Analisis Perbandingan Kompleksitas Waktu Algoritma *Sieve of Sundaram* dengan Algoritma Pencarian Bilangan Prima Lainnya

William Fu 13518055
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13518055@std.stei.itb.ac.id

Abstrak—**Algoritma merupakan aspek yang sudah tidak dapat terpisahkan lagi dari bidang komputasi. Akan tetapi, algoritma tidak terbatas pemanfaatannya dalam bidang pemrograman atau komputasi saja. Tanpa disadari, manusia dalam berbagai bidang keilmuan lainnya pula memanfaatkan algoritma dalam penyelesaian suatu masalah. Dalam beberapa kasus, ada lebih dari satu algoritma yang dapat dimanfaatkan untuk menyelesaikan maslaah yang sama. Akan tetapi, belum tentu semua algoritma yang ada merupakan algoritma yang bagus untuk diimplementasikan pada agen pemroses. Dalam hal ini, diperlukan analisis kompleksitas algoritma untuk menentukan algoritma yang lebih mangkus dan sangkil.**

Kata kunci—**Komplesitas waktu, Algoritma Eratosthenes, Algoritma Atkin, Efektif dan efisien**

I. PENDAHULUAN

Istilah berpikir komputasional (*computational thinking*), yang dulunya hanya diterapkan dalam ilmu komputer dewasa ini mulai dilihat relevansinya dengan berbagai bidang dalam kehidupan. Salah satu aspek yang ada dalam *computational thinking* adalah merancang suatu algoritma yang nantinya akan diimplementasikan untuk penyelesaian masalah. Algoritma yang tersebut tentunya harus dirancang sebaik mungkin sehingga memakan waktu dan ruang memori yang seminimal mungkin. Dengan demikian, algoritma tersebut pula dapat digunakan untuk permasalahan yang lain dengan masukan (*input*) yang ukurannya lebih besar.

Salah satu permasalahan umum yang memiliki banyak algoritma solusi penyelesaiannya adalah mencari daftar bilangan prima. Permasalahan ini dianggap menarik akibat sifat bilangan prima yang unik. Dalam bidang teknologi informasi, bilangan prima bermanfaat dalam algoritma kriptografi untuk menjaga keamanan suatu informasi.

Algoritma yang digunakan untuk pencarian bilangan prima umumnya bekerja dengan "menyaring" bilangan komposit pada rentang terentu hingga tersisa bilangan prima saja. Dalam tulis ini, penulis akan lebih focus pada analisa kompleksitas waktu yang dimiliki algoritma-algoritma ini. Dalam tulisan ini pula, akan dilakukan perbandingan waktu algoritma yang telah diimplementasikan dalam bentuk program dalam Bahasa C.

II. KOMPLEKSITAS ALGORITMA

A. *Kompleksitas Algoritma*

Kompleksitas algoritma meneliti seberapa besar waktu dan ruang (memori) yang dibutuhkan untuk menjalankan algoritma tersebut. Namun, dalam kenyataannya, kedua aspek ini pula bergantung pada arsitektur komputer serta jenis *compiler* yang digunakan dalam mengeksekusi algoritma tersebut, sehingga waktu dan ruang yang digunakan menjadi parameter yang bersifat relatif. Oleh karena itu, pengukuran kompleksitas algoritma tidak memandang aspek-aspek tersebut dan menggunakan model abstrak yang bersifat independen terhadap aspek luar lainnya.

