**SIEVE OF SUNDRAM**

The Prime Numbers -So simple, yet so mysterious.

There is no general formula which can generate all the primes. Their distribution among the other numbers is still a mystery. However, we do have algorithms that can find the prime numbers in certain range. this article we will look at an example of such an algorithm, in which the prime numbers pop out as a very simple as if by a magic. It is called Sieve of Sundaram.

Sieve of Sundaram is one of the best-known approaches to generate the prime numbers in certain range. Along the optimization approaches which generates primes Sieve of Sundaram generates the prime numbers very quickly. The Sieve of Sundaram is named after its creator S.P.Sundaram an Indian Mathematician who introduced the algorithm in the year 1931.

In Mathematics, the sieve of Sundaram is a variant of Sieve of Eratosthenes, a simple deterministic algorithm for finding all prime numbers .It is a better way of sieving out all the unwanted numbers and finding the Primes.

**ALGORITHM**

Step1): Start with the list of numbers from 1 to n.

Step2): In general it produces prime numbers smaller than 2\*x+2 for the given x, so we reduce (n-1) to half.

Step3): Create an array of size (n-1)/2 which can be used to separate the

prime and composite numbers.

Step4): From The list remove all the numbers of the form i+j+2\*i\*j

Where 1<=i<=j and i+j+2\*i\*j<=(n-1)/2.

Step5): Continue the process until i moves from 1 to (n-1)/2.

Step6): Remaining numbers are doubled and incremented by 1 which

Generates the list of primes . Add 2 to the array.

WE ARE DONE!

**EXAMPLE**

1): Generating the prime Numbers which are less than 50.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
| 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 |
| 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 |
| 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 | 50 |

2): Using the condition i+j+2\*i\*j<=N and 1<=1<=j start eliminating the

Numbers.

3): With i=1 eliminating numbers which satisfies i+j+2\*i\*j<=n condition.

Where n=(N-1)/2 and i<=n.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
| 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 |
| 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 |
| 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 | 50 |

4): With i=2 eliminating numbers which satisfies i+j+2\*i\*j<=n condition.

Where n=(N-1)/2 and i<=n.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
| 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 |
| 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 |
| 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 | 50 |

5): With i=3 eliminating numbers which satisfies i+j+2\*i\*j<=n condition.

Where n=(N-1)/2 and i<=n.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
| 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 |
| 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 |
| 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 | 50 |

6): With i=4 eliminating numbers which satisfies i+j+2\*i\*j<=n condition.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
| 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 |
| 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 |
| 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 | 50 |

7): with i=5 the condition i+j+2\*i\*j<=!n (fails).

8): So we are done!

9): List the unmarked elements from the list.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 |  | 5 | 6 |  | 8 | 9 |  |
| 11 |  |  | 14 | 15 |  |  | 18 |  | 20 |
| 21 |  | 23 |  |  | 26 |  |  | 29 | 30 |
|  |  | 33 |  | 35 | 36 |  |  | 39 |  |
| 41 |  |  | 44 |  |  |  | 48 |  | 50 |

10): Express the elements in the list as 2\*i+1. Then the numbers

becomes:

3 5 7 11 13 17 19 23 29 31 37 41 43 47

11): Add 2 to your list then the list becomes:

2 3 5 7 11 13 17 19 23 29 31 37 41 43 47

**IMPLEMENTATION**

**JAVA PROGRAM:**

class sundaram {

public static void main(String[] args)

{

int num=10000000;

int range=(num-1)/2;

int count=1;

boolean array[]=new boolean[range+1];

for(int check1=1;check1<=range;check1++)

{

int check2=check1;

long limit = check1+check2+2l\*check1\*check2;

while(range>=limit)

{

array[(int)limit]=true;

check2+=1;

limit=check1+check2+2l\*check1\*check2;

}

}

for(int index=1;index<range;index++)

{

if(array[index]==false)

{

count+=1;

}

}

System.out.println(count);

}

}

C++ Program:

#include<iostream>

using namespace std;

int main()

{

long long int num=1000000;

long long int range=(num-1)/2;

bool array[range+1]={false};

long long int check2=0,check1=0,count=0;

long long int index=0;

for(check1=1;check1<=range;check1+=1)

{

check2 = check1;

unsigned long long int limit=check1+check2+2\*check1\*check2;

// cout<<"fgfgdf"<<limit;

while(limit<=range)

{

array[limit]=true;

check2+=1;

limit=check1+check2+2l\*check1\*check2;

}

}

for(index=0;index<range;index+=1)

{

if(array[index]==false)

count+=1;

}

cout<<count;

return 0;

}

**PYTHON PROGRAM:**

num=10000000

range1=(num-1)//2

list=[True]\*(range1+1)

for check1 in range(1,range1+1):

check2=check1

while (check1+check2+2\*check1\*check2)<=range1:

list[check1+check2+2\*check1\*check2]=False

check2+=1

count=1

for i in range(1,range1+1):

if list[i]==True:

count+=1

print(count)

**TIME COMPLEXITIES**:

|  |  |  |  |
| --- | --- | --- | --- |
| RANGE | JAVA (sec) | PYTHON (sec) | No of Primes |
| 10^1 | 0.00 | 0.00 | 4 |
| 10^2 | 0.00 | 0.00 | 25 |
| 10^3 | 0.00 | 0.0003 | 168 |
| 10^4 | 0.001 | 0.0035 | 1229 |
| 10^5 | 0.004 | 0.040 | 9592 |
| 10^6 | 0.015 | 0.487 | 78498 |
| 10^7 | 0.057 | 5.8 | 664579 |