Pengujian Tingkat Keacakan dan Kecepatan Beberapa Algoritma Pembangkit Bilangan Acak Semu dengan Perbandingan Standar Deviasi dan Perbandingan Waktu Pembangkitan

Kevin Wibowo-13509065
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13509065@std.stei.itb.ac.id

Abstraksi— ***Pembangkit bilangan acak semu (*Pseudoraandom generator) *adalah salah satu jenis pembangkit bilangan acak yang menggunakan rumus sehingga bilangan yang dibangkitkan tidak sepenuhnya acak tapi bedasarkan kondisi tertentu. Bilangan yang dibangkitkan oleh pembangkit bilangan acak memiliki ciri-ciri yang mirip dengan bilangan acak, akan tetapi pembangkitannya akan lebih cepat daripada pembangkitan bilangan acak biasa.***

Bilangan ini mempunyai pembangkitan yang cepat serta pembangkitan serta mudah dibuat ulang sehingga mudah digunakan untuk simulasi program tertentu. Bilangan ini juga biasa digunakan untuk kriptografi . Terdapat berbagai jenis pembangkit bilangan acak semu seperti* linear congruental*, dan* lagged fibbonaci generator*. Rumus pada* chaos theory *dan* chaos map*, dapat juga digunakan sebagai pembangkit tingkat keacakan.

Oleh karena itu untuk membandingkan mana yang lebih baik akan dilakukan percobaan berupa pengukuran kecepatan perbandingan dan juga pengukuran hasil pembangkitan bilangan dan mengukur tingkat keacakan dengan graf standar deviasi. Setelah itu akan dilakukan analisis mana algoritma yang lebih aman untuk pembangkitan bilangan acak untuk algoritma kriptografi, dan beberapa saran peningkatan kualitas algoritma.

***Kata Kunci*— Pseudorandom generator*, bilangan acak, standar deviasi*, Chaos theory**

I. DASAR TEORI

1.1 *Pseudorandom number generator*

Pseudorandom number generator (pembangkit bilangan acak semu) adalah algoritma yang digunakan untuk membangkitkan bilangan yang mempunyai sifat yang mirip dengan bilangan acak. Bilangan yang dibangkitkan dengan bilangan acak ini tidak sepenuhnya acak tetapi bedasarkan rumus dengan nilai awal yang acak.

Walaupun bilangan yang sepenuhnya acak dapat dibangkitkan dengan *hardware random number generator*, pembangkit ini dapat membangkitkan bilangan cukup cepat dan acak sehingga penggunannya sangat popular di berbagai jenis aplikasi komputer. Sifat statistik yang baik diperlukan untuk pembangkit bilangan acak semu dan hal ini dapat dipenuhi oleh beberapa pembangkit seperti *linear congruental generator, lagged Fibonacci generator,* dan *linear feedback shift register.*

Beberapa aplikasi kriptografi juga membutuhkan bilangan yang tidak linear dan sulit ditebak oleh karena itu terdapat juga algoritma seperti *Blum Blum Shub, Fortuna,* dan *Mersene Twist.* Pembangkit bilangan acak tersebut disebut *cryptographically secure pseudorandom generator*.

Sebuah pembangkit bilangan acak semu selalu mempunyai keadaan awal yang disebut *seed state.* Dan kebanyakan pembangkit bilangan acak semu selalu mempunyai hasil yang sama apabila state awalnya sama. Pembangkit bilangan acak semu juga memiliki periode yang dihitung bedasarkan jumlah *state* yang ada karena semua bilangan acak pasti akan kembali ke state yang sama setelah melewati satu periode.

Pada awalnya algoritma pembangkit bilngan semu acak yang digunakan adalah *mean-square method.* Dimana dipilih sebuah bilangan lalu bilangan tersebut dikuadratkan dan diambil digit yang berada ditengahnya dan bilangan tengah tersebut akan digunakan untuk iterasi selanjutnya.

Algoritma ini tidak terlalu baik karena periodenya yang sangat rendah dan tingkat keacakannya sangat rendah oleh karena itu diciptakan beberapa algoritma lain seperti *Linear Congruental Generator, Blum Blum Shub,* dan masih banyak lagi.