Kompleksitas algoritma umumnya dinotasikan dalam bentuk fungsi yang menunjukkan relasi antara ukuran masukan (*input*) dengan banyaknya operasi fundamental yang dilakukan. Kompleksitas demikian disebut dengan *time complexity* atau kompleksitas waktu. Selain itu, kompleksitas juga dapat diteliti dari banyaknya *storage/*ruang yang dibutuhkan dalam menjalankan algoritma tersebut; kompleksitas ini disebut *space complexity.*

Kompleksitas waktu biasanya dinotasikan dengan $\mathbf{T}(\boldsymbol{n})$ yang menunjukkan berapa waktu yang dibutuhkan untuk memroses *input* dengan ukuran *n.* Kompeksitas waktu terbagi menjadi tiga buah jenis yaitu:

1. $\mathbf{T_{max}(n)}$, yaitu kompleksitas waktu untuk kasus terburuk (*worst case*)
2. $\mathbf{T_{min}(n)}$, yaitu kompleksitas waktu untuk kasus terbaik (*best case*)
3. $\mathbf{T_{avg}(n)}$, yaitu kompleksitas waktu untuk kasus secara rata-rata. (*average case*)

Dalam menyatakan fungsi kompleksitas waktu $\mathbf{T}(\boldsymbol{n})$, umumnya digunakan notasi asimptotik (*Asymptotic Notations*). Sebagai ilustrasi, asumsikan telah diketahui kompleksitas $\mathbf{T}(\boldsymbol{n}) = n^3 + 3n + 3$. Apabila dibandingkan dengan sebuah fungsi $\boldsymbol{f(n)} = n^3$, dapat dikatakan bahwa $\boldsymbol{T(n)}$ dan $\boldsymbol{f(n)}$ memiliki laju pertumbuhan yang sama. Notasi asimptotik juga biasa dikenal dengan istilah *Big-O Notation* yang dapat didefinisikan sebagai $\mathbf{T(n)} = \mathbf{O}(f(n))$. Fungsi *f(n)* dikatakan sebagai *upper bound* dari fungsi $\mathbf{T(n)}$ yang memenuhi persamaan

$$\mathbf{T(n)} \le \mathbf{C} \cdot f(n)$$

untuk setiap $n \geq n_0$, dengan $n_0 \geq 0$. **C** adalah konstanta positif tak nol (C > 0). Dalam menganalisis suatu algoritma, notasi *Big-O* biasa digunakan untuk mendeskripsikan kompleksitas waktu *worst-case behavior.* Tabel 1 di bawah ini menampilkan sejumlah notasi *Big O* yang umum dijumpai mulai dengan urutan waktu yang membesar.

Kelompok Algoritma	*Nama*
O(1)	Konstan
O(*log* n)	Logaritmik
O(n)	Linier
O(n *log* n)	*n log n*
$O(n^2)$	Kuadratik
$O(n^3)$	Kubik
$O(n^k)$	Polinomial
$O(2^n)$ (atau $O(k^n)$)	Eksponensial
O(n!)	Faktorial

Tabel 1. Notasi *Big-O* yang Umum Dijumpai *(Sumber : https://courses.cs.washington.edu/courses/cse373/14wi/lecture4.pdf dengan terjemahan)*

B. Sieve of Sundaram

Saringan Sundaram (*Sieve of Sundaram*) merupakan algoritma pencarian bilangan prima dari rentang bilangan tertentu yang ditemukan oleh seorang matematikawan India bernama S.P. Sundaram pada tahun 1930. Meskipun tidak sepopuler kedua algoritma yang telah dijelaskan sebelumnya, algoritma Sundaram memiliki cara yang cerdik dalam menyaring bilangan-bilangan komposit pada suatu larik.

C. Sieve of Eratosthenes

Saringan Eratosthenes (*Sieve of Eratosthenes*) merupakan salah satu algoritma pencarian bilangan prima tertua. Algoritma ini ditemukan oleh Eratosthenes, seorang matematikawan dan astronom Yunani Kuno. Algoritma ini bekerja dengan 'menyaring' bilangan komposit dari kumpulan bilangan dengan rentang tertentu. Algoritma ini relatif lebih mudah untuk dipahami sehingga lebih mudah dalam implementasinya dalam bentuk program.

D. Sieve of Atkin

Saringan Atkin *(Sieve of Atkin)* merupakan algoritma modern pencarian bilangan prima yang ditemukan oleh A.O.L Atkin dan Daniel J. Bernstein pada tahun 2003. Algoritma ini merupakan versi *optimized* dari algoritma Saringan Eratosthenes yang dianggap masih memiliki beberapa kelemahan.

