

# Bilangan Acak pada Komputer dan Pembangkit Bilangan Acak Semu

Mohammad Dwinta Harits C./13519041  
Program Studi Teknik Informatika  
Sekolah Teknik Elektro dan Informatika  
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia  
13519041@std.stei.itb.ac.id

**Abstract**—Bilangan acak adalah objek yang sangat diperlukan dalam jalannya program. Sebut saja dalam game, gerakan karakter musuh akan monoton jika tidak ada keacakan padanya. Pada makalah ini akan diberi penjelasan singkat mengenai bilangan acak dan sebuah algoritma sederhana yang memiliki keluaran berupa bilangan acak

**Keywords**—Algoritma, Bilangan Acak, Bilangan Acak Semu, PBAS

## I. PENDAHULUAN

Keseharian kita penuh dengan bilangan-bilangan yang kemunculannya acak. Akan tetapi apakah sebenarnya “keacakan” itu? Bagaimana dapat sebuah komputer mengimplementasi “keacakan”?

## II. LANDASAN TEORI

### 2.1 Bilangan Acak

Bilangan acak adalah bilangan yang dihasilkan secara acak. Suatu Bilangan dinyatakan sebagai bilangan acak apabila kemunculannya hanya dapat ditebak dengan kemungkinan. Pada pemilihan bilangan acak dari suatu himpunan, seluruh elemen memiliki kemungkinan yang sama untuk terpilih. Contoh pengambilan bilangan acak adalah dengan menggunakan dadu.

Komputer sebagai mesin *deterministic*, yakni dapat ditebak seluruh keluaran berdasarkan masukan yang diberikan, tidak dapat menghasilkan bilangan acak sejati. Hal ini disebabkan seluruh masukan yang sama akan menghasilkan keluaran yang sama pula.

### 2.2 Bilangan Acak Semu

Bilangan acak semu adalah bilangan acak yang tidak memenuhi seluruh kriteria bilangan acak asli. Bilangan acak semu umumnya dihitung dengan komputer. Perhitungan bilangan acak semu secara berkelanjutan akan membuat pengulangan terjadi.

Bilangan acak semu biasanya cukup untuk menggantikan bilangan acak sejati pada penggunaan dalam program-program. Syarat dapat digunakannya bilangan acak semu pada aplikasi ialah himpunan bilangan yang dihasilkan cukup luas dan prediksi kemunculan bilangan selanjutnya merupakan suatu yang kompleks.

### 2.3 Pembagian pada Bilangan Bulat

Menurut Teorema Euclidean, jika ada dua bilangan bulat  $m$  dan  $n$ , dengan  $n$  lebih besar dari nol, apabila  $m$  dibagi  $n$  maka hasil pembagiannya adalah  $q$  dan sisanya  $r$  sedemikian sehingga

$$m = nq + r$$

dengan  $0 \leq r < n$ .

#### a. Pembagi Bersama Terbesar (PBB)

Pembagi Bersama Terbesar antara dua bilangan  $a$  dan  $b$  adalah  $c$  sehingga  $c$  adalah bilangan terbesar yang habis membagi  $a$  dan habis membagi  $b$ .

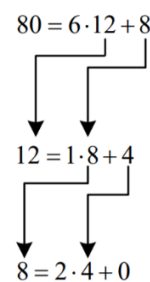
Pada persamaan 2.1 berlaku hubungan

$$PBB(m,n) = PBB(n,r)$$

Dengan fakta di atas dapat dirancang suatu metode untuk mengetahui nilai PBB dari dua buah bilangan, metode tersebut dikenal dengan Algoritma Euclidean dan ditemukan 300 SM.

### Algoritma Euclidean

1. Jika  $n = 0$  maka  $m$  adalah  $PBB(m, n)$ ; stop. tetapi jika  $n \neq 0$ , lanjutkan ke langkah 2.
2. Bagilah  $m$  dengan  $n$  dan misalkan  $r$  adalah sisanya.
3. Ganti nilai  $m$  dengan nilai  $n$  dan nilai  $n$  dengan nilai  $r$ , lalu ulang kembali ke langkah 1.



