

AKS Primality Test

AKS Primality Test is deterministically correct for any general number.

* Some features of the AKS Primality Test are:

- (1) It can be used to verify the primality of any general number given.
- (2) The maximum running time of the algorithm can be expressed as a polynomial over the no. of digits in the target number.
- (3) The algorithm is guaranteed to distinguish deterministically whether the target number is prime or composite.

* Idea of the Algorithm:

The algorithm based on the following lemma:

Let $n \in \mathbb{N}$, $n \geq 2$, $a \in \mathbb{Z}$ such that $\gcd(a, n) = 1$, then:

$$n \text{ is prime} \iff (x+a)^n \equiv x^n + a \pmod{n}$$

* Therefore AKS follows the following steps to test the primality of an integer n :

Given an input $n \in \mathbb{N}$

- (1) choose an integer a such that $\gcd(a, n) = 1$
- (2) calculate $f(x) = (x+a)^n - (x^n+a) \pmod{n}$
- (3) if $f(x) \equiv 0$, then " n is prime"
- (4) else " n is composite"

* The AKS test evaluates the equality by making complexity dependent on the size of r . This is expressed as:

$$(x+a)^n \equiv (x^n+a) \pmod{x^r-1, n}$$

✓ which can be expressed in simpler term as:

$$(x+a)^n - (x^n+a) = (x^r-1)g + nf$$

for some polynomials F and g .

* The congruence for the AKS primality test can be checked in polynomial time when r is polynomial to the digits of n .

* The AKS algorithm evaluates this congruence for a large set of values whose size is polynomial to the digits of n .

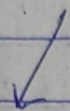
The proof of validity of the AKS algorithm shows that one can find r and a set of a values with the above properties such that if the congruences hold, then n is power of prime.

The brute force approach would require the expansion of the $(x-a)^n$ polynomial and a reduction $(\text{mod } n)$ of the $(n+1)$ coefficients.

* As a should be coprime to n , so we can implement this algorithm by taking $a=1$ for small values of n but for large values of n , take large values of a .

* The Algorithm is based on the condition that if n is any number then it is prime if

$$(x-1)^n - (x^n - 1) \text{ is divisible by } n$$



Ex: checking for $n=3$:

$$(x-1)^3 - (x^3 - 1)$$

$$= (x^3 - 3x^2 + 3x - 1) - (x^3 - 1)$$

$$= -3x^2 + 3x$$

divisible by $n=3$, Hence n is prime.

Python code for AKS Primality test

① `c = [0] * 100` # array used to store coefficients

② `def coeff(n):` # function to calculate the coefficients of $(x-1)^n - (x^n - 1)$ with help of Pascal's triangle

`c[0] = 1`

`for i in range(n):`

`c[i+1] = 1`

1
1 1
1 2 1
1 3 3 1

Page No.

Date / /

For j in range $(i, 0, -1)$:

$$c[j] = \cancel{c[j]} + c[j-1] - c[j]$$

$$c[0] = -c[0];$$

(3)

$\text{def isPrime}(n)$: # To check prime or not

$\text{coeff}(n)$ # calculating all the coefficients by the function coeff and storing all the coefficients in array c

(4)

$$c[0] = c[0] + 1$$

$$c[n] = c[n] - 1$$

subtracting $c[n]$ and adding $c[0]$ by 1 as:

$$(x-1)^n - (x^n - 1)$$

(5)

$$i = n$$

while $(i > -1 \text{ and } c[i] \% n == 0)$:

$$i = i - 1$$



checking all the coefficients whether they are divisible by n or not. if n is not prime, then loop breaks and $i > 0$

return
true if all coefficients
^ divisible by n

Page No.

Date / /

⑥ return True if $i < 0$ else False

⑦ # Driver code
 $n = \text{int}(\text{input}())$
if (isPrime(n)):
 print ("prime")
else:
 print ("composite")

* Runtime Analysis of the AKS Primality test:

Before finding out the running time of the Algorithm, we will look at the running time of the Euclidean algorithm and Perfect power test Algorithm which are:

→ The Runtime of the Euclidean Algorithm:

$$O(\log(n) \cdot \log(m))$$

where, $m, n \in \mathbb{Z}$

→ The Runtime of the Perfect Power Test algorithm is :

$$O^{\sim}(\log^3(n))$$

→ Now runtime Analysis of AKS algorithm :

(i) ~~Step 1~~

(i) Perform the perfect power test and the complexity is $O(\log^3(n))$.

(ii) In this step, the algorithm finds ~~the~~ the list r such that $O_r(n) > \log^2(n)$. There exists an r less than $\lceil \log^5(n) \rceil$.

The easiest way to find such r is simply to calculate $n^k \pmod{r}$ for $k=1, 2, \dots, \log^2(n)$. This involves $O(\log^2(n))$ multiplications modulo r for each r . This takes $O(\log^7(n))$ bit operations.

(iii) In this step, Algorithm computes (a, n) for $a=1 \dots r$ in order to determine whether $(a, n) > 1$ for some $a \leq r$. computing each gcd takes $O^{\sim}(\log^2(n))$ bit operations using Euclidean Algorithm.

resulting in total of $\tilde{O}(\log^7(n))$

(iv) In this step, Algorithm determines whether the congruence $(x+a)^n \equiv (x^n+a) \pmod{(x^r-1, n)}$ holds for a .

Each congruence takes $\tilde{O}(\log^7(n))$ bit operations to verify.

So After summing all the complexities, we get the overall complexity of the algorithm as:

$$\boxed{\tilde{O}(\log^{10.5}(n))}$$