E. Bilangan Prima

Bilangan prima merupakan bilangan positif yang lebih besar dari satu ($p > 1$) dan hanya memiliki dua faktor saja, yaitu bilangan itu sendiri dan satu (1). Selain dua (2), bilangan prima yang ada merupakan bilangan ganjil. Bilangan yang bukan prima biasa dikenal dengan istilah bilangan komposit. Menurut teorema fundamental aritmetika, bilangan komposit dapat dituliskan sebagai perkalian bilangan-bilangan prima. Bilangan prima memiliki sifat 'misterius' sebab ilmuwan masih belum berhasil untuk menemukan pola eksak dari bilangan ini.

Dalam bidang teknologi informasi, bilangan prima aplikasi yang sangat signifikan dalam menjaga kerahasiaan informasi. Bilangan prima berperan di proses kriptografi dalam menghasilkan kunci yang diperlukan dalam proses enkripsi dan dekripsi. Karena sifatnya yang hanya memiliki dua faktor saja, kunci yang dihasilkan biasa memiliki kekuatan yang lebih baik dan lebih sulit untuk diretas dibandingkan dengan bilangan lainnya. Umumnya, bilangan prima yang dimanfaatkan dalam proses kriptografi merupakan bilangan prima dengan berdigit banyak (ratusan digit). Oleh karena itu, para ilmuwan dalam bidang komputasi senantiasa mencari bilangan-bilangan prima yang baru. Dalam dunia maya, terdapat sebuah proyek betakjub *The Great Mersenne Prime Search,* yang bertujuan untuk menemukan bilangan prima terbaru. Per Desember 2018, bilangan prima Mersenne yang ditemukan adalah $2^{82,589,933}-1$.

III. PEMBAHASAN

A. Kompleksitas Algoritma Saringan Sundaram

Algoritma Saringan Sundaram bekerja dengan mengeliminasi bilangan komposit pada suatu larik yang berisi bilangan dengan rentang tertentu. Algoritma ini bermanfaat untuk mencari bilangan ganjil prima yang ada pada rentang yang diinginkan. Sehingga, perlu diingat untuk menampilkan bilangan 2 sebagai satu-satunya bilangan prima yang genap. Algoritma ini bekerja dengan mengeliminasi bilangan yang memenuhi ketidaksamaan berikut.

$$\boldsymbol{i + j + 2ij \leq n}$$

dengan *i* dan *j* merupakan bilangan *integer* yang memenuhi syarat $1 \leq \boldsymbol{i} \leq \boldsymbol{j}$. Langkah-langkah dalam algoritma *Sieve of Sundaram* adalah sebagai berikut.

1. Membuat sebuah larik dengan sebesar setengah dari masukan *n (*sebut saja *size).* Larik ini memiliki elemen *boolean* yang diinisilisasi dengan nilai TRUE.
2. Kemudian, lakukan iterasi dengan variable *i* mulai dari indeks pertama hingga indeks ke-*size.*
3. Di dalam skema *looping* tersebut, lakukan sebuah *inner -loop* dengan variabel *j* dari *i* sampai *(size-i)/(2*i-1).*
4. Larik dengan indeks *i + j + 2ij* di-*assign* dengan nilai FALSE.
5. Saat *looping* berakhir, daftar bilangan prima sebelum *n* merupakan seluruh indeks larik dengan nilai TRUE yang dikali dua dan ditambah satu.