Gambar 2.1 Algoritma Euclidean

(<http://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Teori-Bilangan-2020-Bagian1.pdf>)

Setiap  $PBB(a,b)$  dapat dinyatakan dalam penjumlahan  $a$  dan  $b$  dikali dengan koefisien-koefisiennya. Artinya, setiap  $PBB(a,b)$  merupakan kombinasi linear dari  $a$  dan  $b$ . Salah satu kegunaan Algoritma Euclidean adalah untuk mencari koefisien

a dan b.

b. Aritmetika Modulo

Misalkan a, m, dan r bilangan bulat, maka:

$$a \bmod m = r$$

sehingga  $a = mq + r$  dimana q adalah hasil bagi a dan m.

m disebut modulus atau modulo, dan r merupakan hasil aritmetika modulo yang terlelak dalam himpunan  $\{0,1,2,\dots, m-1\}$ .

- (i)  $23 \bmod 5 = 3$                      $(23 = 5 \cdot 4 + 3)$
- (ii)  $27 \bmod 3 = 0$                      $(27 = 3 \cdot 9 + 0)$
- (iii)  $6 \bmod 8 = 6$                      $(6 = 8 \cdot 0 + 6)$
- (iv)  $0 \bmod 12 = 0$                      $(0 = 12 \cdot 0 + 0)$

Gambar 2.2 contoh perhitungan modulo

(<http://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Teori-Bilangan-2020-Bagian1.pdf>)

c. Kekongruenan Bilangan

Jika terdapat bilangan bulat a dan b serta sebuah bilangan bulat  $m > 0$ , maka a dinyatakan kongruen dengan b dalam modulus m jika dan hanya jika m habis membagi a-b.

Notasi kekongruenan a dan b adalah sebagai berikut:

$$a \equiv b \pmod{m}$$

Bentuk lain dari pernyataan tersebut ialah:

$$a = b + km$$

dimana k adalah suatu bilangan bulat.

1) Jika  $a \equiv b \pmod{m}$  dan c adalah sembarang bilangan bulat maka

- (i)  $(a + c) \equiv (b + c) \pmod{m}$
- (ii)  $ac \equiv bc \pmod{m}$
- (iii)  $a^p \equiv b^p \pmod{m}$  , p bilangan bulat tak-negatif

2) Jika  $a \equiv b \pmod{m}$  dan  $c \equiv d \pmod{m}$ , maka

- (i)  $(a + c) \equiv (b + d) \pmod{m}$
- (ii)  $ac \equiv bd \pmod{m}$

Gambar 2.3 sifat-sifat aritmatika kekongruenan

(<http://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Teori-Bilangan-2020-Bagian1.pdf>)

2.4 Komplekstitas Algoritma

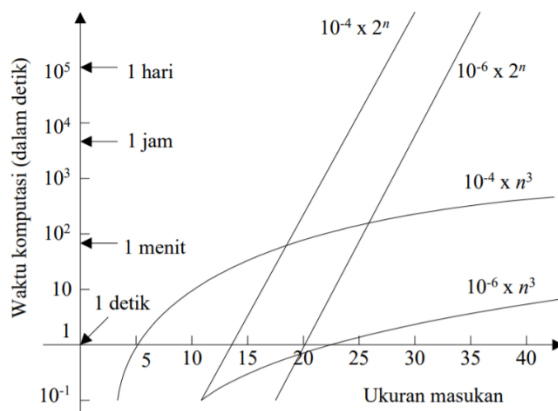
Pembuatan sebuah algoritma tidak hanya harus memenuhi spesifikasi(kebutuhan) akan tetapi juga harus memiliki efektivitas tinggi. Algoritma yang mangkus(efektif) berarti lebih serba guna pada berbagai mesin dan prosesor, sebuah sifat yang penting dalam dunia informatika.

