Abstract:

Apart from **Sieve of Eratosthenes**, **Sieve of Atkin**, **Sieve of Sundaram**, **wheel factorization**, and many more algorithms to generate prime numbers, we can **use a new algorithm to generate prime numbers in support with some new equations.**

Introduction:

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	51	52	53	54	55	56
57	58	59	60	61	62	63
64	65	66	67	68	69	70

Table:1 Some numbers representing pattern of prime numbers.

From Table.1, we can see that prime numbers > 5, are occuring in pattern. We can trace equations:

$$y=6x-1 \qquad \text{. ….. ...i.}$$

$$y=6x+1 \qquad \text{…….ii}$$

Equation i, can help to trace prime numbers such as (5,11,17,23,29…..). Similarly, ***Equation ii***, can help to trace the prime numbers such as (7,13,19, 31, 37….).

****Note:**

We see that in the course of generating the prime numbers we come across numbers such as (25, 35, 125….),in fact these numbers are composite numbers. Let's call these numbers as *pseudo prime numbers*.

Let's consider, A, B, C be any non empty sets. Such that,

A={x | x ∈ prime numbers or pseudo prime numbers}

B={x | x ∈ pseudo prime numbers}

C=A-B (Set Difference)

C={x | x ∈ prime numbers}

As we can see, from the set of prime numbers and pseudo prime numbers (Set A), if we separate all the pseudo prime numbers(Set B) , we can get all the prime numbers(Set C). The challenge is to get all the pseudo prime numbers.

In order to get all, pseudo prime numbers < **n**, where **n** is the number upto which prime numbers are to be generated, we can use equations.

Equation i.

t1=6*i*j;

temp1= t1+7*i;

Equation ii.

temp2=t1+(i*i);

Equation iii.

temp3=temp2 +(12*j) +4*(i+1);

Equation iv.

temp4= temp3 + 4*(i+2);

Where for each iteration, i=5, (5+6), (5+6+6), (5+6+6+6) and j=0,1,2,3,4,5……for all i.

For example:

Taking i=5 and j=2;

t1=60,
temp1=t1+35=95, which is a pseudo prime number.
temp2= 60+25=85, which is again a pseudo prime number.
temp3= 85 + 24+24=133 which is again a pseudo prime number.
temp4 =133 +28=161, which is again a pseudo prime number.

When all the equations calculate pseudo prime numbers greater than **n,** we get an indication for breaking the loop and check with the next **i.**

In this way, we trace out all the pseudo prime numbers less than **n.** Now, from the set of prime and pseudo prime numbers when we eliminate pseudo prime numbers, we get all prime numbers.