Sebagai bantuan untuk visualisasi dari langkah-langkah tersebut, Gambar 1 merupakan kode semu (*pseudocode*) dari Algoritma Saringan Sundaram dengan sedikit optimasi dalam *inner-loop* dengan variabel *j.*

```
/*Sieve of Sundaram*/

a = array of boolean[0..MaxEl]
input(n)

size = n div 2

/*Mengisi larik dengan FALSE*/
for i=0 to size+1 do
  a[i] ← false

a[0] ← a[1] ← false

/*Menyaring bilangan non prima*/
for i=0 to size+1 do
```

```
  for j=i to (size-i)/(2*i-1)
     a[i+j+2*i*j] ← false

/*Sieve selesai*/
for i=0 to size+1 do
  if a[i] then
    print(2*i+1)
```

Gambar 1. *Pseudocode* Algoritma Saringan Sundaram *(Sumber https://www.geeksforgeeks.org/sieve-sundaram-print-primes-smaller-n/:)*

Pada bagian *looping* pertama (variable *i*), kompleksitas waktu yang dibutuhkan adalah sebanyak *O(n)* dengan *n* dalam hal ini merujuk pada ukuran dari larik yang digunakan. Sedangkan, pada bagian *inner-loop* (variabel *j*), kompleksitas waktu yang dimakan adalah sebesar O(*log n*), dengan *n* dalam hal ini pula merujuk pada ukuran dari larik yang digunakan. Hal ini dikarenakan variabel iterasi *j* di-*assign* dengan nilai *i* pada setiap *loop*. Aksi ini mengurangi jumlah operasi *assignment* yang perlu dilakukan secara cukup signifikan. Dengan demikian, algoritma ini memiliki kompleksitas waktu

O(*n log n*).

B. Komplekstias Algoritma Saringan Eratosthenes

Seperti yang telah dijabarkan pada bagian II, algoritma *Sieve of Eratosthenes* merupakan salah satu metode yang banyak digunakan untuk mencari daftar bilangan prima yang ada sebelum bilangan *n* tertentu masukan pengguna. Langkah-langkah saringan Eratosthenes adalah sebagai berikut.

1. Membuat sebuah larik dengan elemen *boolean* TRUE sepanjang bilangan *n* masukan pengguna.
2. Membuat suatu variable *p* untuk melakukan iterasi dimulai saat *p* sama dengan 2 sampai dengan akar kuadrat dari *n*. Larik dengan indeks yang merupakan kelipatan *p* akan diubah nilainya menjadi FALSE.
3. Bilangan *p* akan di-*increment,* bila larik dengn indeks ke-*p* sudah bernilai FALSE, maka akan dicari elemen larik dengan nilai TRUE untuk dilakukan operasi sesuai deskripsi pada langkah sebelumnya.
4. Bila skema iterasi telah selesai, daftar bilangan prima merupakan indeks larik yang bernilai TRUE.

Sebagai bantuan untuk visualisasi dari langkah-langkah tersebut, Gambar 2 merupakan kode semu (*pseudocode*) dari Algoritma Eratosthenes.

```
/*Sieve of Eratosthenes*/
a = array of boolean [0..MaxEl]

input(n)         /*2<=n<=MaxEl*/

/*Assignment larik*/
for i=0 to n do
  a[i] ← true

a[0] ← a[1] ← false

/*Saringan Eratosthenes*/
for i=2 to n do

   if a[i] then
      for j = i to (n div i) do
         a[j*i] ← false

/*Saringan selesai*/
for i=0 to n+1 do
  if a[i] then
    print(i)
```

Gambar 2. *Pseudocode* Algoritma Saringan Eratosthenes *(Sumber: https://www.geeksforgeeks.org/sieve-of-eratosthenes/)*

Algoritma Saringan Eratosthenes memiliki kompleksitas waktu

O(n log(log n)).

Hal ini dapat dilihat pada skema looping yang terjadi saat terjadi penyaringan bilangan komposit. Pada *for-looping* pertama (variabel *i*), dilakukan iterasi sebanyak *n* kali sehingga kompleksitas waktu yang dimilikinya adalah O(*n*). Sementara, untuk *for-looping* dengan variabel *j* (*inner-loop* skema iterasi yang pertama), jumlah operasi yang perlu dilakukan menurun secara logaritmik, sebab setiap proses *assignment* yang terjadi sudah mengurangi banyaknya proses *assignment* yang seharusnya dilakukan untuk langkah selanjutnya. Dengan demikian *inner-loop* tersebut memiliki kompleksitas waktu O(*log (log n)*).

