ALGORITMA PENCARIAN BILANGAN PRIMA

Gok Asido Haro – NIM: 13505072

Program Studi Teknik Informatika, Institut Teknologi Bandung

Jl. Ganesha 10, Bandung

E-mail : if15072@students.if.itb.ac.id

ABSTRAK

Untuk waktu yang lama, teori bilangan secara keseluruhan, dan studi bilangan prima khususnya, terlihat sebagai contoh yang resmi dari matematika murni, tanpa aplikasi di luar keterarikan diri sendiri untuk mempelajari topik ini. Akhirnya, pandangan ini dihancurkan pada tahun 1970-an, saat diumumkan bahwa bilangan prima dapat digunakan sebagai basis dari penciptaan algoritma kriptografi kunci publik. Bilangan prima juga digunakan untuk *hash table* dan pembangkit bilangan semi-acak (*pseudorandom number generators*).

Bilangan prima adalah permasalahan yang serius didalam ilmu komputer dan teori bilangan. Hal tersebut sangat dibenarkan dalam bidang kriptografi, saat protokol-protokol enkripsi kunci publik didasarkan pada penggunaan dari bilangan-bilangan prima dengan ukuran besar, dan keamanannya didasarkan pada kesulitan untuk mendapatkan faktor-faktor prima dari suatu bilangan prima yang sangat besar.

Bilangan prima dapat dibangkitkan dengan proses penyaringan (*sieving*) seperti *Sieve of Eratosthenes* dan *Sieve of Atkin.* Bilangan prima memberikan banyak properti-properti yang aneh dan menakjubkan, bilangan prima disusun menjadi begitu luas dan bilangan prima segikir merupakan nilai praktikal.

Kata Kunci : bilangan prima, algoritma, *Brute Force, Sieve of Eratosthenes, Sieve of Atkin,* fungsi, java.

1. PENDAHULUAN

Bilangan prima adalah permasalahan yang serius didalam ilmu komputer dan teori bilangan. Hal tersebut sangat dibenarkan dalam bidang kriptografi, saat protokol-protokol enkripsi kunci publik didasarkan pada penggunaan dari bilangan-bilangan prima dengan ukuran besar, dan keamanannya didasarkan pada kesulitan untuk mendapatkan faktor-faktor prima dari suatu bilangan prima yang sangat besar.

Dalam jaman modern ini, hampir untuk keseluruhan penggunaan komputer, secara praktiknya setiap orang dalam sebuah masyarakat dengan teknologi yang sudah maju selalu berusaha untuk mendapatkan akses pada teknologi kriptografi yang paling *up-to-date*, yang disebut kriptosistem kunci publik RSA. Bagian yang penting dari sistem kriptografi ini adalah faktorisasi bilangan yang sangat besar menjadi faktor bilangan primanya. Dengan demikian sebuah konsep teori bilangan yang kuno sekarang memainkan suatu peran yang krusial dalam komunikasi antar jutaan manusia yang memiliki sedikit atau bahkan tidak memiliki kemampuan dasar matematika.

Bilangan prima juga sangat bermanfaat untuk kegunaan-kegunaan yang lainnya. Contohnya antara lain adalah *hash table*, dimana *hash table* paling bagus jika diinisialisasi dengan bilangan prima agar dapat meminimumkan kolisi (*collisions*) yang akan terjadi.