1.2 *Linear Congruental Generator*

Linear Congruental Generator adalah salah satu jenis pembangkit bilangan acak yang paling sering digunakan di beberapa aplikasi. Algoritma ini mudah diaplikasikan dan menggenerasikan bilangan acak semu dengan cepat.

Rumus dari algoritma ini adalah:

$$X_n = (aX_{n-1} + b) \bmod m$$

Dimana:

Xn :bilangan acak yang dibangkitkan.

a :bilangan pengali

b :bilangan penambah

m :bilangan modulo

1.3 *Inversive Congruential Generator*

Inversive Congruential Generator adalah salah satu jenis pembangkit bilangan acak semu yang mirip dengan *Linear Congruential Generator* .Perbedaannya adalah sebagai penganti bilangan pengali digunakan bilangan pembagi pada rumus dalam algoritma pembangkit bilangannya.

Rumus dari algoritma ini adalah:

$$X_n = (a(X_{n-1})^{-1} + b) \bmod m$$
$$X_n = 0 \text{ apabila } X_{n-1} = 0$$

Dimana:

Xn :bilangan acak yang dibangkitkan.

a :bilangan pembagi

b :bilangan penambah

m :bilangan modulo

1.4 *Blum Blum Shub*

Blum Blum Shub adalah salah satu jenis *cryptographically secure pseudorandom generator yang sering digunakan.* Algoritma ini diciptakan oleh Lenore Blum, Manuel Blum dan Michael Shub pada tahun 1986.

Rumus dari algoritma ini adalah:

$$x_i = x_{i-1}^2 \bmod n$$
$$z_i = LSB(x_i)$$
$$n = pq$$
$$x_0 = s^2 \bmod n$$
$$0 < s < n$$

Dimana:

X*i* :bilangan acak yang dibangkitkan.

Z_i :bilangan acak yang digunakan

p :bilangan prima yang kongruen dengan 3 mod 4

q :bilangan prima yang kongruen dengan 3 mod 4

s :bilangan yang relatif prima dengan n

1.5 *Chaos Theory*

Chaos theory adalah salah satu bidang di matematika yang diaplikasikan di berbagai bidang lainnya seperti fisika, biologi, komputer dan bahkan filosofi. *Chaos theory* mempelajari sitem yang dinamis serta sangat dipengaruhi oleh kedaan awal, suatu kondisi yang sering disebut dengan *butterfly effect*.

Dalam sistem ini, suatu perbedaan di keadaan awal dapat membuat perubahan besar di keadaan selanjutnya, sehingga membuat sistem sulit diprediksi dalam jangak panjang. Perubahan kecil seperti pembulatan desimal dapat membuat sistem menjadi kacau, hal ini juga terjadi di sistem yang bahkan tidak ada faktor acak sama sekali.

Chaos theory ini dapat digunakan sebagai pembangkit bilangan acak semu di bidang teknologi informasi.

Beberapa con toh sistem dalam *chaos theory* yang *Arnold's cat map* digunakan di dalam bidang teknik informatika adalah *logistic map, henon map*, dan

1.6 *Logistic Map*

Logistic Map adalah salah satu jenis pemetaan dari fungsipolinomial yang menunjukan bagaimana suatu sistem dapat menunjukan tingkat keacakan yang tinggi. Sehingga menunjukan contoh dari *chaos theory*. Pemetaan ini juga dapat dibuat menjadi salah satu algoritma pembangkit bilangan acak semu dalam bidang teknik informatika.

Rumus dari algoritma ini adalah:

$$X_{i+1} = r.X_i(1-X_i)$$

Dimana:

Xi :bilangan acak yang dibangkitkan.$(0 \le X_i \le 1)$

r :bilangan pengali(laju pertumbuhan) $(0 \le r \le 4)$

1.6 *Arnold's Cat Map*

Arnold's Cat Map adalah salah satu jenis pemetaan *chaos theory* yang diperkenalkan oleh Vladimir Arnold dengan gambarans seekor kucing, sehingga disebut *Arnold's Cat Map*.

