# Sieve Of Atkins

Abstract:

The numbers which are divisible by only 1 and divisible by itself are knowns Prime numbers and all the integers excluding prime numbers are known as composite numbers. All the prime numbers are odd numbers. We can find the prime numbers by verifying only the odd numbers. A Primality test is done to verify whether a given number is prime or composite. There are various algorithms to find do the primality test. In general terms, ‘Sieve’ denotes ‘a device which filters the unwanted objects or things in our desired output’. Most of the prime generating algorithms, we filter out the composite numbers in the given range, subsequently we will be left with prime numbers.

# Sieve of Atkins

## Algorithms:

Naïve Algorithm:

1. Declare a Boolean array.
2. Initialize the prime numbers array with **true** or **false**.
3. In naïve Sieve of Atkins, we will compute the modulo 60 from each number N in the prime number array and assign the remainder to the variable **r**. (Consider x, y are some positive integers less the)
   1. If r is 1, 13, 17, 29, 37, 41,49, or 53, invert the Boolean value of the entry N in the prime number array, which in the form of (4 x2 + y2).
   2. If r is 7, 19, 31, or 43, invert the Boolean value of the entry N in the prime number array, which in the form of (3 x2 + y2).
   3. If r is 11, 23, 47, or 59, flip the Boolean value of the entry N in the prime number array, which in the form of (3 x2 - y2) for all y<x.
4. Now, we will be left with all the prime numbers with some extraneous numbers which are perfect square numbers.
5. To remove this square number multiples, we will run a loop untill our limit and mark them as composite numbers.
6. Iterate through the prime number array, we will get the required prime numbers.

Modified Algorithm:

Computing modulo-sixty, we will have {1, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 49, 53, 59}.

|  |  |
| --- | --- |
| Equations | Remainders |
| 4 x2 + y2 = N | 1,13, 17, 29, 37, 41, 49, 53 |
| 3 x2 + y2 = N | 7, 19, 31, 43 |
| 3 x2 - y2 = N | 11, 23, 47, 59 |

For equation **4 x2 + y2 = N**, when divide the remainders with 12. We get

1 % 12 = 1

13 % 12 = 1

17 % 12 = 5

29 % 12 = 5

37 % 12 = 1

41 % 12 = 5

49 % 12 = 1

53 % 12 = 5

When we do modulo operation (%) using 12, we get only two remainders {1,5}.

For equation **3 x2 + y2 = N**, when divide the remainders with 12.

7 % 12 = 7

19 % 12 = 7

31 % 12 = 7

43 % 12 = 7

We get only one remainders {7}.

For equation **3 x2 - y2 = N**, when divide the remainders with 12. We get only one remainder {11}.

The Sieve of Atkins can be modified as:

1. Declare a Boolean array.
2. Initialize the prime numbers array with **true** or **false**.
3. In naïve Sieve of Atkins, we will compute the modulo 12 from each number N in the prime number array and assign the remainder to the variable **r**. (Consider x, y are some positive integers less the)
   1. If r is 1 or 5, invert the Boolean value of the entry N in the prime number array, which in the form of (4 x2 + y2).
   2. If r is 7, invert the Boolean value of the entry N in the prime number array, which in the form of (3 x2 + y2).
   3. If r is 11, flip the Boolean value of the entry N in the prime number array, which in the form of (3 x2 - y2) for all y<x.
4. Now, we will be left with all the prime numbers with some extraneous numbers which are perfect square numbers.
5. To remove this square number multiples, we will run a loop untill our limit and mark them as composite numbers.
6. Iterate through the prime number array, we will get the required prime numbers.

## Psuedocode of Sieve of Atkins:

// assigning value to the range

range <- 1000000

// initialize the prime number array with default value

// considering 0 as a default value for prime number array

// Consider 0 indicates prime numbers and 1 indicates composite numbers

for i <- 0 to range+1 do

myPrime[i] <- 0

// a positive integer variable x and y are used for solving the equations

for x <- 1 by incrementing x by 1 until (x \* x) <= range:

for y <- 1 by incrementing y by 1 until (y \* y) <= range:

// compute N using equation 1

**N <- 4 x2 + y2**

If N%12 == 1 or N%12 == 5 where *N<=range* then:

// inverting the Nth index value

myPrime[N] =! myPrime[n];

**N <- 3 x2 + y2**

If N%12 == 7 where *N<=range* then:

myPrime[N] =! myPrime[n];

**N <- 3 x2 - y2**

If N%12 == 11 where *N<=range* and *x > y* then:

myPrime[N] =! myPrime[n];

// removal of perfect numbers

for n <- 5 incrementing n by 1 until *(n \* n)<=range* do:

// if it is still unmarked

If myPrime[n] == 0 then:

for x <- (n\*n) incrementing *x+=(n\*n)* until *(x < range)* do:

// marking as composite numbers

myPrime[x] = 1;

// printing prime numbers

print 2, 3, 5

for n <- 7 incrementing by 2 until *n <= range* do:

if myPrime[n] == 1 then:

print n

// end of the code

## Programs:

### Java Program:

public class sieveOfAtkins

{

public static void main(String[] args) {

int limit = 10000000;

boolean[] list = new boolean[limit];

// taking Boolean array for marking up the composite numbers.

int x;

int y;

int n;

for (x=1; (x\*x) <= limit; x++)

{ // for loop of variable x starts here

for(y=1; (y\*y) <= limit; y++)

{ // for loop of variable y starts here

int sqX = x\*x;

int sqY = y\*y;

n = 4 \* sqX + sqY;

if (n<=limit && (n%12==1 || n%12 == 5))

{

// inverting the list[n] value using XOR

list[n]^=true;

}

// n = 3 \* sqX + sqY;

n-=sqX;

if (n<=limit && (n%12==7))

{

list[n]^=true;

}

// n = 3 \* sqX - sqY;

n-=2\*sqY;

if(x>y && n<=limit && (n%12==11))

{

list[n]^=true;

}

} // for loop of variable x starts ends here

} // for loop of variable x starts ends here

// for loop for removal of square numbers

for(n=5;(n\*n) <= limit; n++)

{

if(list[n])

{

for (int i = n\*n; i<limit; i+=n\*n)

{

list[i]=false;

}

}

}

int count=2; // As we are starting from 5, make count as 2 {2,3}

for(n=5; n<limit ;n+=2)

{

if(list[n]) count++;

}

System.out.println(count);

}

}

### C++ program:

#include <iostream>

using namespace std;

int main()

{

long long int limit = 1000000;

bool list[limit]={0};

long long int x;

long long int y;

long long int n;

for(x=1;(x\*x)<=limit;x++)

{

for(y=1;(y\*y)<=limit;y++)

{

long long int sqX = x\*x;

long long int sqY = y\*y;

n = 4 \* sqX + sqY;

if(n<=limit && (n%12==1 || n%12 == 5))

{

list[n]^=true;

}

// n = 3 \* sqX + sqY;

n-=sqX;

if(n<=limit && (n%12==7))

{

list[n]^=true;

}

// n = 3 \* sqX - sqY;

n-=2\*sqY;

if(x>y && n<=limit && (n%12==11))

{

list[n]^=true;

}

}

}

for(n=5;(n\*n)<=limit;n++)

{

if(list[n])

{

for(long long int i = n\*n;i<limit;i+=n\*n)

{

list[i]=false;

}

}

}

long long int count=2;

for(n=5;n<limit;n+=2)

{

if(list[n]) count++;

}

cout << count << endl;

return 0;

}