C. Komplekstias Algoritma Saringan Atkin

Algoritma Atkin merupakan algoritma modern hasil optimasi dari algoritma *Sieve of Eratosthenes.* Dalam kasus algoritma ini, sudah dilakukan beberapa penanganan untuk kasus tertentu sehingga beberapa pengecekan yang berulang tidak perlu dilakukan. Dalam saringan Atkin, dua, tiga, dan lima sudah dianggap sebagai bilangan prima, sehingga pemeriksaan dilakukan mulai dari 5 sampai masukan yang diinginkan. Langkah-langkah algoritma pada Saringan Atkin adalah sebagai berikut.

1. Membuat sebuah larik bilangan dari indeks 5 sampai masukan *n*.
2. Membuat skema iterasi dengan variable *x* mulai dari indeks 1 sampai $\sqrt{n}$ serta *inner-loop* dengan variable *y* mulai dari indeks 1 sampai $\sqrt{n}$ pula.
3. Menginisialisasi sebuah variabel *cek* dengan syarat berikut
   a. Apabila sisa pembagian *cek* dengan 60 adalah 1,13,17,29,37,41,49,atau 53, maka *cek* adalah $4x^2+y^2$
   b. Apabila sisa pembagian *cek* dengan 60 adalah 7,19,31,43 maka *cek* adalah $3x^2+y^2$
   c. Apabila sisa pembagian *cek* dengan 60 adalah 11,23,47,59, maka *cek* adalah $3x^2-y^2$ saat $x>y$
4. Memulai dari indeks terkecil pada larik, dan melakukan skema *looping* seperti langkah 3 dan dilanjutkan dengan indeks berikutnya yang belum ditandai.
5. Untuk setiap indeks yang merupakan bilangan kuadrat sempurna, ditandai sebagai bilangan prima (elemen larik diberi nilai FALSE)

Gambar 3 berikut ini menunjukkan kode semu (*pseduocode*) dari implementasi Saringan Atkin dalam model program.

```
/*Sieve of Atkin*/

    for i=0 to n do
        sieve[i] ← False
```

```
    /*Penyaringan pertama*/
    x ← 1
    while(x * x < n) do
        y ← 1
        while(y * y < n) do

            cek ← (4 * x * x) + (y * y)
          if (cek <=n and (cek mod 12 = 1
or cek mod 12 = 5))
                sieve[n] ← True

            cek ← (3 * x * x) + (y * y)
            if (cek <= n and cek mod 12 = 7)
                sieve[n] ← True

            cek ← (3 * x * x) - (y * y)
        if (x > y and cek <= n and cek mod 12
= 11)
                sieve[n] ← True
            y ++
        x ++

    /*Penyaringan kedua*/
    r ← 5
    while(r * r < n) do
        if (sieve[r]) :
           for i in range(r * r, n, r * r) do
                sieve[i] ← False
```

Gambar 3. *Pseudocode* Algoritma Saringan Atkin *(Sumber* [*https://www.geeksforgeeks.org/sieve-sundaram-print-primes-smaller-n/*](https://www.geeksforgeeks.org/sieve-sundaram-print-primes-smaller-n/)*:)*