Efektivitas algoritma dinilai dari memori dan waktu yang digunakan, semakin sedikit memori dan waktu terpakai maka algoritma dinilai lebih efektif.

Kebutuhan memori dan waktu hanya bergantung pada masukan algoritma, yang dikenal dengan n, semakin besar nilai masukan maka memori dan waktu yang dibutuhkan untuk menjalankan proses lebih besar pula. Efektivitas algoritma tidak bergantung pada prosesor maupun *compiler*.

Kompleksitas algoritma adalah besaran abstrak yang digunakan untuk mengukur waktu dan ruang terpakai dalam

sebuah program. Terdapat dua jenis kompleksitas algoritma, yakni kompleksitas ruang(memori) dan kompleksitas waktu. Dengan menggunakan besaran kompleksitas waktu dan ruang algoritma, dapat ditentukan laju peningkatan waktu yang diperlukan algoritma dengan meningkatnya ukuran masukan(n).



Gambar 2.3 ilustrasi kompleksitas algoritma (<http://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Kompleksitas-Algoritma-2020-Bagian1.pdf>)

a. Kompleksitas Ruang

Kompleksitas ruang, S(n), adalah besaran yang digunakan untuk mengukur jumlah memori yang dipakai algoritma. Hal tersebut berarti jumlah memori terbanyak yang dibutuhkan suatu algoritma pada tiap-tiap waktu algoritma dijalankan. Sebagaimana dengan kompleksitas waktu, perhatian utama kompleksitas ruang adalah bagaimana kebutuhan memori bertambah seiring bertambahnya input.

b. Kompleksitas Waktu

Kompleksitas waktu, T(n), berguna untuk mengukur waktu yang terpakai selama algoritma bekerja tanpa menghiraukan penggunaan memori. Besaran ini dihitung berdasarkan jumlah operasi yang dilakukan satu algoritma, operasi yang dimaksud ialah: baca, tulis, aritmetika, *assignment*, perbandingan, dan pengaksesan/pemanggilan.

Sama halnya dengan kompleksitas ruang, terdapat tiga jenis kompleksitas waktu, yaitu

- Tmax(n): kebutuhan waktu maksimum pada *worst-case scenario*. Contoh: tidak ada elemen yang dicari pada *sequential search*
- Tmin(n): waktu tercepat algoritma dapat terselesaikan. Contoh: elemen yang dicari berada pada awal array pada *sequential search*
- Tavg(n): waktu yang dibutuhkan pada umumnya untuk suatu algoritma menyelesaikan proses.

Misalkan terdapat algoritma *Bubble Sort* dalam *pseudocode* sebagai berikut:

```

procedure BubbleSort(input/output  $a_1, a_2, \dots, a_n$  : integer)
{ Mengurut larik A yang berisi n elemen integer sehingga terurut menaik }
Deklarasi
   $i, j, temp$  : integer
Algoritma
  for  $i \leftarrow n-1$  downto 1 do
    for  $j \leftarrow 1$  to  $i$  do
      if  $a_{j+1} < a_j$  then
        { pertukarkan  $a_j$  dengan  $a_{j+1}$  }
         $temp \leftarrow a_j$ 
         $a_j \leftarrow a_{j+1}$ 
         $a_{j+1} \leftarrow temp$ 
      endif
    endfor
  endfor

```

Gambar 2.4 Bubble Sort

(<http://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Kompleksitas-Algoritma-2020-Bagian1.pdf>)

Untuk mencari jumlah operasi yang dilakukan perhitungan jumlah perbandingan pada loop terdalam:

**for**  $j \leftarrow 1$  **to**  $i$  **do**

{ proses }

berarti dilakukan  $i$  kali proses di loop terdalam.