2. BILANGAN PRIMA

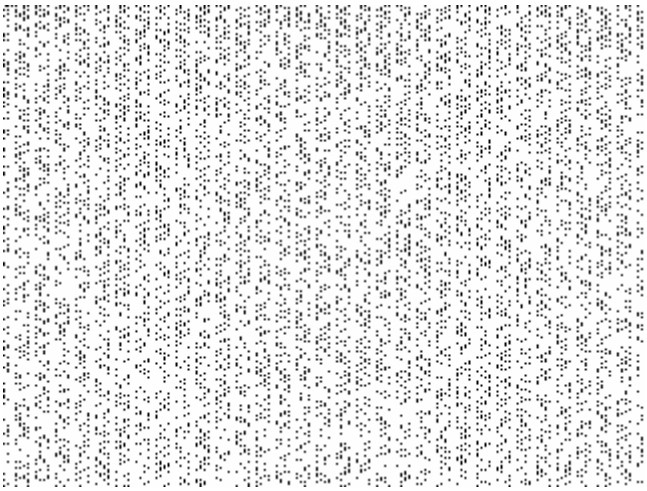
Bilangan prima (atau integer prima, sering secara singkat disebut "prima") adalah sebuah integer positif $p > 1$ yang tidak memiliki pembagi integer positif selain 1 dan p itu sendiri. (Lebih singkatnya, sebuah bilangan prima p adalah sebuah integer positif dengan tepat satu pembagi positif selain 1). Sebagai contoh, satu-satunya pembagi 13 adalah 1 dan 13, hal ini membuat 13 adalah sebuah bilangan prima. Sementara itu angka 24 memiliki pembagi 1, 2, 3, 4, 6, 8, 12, 24 (sesuai dengan faktorisasi dari $24 = 2^3.3$), hal ini membuat 24 bukanlah sebuah bilangan prima. Integer positif selain 1 yang bukan bilangan prima disebut dengan bilangan komposit (*composite numbers*).

Dengan demikian, bilangan prima adalah bilangan-bilangan yang tidak bisa difaktorkan, lebih singkatnya adalah bilangan-bilangan n yang pembaginya sedikit dan hanya 1 dan n.

Angka 1 sendiri adalah kasus spesial dimana 1 dianggap bukanlah prima dan bukan komposit. Walaupun angka 1 dulu

biasanya dianggap sebagai bilangan prima, angka 1 membutuhkan perlakuan spesial dalam banyak definisi-definisi dan aplikasi-aplikasi yang melibatkan bilangan prima.yang lebih besar atau sama dengan 2 yang biasanya dikelaskan sendiri.

Dengan angka 1 dikeluarkan, maka bilangan prima terkecil adalah 2. Bagaimanapun, karena 2 adalah satu-satunya bilangan prima yang genap (yang, secara ironis, terasa bahwa 2 merupakan bilangan prima "paling ganjil"), angka 2 juga menjadi spesial, dan keseluruhan bilangan prima tanpa 2 kemudian disebut sebagai "bilangan prima ganjil".



Gambar 1. Persebaran dari semua bilangan prima dari 1 sampai 76.800

Gambar 1 menggambarkan pola persebaran dari semua bilangan prima dari 1 sampai 76.800. Setiap *pixel* mewakili setiap bilangan dengan *pixel* hitam berarti bilangan yang prima dan *pixel* putih mewakili bilangan yang tidak prima.

Permasalahan memodelkan distribusi dari bilangan-bilangan prima merupakan sebuah subjek yang popular dari penelitian-penelitian di bidang teori bilangan. Bilangan-bilangan prima didistribusikan di antara bilangan natural dalam sebuah (sangat jauh) cara yang tidak bisa diprediksi, tetapi sepertinya terdapat hokum-hukum yang mengatur kelakuan bilangan ini. Leonhard Euler berkomentar

> "*Seorang matematikawan mencoba dengan sia-sia dalam hari-harinya untuk menemukan beberapa pengaturan dalam urutan bilangan-bilangan prima dan kita memiliki alas an untuk mempercayai bahwa hal ini merupakan sebuah misteri dimana pikiran tidak akan bisa mendekatinya.* "

Untuk waktu yang lama, teori bilangan secara keseluruhan, dan studi bilangan prima khususnya, terlihat sebagai contoh yang resmi dari matematika murni, tanpa aplikasi di luar keterarikan diri sendiri untuk mempelajari topik ini. akhirnya, pandangan ini dihancurkan pada tahun 1970-an, saat diumumkan bahwa bilangan prima dapat digunakan sebagai basis dari penciptaan algoritma kriptografi kunci publik. Bilangan prima juga digunakan untuk *hash table* dan pembangkit bilangan semi-acak (*pseudorandom number generators*).

Beberapa mesin-mesin dengan baling-baling dirancang dengan jumlah pin yang berbeda untuk setiap baling-baling, dengan jumlah pin pada setiap baling-baling prima, atau ko-prima terhadap jumlah pin dari baling-baling yang lain. Hal ini membantu membangkitkan lingkaran penuh dari posisi-posisi baling-baling yang mungkin sebelum memulai posisi lain.