Pada skema iterasi untuk penyaringan pertama, dapat dilihat bahwa skem tersebut memiliki kompleksitas algoritma $O(\sqrt{n} \cdot \sqrt{n})$ atau sama dengan O(n). Sedangkan, skema penyaringan yang kedua akan memilki kompleksitas $O(\sqrt{n} \cdot \log(\log n))$. Hal ini dikarenakan pada loop pertama memiliki kompleksitas $O(\sqrt{n})$ dikarenakan operasi dilakukan sebanyak $\sqrt{n}$ kali, dengan *n* merupakan ukuran larik yang digunakan. Lalu, untuk skema looping yang ada di dalamnya akan memiliki kompleksitas sebesar $O(\log(\log n))$ sebab setiap operasi *assignment* yang terjadi akan mengurangi kombinasi operasi yang dilakukan pada langkah berikutnya secara logaritmik. Dengan demikian, dapat dikatakan bahwa algoritma *Sieve of Atkin* memiliki kompleksitas waktu

$$O(\sqrt{n} \cdot \log(\log n))$$

IV. Implementasi dan Analisis

Untuk membandingkan waktu algoritma secara riil, penulis mengimplementasikan ketiga algoritma yang ada dalam bahasa pemrograman C berdasarkan referensi yang didapatkan. Algoritma tersebut diimplementasikan dalam bentuk prosedur yang dapat dipanggil saat program utama dijalankan. Implementasi ketiga algoritma ini dapat dilihat pada Gambar 4,5,dan 6.

```c
void SieveOfSundaram(int N){

    int m = N/2;
    int i,j;
    boolean prime[m+1];

    for(i=0; i<=m; i++){
        prime[i] = true;
    }
    prime[0] = false; prime[1] = false;

    for(i=1; i<m; i++){
        for(j=i; j<=(m-i)/(2*i+1); j++){
            prime[i+j+2*i*j] = 0;
        }
    }

    printf("2 ");
    for(i=0; i<m; i++){
        if(prime[i]){
            printf("%d ", 2*i+1);
```

Gambar 4. Implementasi *Sieve Of Sundaram dalam Bahasa C* (*Referensi*: https://www.geeksforgeeks.org/sieve-of-sundaram/)

```c
void SieveOfEratosthenes (int N){

    boolean prime[N+1];
    int i,j;

    for(i=0; i<=N; i++){
        prime[i] = true;
    }
    prime[0] = false; prime[1] = false;

    for(i=2; i*i<=N; i++){
        if(prime[i]){
            for(j=i; j<=N/i; j++){
                prime[j*i]= false;
            }
        }
    }

    int ctr = 0;
    for(i=0; i<=N; i++){
        if(prime[i]){
            ctr++;
        }
    }
    printf("%d\n",ctr);
```

Gambar 5. Implementasi *Sieve Of Eratosthenes dalam Bahasa C* (*Referensi*: https://www.geeksforgeeks.org/sieve-of-eratosthenes/)

```
void SieveOfAtkin(int N){

    int i,x,y,n,j;
    boolean prime[N];

    for(i=0; i<N; i++){
        prime[i] = false;
    }
    // prime[0] = false; prime[1] = false;

    int limit = sqrt(N);
    for (x = 1; x <= limit ; x++) {
        for (y = 1; y <= limit; y++) {

            // Main part of Sieve of Atkin
            n = (4 * x * x) + (y * y);
            if (n <= N && (n % 12 == 1 || n % 12 == 5))
                prime[n] = !prime[n];

            n = (3 * x * x) + (y * y);
            if (n <= N && n % 12 == 7)
                prime[n] = !prime[n];

            n = (3 * x * x) - (y * y);
            if (x > y && n <= N && n % 12 == 11)
                prime[n] = !prime[n];
```

```
    for (int r = 5; r <= N; r++) {
        if (prime[r]) {
            for (j=1; j*r*r <= N; j++){
                prime[j*r*r] = false;
            }
        }
    }

    printf("2 3 ");
    for(i=0; i<N; i++){
        if(prime[i]){
            printf("%d ", i);
```

Gambar 6. Implementasi *Sieve Of Atkin dalam Bahasa C* (*Referensi*: https://www.geeksforgeeks.org/sieve-of-atkin/)

Untuk memeriksa kebenaran ketiga algoritma tersebut, dilakukan percobaan kasus uji untuk *input* 100. Sehingga, program akan menampilkan bilangan prima sebelum seratus.

```
C:\Users\ASUS\Desktop>gcc -o sunda
ram sundaram.c

C:\Users\ASUS\Desktop>sundaram
100
2 5 7 11 13 17 19 23 29 31 37 41 4
3 47 53 59 61 67 71 73 79 83 89 97
 runtime : 0.0000000 seconds
```

