$. Thus a prime is always identified as a prime perfectly by this algorithm, and thus $P[PRIME] = 1$.

## Proof that probability of finding a prime is 1 for input to 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 $k$.