3.ALGORITMA PENCARIAN BILANGAN PRIMA

3.1 Algoritma *Brute Force*

Tidak ada tetapan dalam pencarian bilangan prima menggunakan algoritma *Brute Force.* Algoritma di bawah ini adalah algoritma *Brute Force* (cara naif) dan merupakan algoritma yang sederhana.

Pertama-tama kita membuat suatu *list* yang akan diisi dengan bilangan prima yang kita dapatkan. Pada awalnya di dalam *list* tersebut tidak ada bilangan lain selain 3. Kemudian kita akan mengecek seluruh bilangan ganjil sampai dengan batas yang telah kita tentukan sendiri, dan membandingkannya dengan bilangan prima yang ada dalam *list* bilangan prima tadi. Jika bilangan tersebut tidak bisa dibagi oleh seluruh bilangan prima yang ada dalam *list* maka bilangan tersebut adalah bilangan prima dan kita bisa menambahkannya ke dalam *list* bilangan prima tadi. Pada akhirnya kita harus menambahkan angka 2 ke dalam *list* tersebut. Satu hal lainnya yang patut dicatat, kita hanya harus mengecek bilangan prima sampai akar kuadrat dari batas nilai dari *list* tersebut.

Hal ini terlihat baik, sebagai sebuah pendekatan kasar terhadap fungsi perhitungan bilangan prima adalah x/ln x, dan kita hanya perlu mengecek bilangan prima sampai √(n). Secara kasar kita harus melakukan paling banyak √(n)/ln√(n) tes terhadap suatu angka untuk menentukan dengan keakuratan 100% bahwa bilangan tersebut adalah prima. Fungsi ini memang akan memakan waktu yang lama, tapi pada kenyataannya tidak teralalu lama. Berikut kodenya dalam *pseudocode*:

```
procedure deterministicPrimeSeqence(end):
      if end < 2: return []
      if end < 3: return [2]
      primes = [3]
      for i in range(5,end,2):
          sqt = int(sqrt(i))
          for prime in primes:
              if prime > sqt:
                    primes.append(i)
                    break
               if i%prime == 0: break
      primes.insert(0,2)
      return primes
```

3.2 Algoritma *Sieve Of Eratosthenes*

Eratosthenes (276-194 S.M) adalah petugas perpustakaan ketiga dari perpustakaan terkenal di Alexandria dan adalah seorang sarjana yang sangat hebat. Eratosthenes dikenang dengan pengukurannya terhadap keliling dari dari bumi, memperkirakan jarak antara bumi dengan matahari dan bulan, dan, dalam matematika, untuk penemuan dari sebuah algoritma untuk mencari bilangan-bilangan prima, yang dikenal sebagai Algoritma *Sieve Of Eratosthenes.*

Sieve Of Eratosthenes adalah sebuah algoritma klasik untuk menemukan seluruh bilangan prima sampai ke sebuah N yang ditentukan. Mulai dengan *array of integer* yang belum dicoret dari 2 ke N. Integer pertama yang belum dicoret yaitu 2, adalah bilangan prima pertama. Coret seluruh kelipatan dari bilangan prima ini. Ulangi pada integer selanjutnya yang belum dicoret.

Sebagai contoh, berikut adalah *array* pada awalnya:

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

Karena 2 belum dicoret, maka 2 adalah bilangan 2 pertama kita. Kita coret seluruh kelipatan 2, yaitu 4, 6, 8, 10, 12, dst.

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

Integer selanjutnya yang belum dicoret adalah 3, maka 3 adalah prima dan kita coret seluruh kelipatan 3, seperti 6, 9. 12, dst.

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~~

5 adalah bilangan prima selanjutnya dan kita mencoret seluruh kelipatan 5. Satu-satunya bilangan yang dicoret dalam *range* ini adalah 25.

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~~

Maka kita mendapatkan bilangan prima selanjutnya yaitu 7, 11, 13, 17, 19, dan 23.