Gambar 7. Kasus Uji Program Saringan Sundaram untuk input 100 (*Sumber : milik pribadi*)

```
C:\Users\ASUS\Desktop>gcc -o eratosthe
nes eratos.c

C:\Users\ASUS\Desktop>eratosthenes
100
2 3 5 7 11 13 17 19 23 29 31 37 41 43
47 53 59 61 67 71 73 79 83 89 97 runti
me : 0.0050000 seconds
```

Gambar 8. Kasus Uji Program Saringan Eratothenes untuk input 100 (*Sumber : milik pribadi*)

```
C:\Users\ASUS\Desktop>gcc -o atkin atkin.c

C:\Users\ASUS\Desktop>atkin
100
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67
71 73 79 83 89 97 runtime : 0.0050000 seconds
```

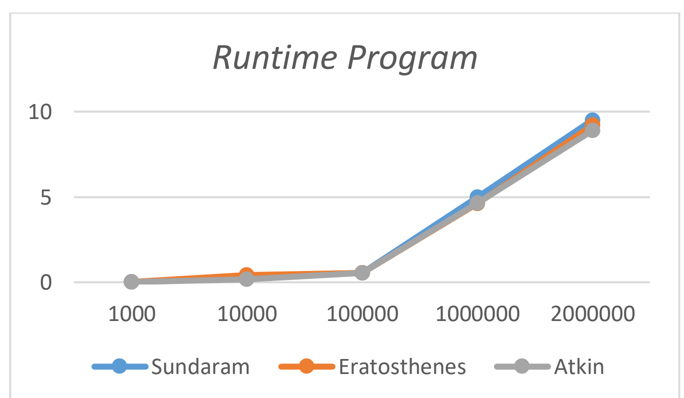
Gambar 9. Kasus Uji Program Saringan Atkin untuk input 100 (*Sumber : milik pribadi*)

Gambar 7, 8, dan 9 diatas merupakan pembuktian bahwa program sudah dapat berjalan sebagaimana mestinya tanpa melakukan kesalahan. Dengan demikian, dapat dilakukan uji coba untuk mengukur waktu berjalannya program dengan masukan yang berbeda-beda. Hasil uji coba yang telah dilakukan direkam pada Tabel 2 berikut ini.

Input (*n*)	Waktu (s)		
	Sundaram	Eratosthenes	Atkin
1,000	0.015	0.018	0.014
10,000	0.254	0.413	0.163
100,000	0.554	0.538	0.560
1,000,000	4.628	4.599	4.628
2,000,000	8.838	8.783	8.888

Tabel 2. Tabel Perbandingan Waktu Berjalan Pencarian Bilangan Prima dengan Algoritma Sundaram, Eratosthenes, dan Atkin (*Sumber : milik pribadi*)

Melalui tabel ini, tidak ada perbedaan jumlah waktu yang signifikan antar ketiga algoritma. Hal ini mungkin disebabkan oleh input realtif masih kurang besar untuk melihat sebuah perbedaan yang signifikan. Selain itu, program juga mengalami kegagalan apabila masukan sudah mencapai nilai lebih dari 2 juta. Penulis memprediksi permasalahan ini terjadi akibat keterbatasan larik yang hanya memiliki ukuran tertentu saja. Permasalahan ini mungkin dapat diakali dengan mengubah tipe data masukan serta indeks larik dengan tipe data dengan ukuran bit yang lebih besar. Untuk melihat perbandingan waktu ketiga algoritma ini, Grafik 1 menampilkan hasil dengan lebih jelas.