Sementara pada loop terluar:

**for**  $i \leftarrow (n-1)$  **downto** 1 **do**

{ pengulangan dalam }

Dengan secara keseluruhan jumlah proses yang dilakukan oleh algoritma Bubble Sort adalah:

$$(n-1)+(n-2)+(n-3)+\dots+2+1 = n(n-1)/2 \text{ (deret aritmatik)}$$

Jika diasumsikan proses pada pengulangan terdalam sebagai proses tunggal, maka kompleksitas waktu di atas adalah kasus terbaik, terburuk, dan rata-rata secara bersamaan. Hal ini disebabkan algoritma Bubble Sort tidak membedakan larik yang sudah terurut atau belum dalam menjalankan proses.

### c. Kompleksitas Waktu Asimptotik

Sering kali waktu presisi dari sebuah proses kompleks tidak dipedulikan. Pada  $n$  yang kecil, waktu yang terpakai oleh tiap-tiap algoritma cenderung sama karena. Hal yang lebih penting adalah bagaimana trend penggunaan waktu algoritma berdasarkan perubahan masukan.

Sebagai contoh, sebuah algoritma memiliki kompleksitas waktu rata-rata sebagai berikut:

$$T(n) = n^2 + 2n \log(n) + 3$$

Waktu yang diperlukan oleh algoritma tersebut untuk melakukan proses pada masukan tertentu tidak terlalu penting. Yang lebih penting ialah bagaimana waktu yang dibutuhkan algoritma tersebut berubah seiring masukan bertambah besar. Notasi waktu algoritma untuk jumlah  $n$  yang besar disebut kompleksitas waktu asimptotik.

Dalam kasus  $T(n)$  di atas, suku yang dominan terhadap perubahan  $n$  adalah  $n^2$  (pada  $n$  yang cukup besar  $n^2$  saja tidak terlalu berbeda dengan  $T(n)$ ). Fakta tersebut dinyatakan dalam notasi Big-O yakni:

$$T(n) = O(n^2)$$

### d. Notasi Big-O

Notasi Big-O adalah cara untuk mengonversi keseluruhan langkah-langkah suatu algoritma kedalam bentuk aljabar, yaitu dengan menghiraukan konstanta yang lebih kecil dan koefisien

yang tidak berdampak besar terhadap keseluruhan kompleksitas permasalahan yang diselesaikan oleh algoritma tersebut. Secara sederhana, Big-O adalah suku yang memiliki pengaruh terbesar pada lama jalannya algoritma seiring membesarnya input.

Secara matematis, definisi Big-O adalah sebagai berikut:

$$T(n) = O(f(n))$$

bila terdapat konstanta  $C$  dan  $n_0$  sedemikian sehingga

$$T(n) \leq C f(n)$$

$$\text{untuk } n \geq n_0$$

dapat diamati bahwa  $f(n)$  adalah batas lebih atas dari  $T(n)$ , dengan demikian pada pemilihan notasi Big-O diambil  $f(n)$  yang paling kecil dan mencukupi.

Selain notasi Big-O terdapat bentuk kompleksitas waktu asimptotik lain seperti Big-Theta ( $n = n_0$ ) dan Big-Omega ( $n \leq n_0$ ).

Algorithm	Best Time Complexity	Average Time Complexity	Worst Time Complexity	Worst Space Complexity
Linear Search	$O(1)$	$O(n)$	$O(n)$	$O(1)$
Binary Search	$O(1)$	$O(\log n)$	$O(\log n)$	$O(1)$
Bubble Sort	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$
Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$
Insertion Sort	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$
Merge Sort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	$O(n)$
Quick Sort	$O(n \log n)$	$O(n \log n)$	$O(n^2)$	$O(\log n)$
Heap Sort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	$O(n)$
Bucket Sort	$O(n+k)$	$O(n+k)$	$O(n^2)$	$O(n)$
Radix Sort	$O(nk)$	$O(nk)$	$O(nk)$	$O(n+k)$
Tim Sort	$O(n)$	$O(n \log n)$	$O(n \log n)$	$O(n)$
Shell Sort	$O(n)$	$O((\log(n))^2)$	$O((\log(n))^2)$	$O(1)$

Gambar 2.5 Tabel Big-O dari beberapa algoritma pengurutan (<https://www.hackerearth.com/practice/notes/sorting-and-searching-algorithms-time-complexities-cheat-sheet/>)