Kode dalam *pseudocode* adalah:

```
// tentukan batas
limit ← 1.000.000

// set semua bilangan dengan true
is_prime(i) ← true, i □ [2, limit]

for n in [2, √limit]:
  if is_prime(n):
    is_prime(i) ← false,
        i □ {n², n²+n, n²+2n, ..., limit}

for n in [2, limit]:
    if is_prime(n): print n
```

Atau lebih sederhananya:

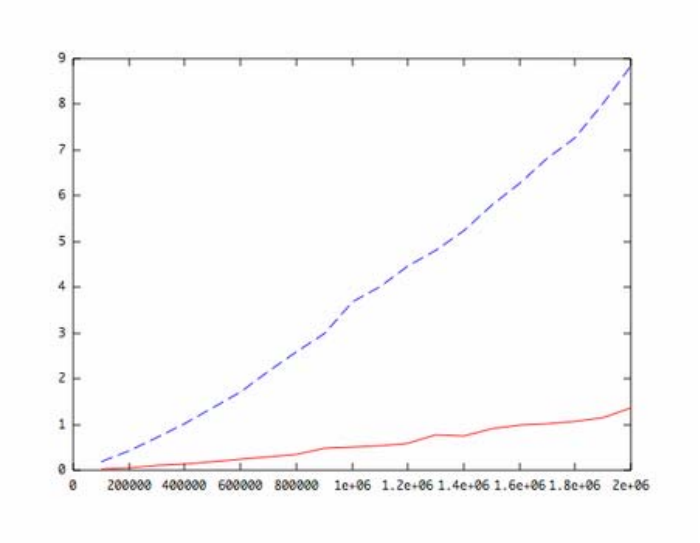
```
limit = 1000000
sieve$ = string "P" with batas panjang

prime = 2
repeat while prime² < limit
    set karakter pada index dari setiap
kelipatan bilangan prima (di luar bilangan
tersebut *1) dalam sieve$ menjadi "n"

    prime = index dari setiap "p" dalam sieve
$
end repeat
```

Setelah penulis mencoba untuk menerjemahkan ke dalam potongan kode dalam bahasa pemrograman Java, maka kode untuk algoritma *Sieve Of Eratosthenes* adalah :

```
class primaEratosthenes{
public static void main(String args[]){
int m=10000;//batas
boolean[] prima=new boolean[m+1];

for(int i=0; i<=m; i++)
   {
      prima[i]=true;
   }

prima[0]=prima[1]=false;
double akarN=Math.sqrt(m);//akar
kuadrat dari n

//coret bilangan yang bukan prima
for(int i=2; i<=akarN; i++)
  {
     if (prima[i])
       {
         for (int j=i*i; j<=m; j++)
            {
               if ((j%i)==0)
                  {
                     prima[j]=false;
                  }
            }
       }
   }

//tampilkan seluruh bilangan prima
for(int i=0; i<=m; i++)
   {
     if (prima[i]&&(i>=n))
      System.out.println (i + "\n");
   }
  }
}
```

Dengan kode implementasi dalam bahasa Java seperti di atas maka dapat kita dapatkan kecepatan dari algoritma ini untuk mendapatkan urutan bilangan prima dari suatu batas yang sudah ditentukan sebelumnya.

Dengan perhitungan secara kasar, algoritma ini membutuhkan 1 detik untuk mendapatkan urutan bilangan prima dengan batas atas 2000000. Berikut adalah perbandingan kecepatan Algoritma *Sieve of Eratosthenes* dengan algoritma *Brute Force* (naif):



Gambar 2 Perbandingan Algoritma *Sieve of Eratosthenes* dengan Algoritma *Brute Force* (naif)

Pada gambar di atas, garis biru adalah metode pencarian dengan algoritma *Brute Force* dan garis merah adalah metode prncarian dengan algoritma *Sieve of Eratosthenes*. Terlihat perbedaan yang sangat jauh antara kedua metode ini.

Metode *Sieve of Eratosthenes* sangat cepat, sehingga tidak ada alas an untuk menyimpan daftar bilangan prima yang besar pada komputer, karena implementasi yang efisien dari algoritma ini dapat mendapatkan bilangan-bilangan tersebut lebih cepat daripada komputer harus membacanya dari media penyimpanan. Pada faktanya permasalahan dengan algoritma seperti yang disajikan di atas tidaklah selalu tentang kecepatan (algoritma ini menggunakan $\Theta(n(\log n))$ log log n) bit operasi), tetapi lebih ke penggunaan tempat ($\Theta(n)$).