Grafik 1. Perbandingan Waktu Berjalan (*runtime*) Ketiga Program Pencarian bilangan prima (*Sumber : milik penulis*)

V. KESIMPULAN

Melalui analisis serta implementasi yang ada dalam tulisan ini, penulis dapat menarik kesimpulan bahwa setiap algoritma untuk penyelesaian permasalahan yang sama dapat memiliki kompleksitas yang berbeda-beda. Dalam pencarian bilangan prima, algoritma Sundaram, Atkin, dan Eratosthenes masing-masing memiliki kompleksitas yang ternyata berbeda-berbeda. Algoritma Saringan Sundaram, Eratosthenes dan Atkin berturut-turut memiliki $\mathbf{O(n \log n)}$, $\boldsymbol{O}(\boldsymbol{n} \cdot \mathbf{log}\,(\mathbf{log}\,\boldsymbol{n}))$, dan $\boldsymbol{O}(\sqrt{\boldsymbol{n}} \cdot \mathbf{log}\,(\mathbf{log}\,\boldsymbol{n}))$ Secara teoritis, hal ini berarti algoritma saringan Atkin memiliki waktu berjalan yang paling cepat, disusul dengan algoritma saringan Eratosthenes lalu algoritma Sundaram.

Dalam implementasinya dalam bentuk program dengan bahasa C, ketiga algoritma belum terlihat perbedaan waktu berjalan yang signifikan. Hal ini mungkin disebabkan keterbatasan program yang telah dirancang serta kesalahan dalam mengambil tipe data untuk masukan dan larik. Hal ini dapat diperbaiki dengan mengoptimasi algoritma yang ada serta menggunakan tipe data yang memungkinkan untuk komputasi nilai-nilai yang besar.

VII. UCAPAN TERIMA KASIH

Pertama-tama, penulis ingin menghaturkan puji dan syukur kepada Tuhan Yang Maha Esa sebab atas rahmat dan berkat-Nya, tulisan ini dapat diselesaikan secara lengkap. Penulis pula ingin mengucapkan terima kasih kepada Bapak Dr. Ir. Rinaldi, MT selaku dosen mata kuliah Matematika Diskrit atas bimbingannya selama satu semester perkuliahan ini. Tak lupa juga penulis ingin mengucapkan terima kasih pula kepada seluruh pihak yang terlibat dalam penyelesaian makalah ini.

REFERENSI

[1] Kenneth H. Rosen, *Discrete Mathematics and Its Applications.* New York, NY : McGraw-Hill, 2019.
[2] https://www.cs.cmu.edu/~adamchik/15-122/lectures/complexity/complexity.pdf (diakses pada 5 Desember 2019 pukul 16:21 WIB)
[3] http://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2015-2016/Kompleksitas%20Algoritma%20(2015).pdf (diakses pada 5 Desember 2019 pukul 08:41 WIB)
[4] https://theconversation.com/why-do-we-need-to-know-about-prime-numbers-with-millions-of-digits-89878 (diakses pada 5 Desember 2019 pada pukul 09:48 WIB)
[5] https://plus.maths.org/issue50/features/havil/2pdf/index.html/op.pdf (diakses pada 5 Desember 2019 pukul 17:37 WIB).
[6] https://luckytoilet.wordpress.com/2010/04/18/the-sieve-of-sundaram/ (diakses pada 6 Desember 2019 pukul 00:50 WIB).
[7] http://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2015-2016/Teori%20Bilangan%20(2015).pdf (diakses pada 6 Desember 2019 pukul 06:00 WIB)
[8] https://www.mersenne.org/ (diakses pada 6 Desember 2019 pukul 06:15 WIB)
[9] https://www.geeksforgeeks.org/sieve-of-sundaram/ (diakses pada 5 Desember 2019 pukul 22:03 WIB)
[10] https://www.geeksforgeeks.org/sieve-of-eratosthenes/ (diakses pada 5 Desember 2019 pukul 22:10 WIB)
[11] https://www.geeksforgeeks.org/sieve-of-atkin/ (diakses pada 5 Desember 2019 pukul 23:45 WIB)

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 6 Desember 2019

William Fu
13518055