### 2.5 Pembangkit Bilangan Acak Semu

Pembangkit Bilangan Acak Semu (PBAS) adalah algoritma yang memberikan keluaran berupa bilangan yang memiliki sifat kemunculan mirip bilangan acak asli. Bilangan yang dihasilkan oleh PBAS tentu bukan merupakan bilangan acak asli, karena kemunculan keluaran bergantung pada bilangan pancingan awal (seed). Walaupun komputer mampu menghasilkan bilangan acak asli dengan bantuan perangkat keras (contoh: bangkitnya transistor dalam prosesor yang tidak dapat diduga), PBAS tetap umum digunakan dalam program komputer karena kecepatan proses dan nilai yang dihasilkan mumpuni.

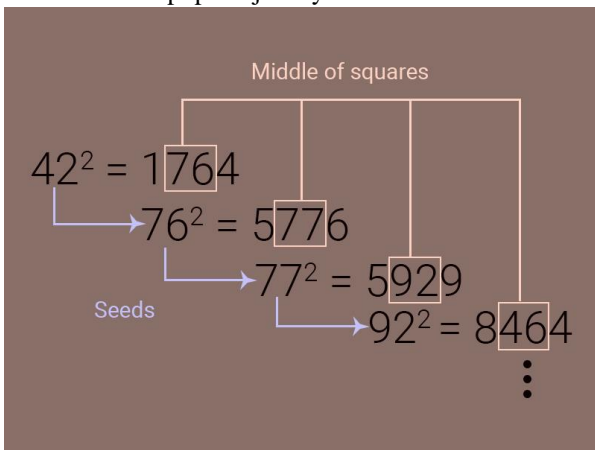
Badan Federal Keamanan Informasi Jerman memberikan 4 kriteria kualitas dari sebuah PBAS, antara lain:

1. Kemungkinan yang tinggi barisan bilangan keluaran PBAS berbeda tiap-tiap iterasi.
  2. Tidak dapat dibedakan dengan bilangan acak asli berdasarkan uji statistik tertentu.
  3. Tidak mungkin bagi manusia untuk menebak dengan pasti bilangan yang dihasilkan PBAS beberapa keluaran ke depan secara terurut.
  4. Tidak mungkin bagi manusia untuk menebak dengan pasti bilangan yang dihasilkan PBAS sebelumnya secara terurut.
- Standar yang memenuhi penggunaan program adalah standar 3 atau 4 saja.

PBAS awalnya diajukan oleh Matematikawan John von

Neumann pada tahun 1946. Algoritma yang dia ciptakan dikenal sebagai “middle-squared method” atau “metode tengah-kuadrat”. Prosesnya adalah sebagai berikut; ambil pancangan sebanyak  $x$  digit, lalu dikuadratkan, tulis hasil perhitungan sebagai  $2x$  digit dan ambil  $x$  digit yang paling tengah dari hasil, bilangan tengah tersebut adalah hasil dan digunakan sebagai masukan iterasi berikutnya. Algoritma ini tidak cukup dipakai pada masa ini karena bilangan akan mengulang terlalu cepat, sebagai contoh bilangan 0000.

Neumann merasa tidak perlu melakukan perbaikan terlalu mendalam terhadap algoritma yang ia ciptakan karena saat itu metode ini mencukupi pekerjaannya.



Gambar 2.6 Ilustrasi algoritma Neumann pada 2 digit angka (<https://svijaykoushik.github.io/blog/2019/10/04/three-awesome-ways-to-generate-random-number-in-javascript/>)

Berikut beberapa contoh algoritma PBAS:

#### a. Metode Linear Kongruen

Metode linear kongruen adalah algoritma PBAS, termasuk PBAS baik yang pertama ditemukan. LCM memanfaatkan model linier untuk membangkitkan bilangan acak yang didefinisikan dengan:

$$X_{n+1} = (aX_n + c) \bmod m$$

Dengan ketentuan sebagai berikut:

- $m > 0$
- $0 < a < m$
- $0 \leq c < m$
- $0 \leq X_0 < m$
- $PBB(c,m) = 1$
- $a-1$  dapat dibagi faktor prima dari  $m$
- $a-1$  kelipatan 4 jika  $m$  kelipatan 4
- $a$  yang sangat besar meningkatkan efektivitas algoritma.