3.3 Algoritma *Sieve of Atkins*

Dalam matematika, algoritma *Sieve of Atkin* adalah sebuah algoritma yang cepat dan modern untuk mendapatkan seluruh bilangan prima sampai pada suatu batas integer yang telah ditentukan. Algoritma ini adalah versi optimasi dari algoritma kuno *Sieve of Eratosthenes*. *Sieve of Atkin* melakukan dahulu beberapa pekerjaan persiapan dan kemudian mencoret kelipatan dari kuadrat bilangan prima, bukannya kelipatan bilangan prima. Algoritma ini diciptakan oleh A. O. L. Atkin dan Daniel J. Bernstein pada tahun 2004.

Pada algoritma ini, suatu bilangan dapat disebut bilangan prima jika bilangan tersebut memenuhi beberapa persyaratan, yaitu:

1. Persamaan berikut ini harus memiliki jumlah solusi n yang ganjil:

   a. $4x^2 + y^2 = n$ dimana n mod 4 =1

   b. $3x^2 + y^2 = n$ dimana n mod 6 =1

   c. $4x^2 - y^2 = n$ dimana n mod 12 =11

2. Bilangan tersebut haruslah *square-free*. Sebuah bilangan yang disebut *square-free* adalah bilangan yang tidak memiliki faktor prima lebih dari sekali. (Contoh: 20=2*2*5, 20 memiliki 2 buah faktor 2 sehingga bukan merupakan *square-free*. 21=7*3, 21 tidak memiliki faktor yang dobel dan merupakan bilangan yang *square free*).

Cara implementasi dari algoritma ini adalah:

1. Ciptakan sebuah *array of Boolean* dengan ukuran n dan seluruh elemen diset menjadi *false*

2. Ulangi melalui setiap nilai valid untuk x dan y, dan balikkan elemen dalam posisi hasil perhitungan (Contoh: x = 3, y = 5, $4x^2 + y^2 = 61$ maka apapun nilai elemen ke 61 dari *array of Boolean* tadi harus dibalik).

3. Langkah terakhir adalah dengan mencoret semua bilangan yang tidak *square-free*. Hal ini dapat dilakukan dengan memulai pada awal dari *array of Boolean*, dan jika elemen pertama bernilai *true* (yang berarti bilangan tersebut prima) maka set seluruh elemen yang lain yang merupakan kelipatan dari kuadrat bilangan tersebut menjadi *false*. Lakukan untuk semua elemen kurang dari akar dari batas angka.

4. Elemen yang berisi nilai *true* mewakilkan bilangan prima.

Untuk algoritma ini *pseudocode*-nya adalah:

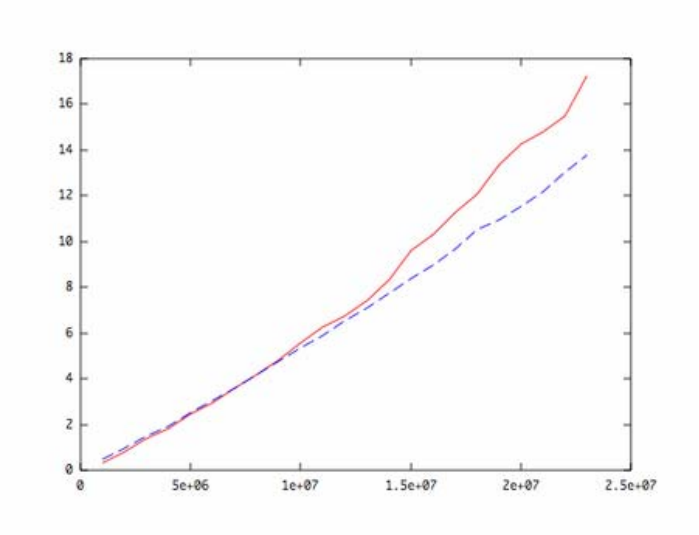
```
batas ← 1.000.000

prima(i) ← false, i □ [5, batas]

for (x, y) in [1, √batas] × [1, √batas]:
  n ← 4x²+y²
  if (n ≤ batas) □ (n mod 12 = 1 □ n mod
12 = 5):
        prima(n) ← ¬prima(n)
  n ← 3x²+y²
    if (n ≤ batas) □ (n mod 12 = 7):
        prima(n) ← ¬prima(n)
  n ← 3x²-y²
    if (x > y) □ (n ≤ batas) □ (n mod 12 =
11):
        prima(n) ← ¬prima(n)

for n in [5, √batas]:
  if prima(n):
    prima(k) ← false, k □ {n², 2n², 3n²,
..., batas}

print 2, 3
for n in [5, batas]:
    if prima(n): print n
```