The code Implementation:

```
/*C++ Program to Illustrate Rajan's Algorithm to Generate Prime Numbers up to Nth Number.
*for example:
*if N=10
*Output: 2,3,5,7
*/

#include<algorithm>
#include<string.h>
#include<stdlib.h>
#include<time.h>
#include<iostream>
#include<fstream>

using namespace std;

typedef long long int lli;

bool *arr;

int main()
{
	lli n,count=0;

	scanf("%lld", &n);
	clock_t t;
	t=clock();

	ofstream fout;
	fout.open("Result_RajanAlgo.txt",ios::app);

	arr =(bool*)calloc(n+1,1);
	bool *arry=(bool*)calloc(n+1,1);

	if(!arr || ! arry)
	{
		printf("Memory Not Located");
	}

	memset(arry,1,n+1);

	lli d=5;
```

```
//Array Initialization using 2 equations


while(d<=n)
{
memset(arr+d,1,1);
memset(arr+(d+2),1,1);
d=d+6;
}

//Loop to Rule out all the Pseudo-Prime Numbers using 4 equations.

for(lli i=5 ; i*i<=n ; i=i+6)
{
        lli j=0;
                while(1)
                {
                        int flag=0;
                        /*Equation i.*/
                        lli t1=6*i*j;

                        lli temp1= t1+7*i;
                        /*Equation ii.*/
                        lli temp2=t1+(i*i);
                        /*Equation iii.*/
                        lli temp3=temp2 +(12*j) +4*(i+1);
                        /*Equation iv.*/
                        lli temp4= temp3 + 4*(i+2);

                        if(temp1<=n)
                        {
                                arry[temp1]=0;
                        }

                        else
                        {
                                flag++;
                        }
                        if(temp2<=n)
                        {
                                arry[temp2]=0;
                        }
```

```
                        else
                        {
                                flag++;
                        }
                                if(temp3<=n)
                        {
                                arry[temp3]=0;
                        }
                        else
                        {
                                flag++;
                        }
                        if(temp4<=n)
                        {
                                arry[temp4]=0;
                        }
                        else
                        {
                                flag++;
                        }

                        if(flag==4)
                        {
                                break;
                        }
                        j++;
                }
}

if(n>=2)
{
        cout<<"2"<<" ";
        fout<<to_string(2)<<"\t";
        count++;
}

if(n>=3)
{
        cout<<"3"<<" ";
        fout<<to_string(3)<<"\t";
        count++;
}
```

```
        //Printing Prime Numbers up to nth number

        for(lli p=5; p<=n; p=p+6)
        {
                if(arry[p]==1 && arr[p]==1)
                {

                        cout<< p << " ";
                        fout<<to_string(p)<<"\t";
                        count++;
                }

                if(arry[p+2]==1 && arr[p+2]==1)

                {
                        cout<< p+2 << " ";
                        fout<<to_string(p+2)<<"\t";
                        count++;

                }
        }

        t=clock()-t;
        double time=(double (t))/CLOCKS_PER_SEC;

cout<<"\n";
//Printing the total prime numbers from 1 to nth number.
cout<<count<<endl;
cout << "Time taken by function: "<<time<<" Seconds"<<endl;
fout.close();
        return 0;
}
```

Code Implementation of **Sieve of Eratosthenes**:

```
// C++ program to print all primes smaller than or equal to
// n using Sieve of Eratosthenes
#include <bits/stdc++.h>
using namespace std;
#include <algorithm>
#include<stdlib.h>
#include<fstream>

typedef long long int lli;
void SieveOfEratosthenes(int n)
{
   // Create a boolean array "prime[0..n]" and initialize
   // all entries it as true. A value in prime[i] will
   // finally be false if i is Not a prime, else true.
   bool* prime=(bool*)calloc(n+1,1);
   //bool prime[n+1];
   lli count=0;


   memset(prime,1,n+1);
           ofstream fout;
           fout.open("Result_Eratosthenes.txt",ios::app);



   for (int p=2; p*p<=n; p++)
   {
      // If prime[p] is not changed, then it is a prime
      if (prime[p] == 1)
      {
         // Update all multiples of p
         for (int i=p*2; i<=n; i += p)
            //cout<< i << "\v";
            prime[i] = 0;


      }
    //cout<<"\n";
   }

   // Print all prime numbers
  for (int p=2; p<=n; p++)
```

```
    {
      if (prime[p]==1)
      {
       count++;
       cout << p << " ";
            fout<<to_string(p)<<"\t";

      }
    }
        cout<<"\n"<<count;
                    fout.close();
}

// Driver Program to test above function
int main()
{

    lli n ;
    cin>>n;


   clock_t t;
            t=clock();
    SieveOfEratosthenes(n);
            t=clock()-t;

            double time=(double (t))/CLOCKS_PER_SEC;
       cout<<"\n";
       cout << "Time taken by function: "<<time<<" Seconds"<<endl;

    return 0;
}
```

Time Comparison in seconds:

Value of n	Rajan's Algorithm	Sieve of Eratosthenes
10	0.000236	0.000612
100	0.000282	0.000500
1000	0.000470	0.000927
10000	0.002642	0.002889
100000	0.007502	0.009627
1000000	0.070071	0.090379
10000000	0.279753	0.35643
100000000	2.74204	3.24373
1000000000	28.001	32.8393