#### b. Mersenne Twister

Mersenne Twister adalah PBAS yang paling banyak diimplementasikan dalam dunia komputer. Nama PBAS ini diambil karena periode bilangan hasil adalah bilangan prima Mersenne pilihan. Algoritma ini dibuat tahun 1997 untuk menutupi kelemahan-kelemahan dari PBAS sebelumnya.

Mersenne Twister digunakan sebagai PBAS standar dalam berbagai bahasa pemrograman, seperti Python, R, Matlab, dan PHP.

```
// Create a Length n array to store the state of the generator
int[0..n-1] MT
int index := n-1
const int lower_mask = (1 << r) - 1 // That is, the binary number of r 1's
const int upper_mask = lowest w bits of (not lower_mask)

// Initialize the generator from a seed
function seed_mt(int seed) {
  MT[0] := seed
  for i from 1 to (n - 1) { // Loop over each element
    MT[i] := lowest w bits of (f * (MT[i-1] xor (MT[i-1] >> (w-2))) + i)
  }
}

// Extract a tempered value based on MT[index]
// calling twist() every n times
function extract_number() {
  if index >= n {
    error "Generator was never seeded"
    // Alternatively, seed with constant value; 5489 is used in reference C code[53]
  }
  twist()
}

int y := MT[index]
y := y xor ((y >> u) and d)
y := y xor ((y << s) and b)
y := y xor ((y << t) and c)
y := y xor (y >> 1)

index := index + 1
return lowest w bits of (y)
}

// Generate the next n values from the series x_i
function twist() {
  for i from 0 to (n-1) {
    int x := (MT[i] and upper_mask)
    + (MT[(i+1) mod n] and lower_mask)
    int xA := x >> 1
    if (x mod 2) != 0 { // Lowest bit of x is 1
      xA := xA xor a
    }
    MT[i] := (MT[(i + m) mod n] xor xA)
  }
  index := 0
}
```

Gambar 2.7 Pseudocode dari Mersenne Twister ([https://en.wikipedia.org/wiki/Mersenne\\_Twister](https://en.wikipedia.org/wiki/Mersenne_Twister))

### III. METODOLOGI

Dalam pembuatan PBAS ini digunakan bahasa Python 3.7 dengan IDE Idle. Alasan pemilihan bahasa Python adalah karena kemudahan *syntax* dan luasnya modul yang dapat langsung diimpor. Kelemahan python adalah *compile time* yang lama pada program yang besar atau pengolahan data yang banyak, akan tetapi untuk realisasi fungsi PBAS ini kelemahan itu tidak terlalu berpengaruh.

#### 1. Rancangan Algoritma

Kelemahan utama dari PBAS adalah bilangan acak yang dihasilkan dapat diprediksi jika seseorang mengetahui bilangan pancangan dan algoritma yang digunakan. Oleh karena itu, dalam perancangan PBAS ini digunakan variabel yang selalu berubah ubah, dalam kasus ini saya menggunakan waktu.

Bilangan pancangan yang digunakan akan bergantung dengan waktu sehingga jika waktu tidak diketahui maka bilangan hasil tidak dapat ditebak. Selain itu dengan adanya variabel berubah periode dari PBAS ini menjadi sangat besar selama digunakan tidak secara terus menerus.

PBAS ini dirancang untuk kasus-kasus umum dan bukan untuk kasus spesifik yang mengacak bilangan beberapa kali dalam waktu yang singkat(karena PBAS ini menunggu perubahan detik). Selain waktu, saya juga menggunakan tanggal dalam bilangan pancangan agar semakin teracak.