Penulis menterjemahkan *pseudocode* di atas menjadi sebuah program dalam bahasa java dan kodenya adalah berikut:

```
boolean[] prima = new boolean [atas+1];
prima[0]=prima[1]=prima[4]=false;
prima[2]=prima[3]=true;

for(int i=5;i<=atas;i++)
  {
     prima[i]=false;
  }

double akarN=Math.sqrt(atas);     //akar
kuadrat dari n

int n;
for (int i=1;i<=akarN;i++)
   {
      for(int j=1;j<=akarN;j++)
         {
            n=(4*(i*i))+(j*j);
```

```
          if((n<=atas)&&(((n % 12)==1) ||
       ((n % 12)==5)))                {

       prima[n]=!prima[n];
             }

       n=(3*(i*i))+(j*j);
       if((n<=atas)&&((n % 12)==7))
          {
             prima[n]=!prima[n];
          }

       n=(3*(i*i))-(j*j);
       if((i>j)&&(n<=atas) &&
((n%12)==11))
         {
            prima[n]=!prima[n];
          }
       }
     }
     for (int a=5;a<=akarN;a++)
       {
          if(prima[a])
             {
               int j=a*a;
               int m=j;
               int l=1;
               while(m<=atas)
                 {
                   prima[m]=false;
                   l++;
                   m=l*j;
                 }
             }
       }
    for(int b=0; b<=atas; b++)
       {
          if (prima[b])
            System.out.println(b + "\n");
       }
```

Algoritma ini menghitung bilangan prima sampai N dengan menggunakan $\Theta(N/\log \log N)$ operasi dengan hanya $N^{1/2+o(1)}$ bit memori. Bandingkan dengan algoritma Sieve of Eratosthenes yang menggunakan $\Theta(N)$ operasi dan $\Theta(N^{1/2}(\log \log N)/\log N)$ bit memori.



Ganbar 3. Perbandingan kecepatan algoritma *Sieve of Eratosthenes* dengan algoritma *Sieve of Atkin*

Pada gambar di atas garis merah mewakili metode *Sieve of Eratosthenes* dan garis biru mewakili metode *Sieve of Atkin*. Perbedaannya terlihat kecil, tetapi menjadi secara signifikan lebih penting jika N bertambah besar.

4. KESIMPULAN

Dari penjelasan di atas maka dapat ditarik beberapa kesimpulan, di antaranya:

1. Bilangan prima adalah permasalahan yang serius didalam ilmu komputer dan teori bilangan. Hal tersebut sangat dibenarkan dalam bidang kriptografi, saat protokol-protokol enkripsi kunci publik didasarkan pada penggunaan dari bilangan-bilangan prima dengan ukuran besar, dan keamanannya didasarkan pada kesulitan untuk mendapatkan faktor-faktor prima dari suatu bilangan prima yang sangat besar.

2. Terdapat banyak cara untuk mendapatkan urutan bilangan prima, di antaranya dengan metode *Brute Force* (naif), *Sieve of Eratosthenes,* dan *Sieve of Atkin.*

3. Algoritma *Sieve of Eratosthenes* dapat mendapatkan urutan bilangan prima dengan cepat. Algoritma ini menggunakan $\Theta(N)$ operasi dan $\Theta(N^{1/2}(\log \log N)/\log N)$ bit memori.

4. Algoritma *Sieve of Atkin* merupakan algoritma tercepat dan termodern untuk pencarian bilangan prima sejauh ini. algoritma ini menggunakan $\Theta(N/\log \log N)$ operasi dengan hanya $N^{1/2+o(1)}$ bit memori.

5. DAFTAR PUSTAKA

[1] Krenzel.info >> Blog Archieve >> P is for Prime. (2007). *http://krenzel.info/?p=83*. Tanggal akses 6 Mei 2007 pukul 15.00.

[2] Sieve of Atkin - Wikipedia, the free encyclopedia (2007). http://*en.wikipedia.org/sieve_of_atkin*. Tanggal akses 6 Mei 2007 pukul 15.00.

[3] Sieve of Eratosthenes – Wikipedia, the free encyclopedia (2007). *http://en.wikipedia.org/sieve_of_eratosthenes*. Tanggal akses 6 Mei 2007 pukul 15.00