Dari segi spesifikasi sendiri, PBAS ini memiliki masukan berupa *integer* yang jumlah digit bilangan acak yang ingin dihasilkan.

## 2. Modul yang Digunakan

Digunakan modul `datetime` bawaan Python 3.7 untuk mendapatkan tanggal dan waktu.

## 3. Realisasi Fungsi

0. Disiapkan fungsi `countdigit` untuk menghitung digit sebagai berikut:

```
def countdigit(bil):  
    return len(str(bil))
```

1. Pertama-tama saya membuat variabel *integer* guna menyimpan waktu:

```
hour = now.hour  
minute = now.minute  
second = now.second  
time = hour + minute + second  
date = now.day*(10**6) + now.month*(10**4) + now.year
```

2. Lalu deklarasi fungsi PBAS, dengan *parameter* integer *n*.

3. Jika `time = 0` (kasus tengah malam) maka PBAS akan tidak menghasilkan apapun.

4. Selain itu, bilangan pancangan dideklarasikan, yakni: `date` div `time`.

5. Bilangan pancangan diubah menjadi bilangan 2 digit dalam `while` loop

6. Bilangan pancangan kemudian dipindahkan ke `while` loop berikutnya dan dioperasikan hingga memiliki digit yang diinginkan.

7. Bilangan pancangan dikembalikan.

Realisasi fungsi ini sangat terinspirasi metode tengah-kuadrat John von Neumann.

## IV. ANALISIS FUNGSI

### 1. Studi Kasus

- Kasus  $n = 1, n = 2, n = 3, n = 5, n = 7$ :

```
>>> PBAS (1)  
3  
>>> PBAS (2)  
36  
>>> PBAS (3)  
315  
>>> PBAS (5)  
31526  
>>> PBAS (7)  
3152698
```

Gambar 4.1 Studi Kasus 1

```
>>> PBAS (1)  
6  
>>> PBAS (2)  
60  
>>> PBAS (3)  
878  
>>> PBAS (5)  
87805  
>>> PBAS (7)  
8780542  
>>> |
```

Gambar 4.2 Studi Kasus 2

```
>>> PBAS (1)  
3  
>>> PBAS (2)  
38  
>>> PBAS (3)  
355  
>>> PBAS (5)  
35515  
>>> PBAS (7)  
3551569  
>>>
```

Gambar 4.3 Studi Kasus 3

### 2. Kelemahan fungsi

- Seperti yang diamati, pada jarak waktu yang dekat PBAS ini tidak begitu variatif
- Memiliki batas besar angka, yakni 9 digit

### 3. Kompleksitas Waktu

Dengan realisasi sebagai berikut:

```
def PBAS (n):  
    if (time==0):  
        return  
    else:  
        seed = date//time  
        while (countdigit (seed) != 2):  
            if (countdigit (seed) < 2):  
                seed = seed*5  
            else:  
                seed = seed//7  
        while (countdigit (seed) != n):  
            if (countdigit (seed) < n):  
                seed = seed%time  
                seed = seed*date  
                seed = seed//time  
            else:  
                seed = seed//10  
    return seed
```

Gambar 4.4 Realisasi Fungsi

Pada loop teratas diamati bahwa pada kasus terburuk akan dilakukan sekitar  $|countdigit(seed)-2|$  pengurangan, maka dalam notasi Big-O adalah  $O(1)$ .

Pada loop bawah, mirip dengan atasnya, dilakukan sekitar  $|n-2|$  kali pengulangan dalam kasus terburuk. Sehingga dalam Big-O adalah  $O(n)$ .

Dapat disimpulkan karena Big-O adalah notasi yang membatasi dari atas maka secara keseluruhan program ini memiliki kompleksitas waktu  $O(n)$ .

## V. KESIMPULAN

Bilangan acak adalah bilangan yang kemunculannya baik secara tunggal maupun berurutan tidak dapat diprediksi dengan pasti dan harus mengandalkan statistik untuk menebak kemunculannya. Dalam kehidupan sehari-hari banyak hal acak yang dapat terjadi, namun dalam ranah pemrograman "keacakan" adalah hal abstrak yang tidak dapat diukur. Dengan demikian dengan perhitungan matematis biasa tidak akan bisa dihasilkan bilangan acak asli di komputer.

Lain halnya dengan bilangan acak semu, ialah sesuatu yang dapat dihitung mesin dan seringkali menutup kebutuhan akan bilangan acak asli. Terdapat kriteria-kriteria yang menilai apakah sebuah urutan (atau tunggal) bilangan acak semu dapat dibandingkan dengan saudaranya.

PBAS adalah algoritma yang memberi keluaran bilangan acak semu dan pada masa ini algoritma PBAS sangat maju dan mencukupi kebutuhan akan bilangan acak asli.

## VI. UCAPAN TERIMA KASIH

Penulis berterima kasih kehadirat Allah SWT. atas nikmat dan hidayah yang Ia karuniai kemudian rasul-rasulNya yang telah bedakwah menebar ajaran kebenaran di muka bumi.

Terima kasih kepada:

1. Pak Rinaldi, dosen pembimbing sekaligus pengajar IF2120 kelas saya. Terima kasih atas bimbingan bapak, maaf saya tidak menjadi mahasiswa yang berprestasi di Mata Kuliah ini

2. Para Asisten Dosen, kak Ayyub, kak Aqil, kak William Fu, kak Fadhil dan kakak-kakak asisten lain. Semoga kita dapat segera menjalin silaturahmi di Labtek V.

3. Teman-teman Async, teman seperjuangan perkuliahan tingkat 2 IF ini.

4. Teman-teman SMA, para sahabat yang menjadi tempat melepas jenuh perkuliahan.

5. Kedua Orang Tua saya, yang jasanya lebih dari membiayai kuliah, namun segala ilmu, kasih sayang, candaan, emosi, dan kebersamaan. Semoga Ayah dan Ibu menjadi orang yang diridhoi Allah dan saya menjadi anak yang sholeh.

6. Orang-orang yang spesial yang belum sempat saya sebut

## DAFTAR PUSTAKA

- [1] Barker, Elaine; Barker, William; Burr, William; Polk, William; Smid, Miles (2012). "Recommendation for Key Management"(PDF). NIST Special Publication 800-57. NIST. Diakses pada 11 Desember 2020
- [2] Matsumoto, M.; Nishimura, T. (1998). "Mersenne twister: 623-dimensionally equidistributed uniform pseudo-random number generator" (PDF). *ACM Transactions on Modeling and Computer Simulation*.
- [3] Munir, Rinaldi. (2020). "Kompleksitas Algoritma(Bagian 1)". Diambil dari <http://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Kompleksitas-Algoritma-2020-Bagian1.pdf> pada 11 Desember 2020
- [4] Munir, Rinaldi. (2020). "Teori Bilangan(Bagian 1)". Diambil dari <http://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Teori-Bilangan-2020-Bagian1.pdf> pada 11 Desember 2020

## 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, 12 Desember 2020


Ttd ,



Nama dan NIM  
Mohammad Dwinta Harits Cahyana/13519041

## LAMPIRAN

### Kode Program:

 PBASwithTime.py - C:/Users/LENOVO/AppData/Local/Programs/Python/Python37

File Edit Format Run Options Window Help

```
from datetime import datetime

def countdigit(bil):
    return len(str(bil))

# datetime object containing current date and time
now = datetime.now()

hour = now.hour
minute = now.minute
second = now.second
time = hour + minute + second
date = now.day*(10**6) + now.month*(10**4) + now.year

def PBAS(n):
    if(time==0):
        return
    else:
        seed = date//time
        while(countdigit(seed) != 2):
            if(countdigit(seed) < 2):
                seed = seed*5
            else:
                seed = seed//7

        while(countdigit(seed) != n):
            if(countdigit(seed) < n):
                seed = seed%time
                seed = seed*date
                seed = seed//time
            else:
                seed = seed//10
    return seed
```