Rumus dari algoritma(dengan modifikasi dari rumus pemetaan) ini adalah:

$$f(x,y) = \begin{cases} (2x, y/2), & 0 \le x \le 1/2, 0 \le y \le 1 \\ (2x-1, (y+1)/2), & 1/2 \le x \le 1, 0 \le y \le 1 \end{cases}$$

Dimana:

x :bilagan acak 1

y :bilangan acak 2

II. IMPLEMENTASI UNTUK PENGUJIAN

Algoritma-algoritm diatas diimplementasikan untuk diujikan. Implementasi dibuat dalam bahasa Java:

Linear Congruental Generator:

```
public class lcg {
   int a;
   int b;
   int x;
   int m;
  lcg(int newa, int newb,int newm, int x0)
   {this.a=newa;
    this.b=newb;
    this.m=newm;
```

```
        this.x=x0;
        }
       int generate()
        {int temp;
          temp=((this.a*this.x)+this.b)% this.m;
         this.x=temp;
          return temp;
      }
}
```

Inversive Congruential Generator:

```
    public class lcg {
       int a;
       int b;
       int x;
       int m;
      lcg(int newa, int newb,int newm, int x0)
       {this.a=newa;
        this.b=newb;
        this.m=newm;
       this.x=x0;
       }
      int generate()
       {int temp=0;
        if(this.x!=0){
       temp=((this.a/this.x)+this.b)% this.m;
          }
         this.x=temp;
          return temp;
      }
}
```

Blum Blum Shub:

```
    public class BBS {
       int x;
       int p;
       int q;
       int s;
       int n;
    BBS(int newp,int newq,int news)
       {this.p=newp;
        this.q=newq;
        this.n=this.p*this.q;
        this.s=news;
        this.x=(news*news) % this.n;
        this.z=new String();
       }
       int generate()
       {int temp1;

       temp1=(this.x*this.x) % this.n;
       this.x=temp1;
       if(temp1%2==0)
       {this.z=this.z+'0';}
       else this.z=this.z+'1';
    return Integer.parseInt(this.z,2);}
}
```

Logistic Map:

```
    public class LM {
    double x;
    double r;
    LM(double newr, double newx)
       {this.x=newx;
       this.r=newr;}
    double generate()
       {double temp;
       temp=this.r*this.x*(1-this.x);
       this.x=temp;
       return temp;}
}
```

Arnold's Cat Map:

```
    public class ACM {
    double x;
    double y;
    ACM(double newx, double newy)
       {this.x=newx;
       this.y=newy;}
    double generate()
       {if(this.x<=0.5)
             {this.x=2*this.x;
             this.y=this.y/2;}
       else
          {
          this.x=(this.x*2)-1;
          this.y=(this.y+1)/2;
          }
        return this.x;
       }
}
```

Perhitungan waktu(dalam nanosecond):

```
    long start = System.nanoTime();
    int n=20;//diganti sesuai kebutuhan
    for(int i=1;i<n;i++)
    {
    //(fungsi pembangkit bilangan acak)
    }
    long end = System.nanoTime();
     long elapsedTime = end - start;
  System.out.println("time:"+elapsedTime);
```

III. DATA PENGUJIAN

Berikut beberapa pengujian yang telah dilakukan:

Linear Congruential Generator:

(1) Masukan:
a:32
b:13
x0:2
m:17
Hasil:

n	Xn	Waktu generasi (nanosec)		
1	9	787251	814838	888451
2	12	890197	1083239	850807

3	6	895435	882319	912319
4	1	922614	924140	888044
5	11	951657	928819	960387
6	8	944953	1027715	1013327
7	14	1016121	1011651	1039937
8	2	1026248	1493486	1342000
9	9	1134013	1477232	1056559
10	12	1138412	1160133	1481911
15	14	1873213	1479657	1772222
20	1	1905899	1905899	1932412
50	12	2301689	2723182	2312165
100	1	5290686	4928280	17351577

Tabel 4.1 Data pengujian(1) dengan *Linear Congruential Generator*

(2) Masukan:
a:3241
b:1312
x0:25
m:171
Hasil:

n	Xn	Waktu generasi (nanosec)		
1	86	784737	804838	838451
2	111	822311	869664	808343
3	82	862680	862051	863377
4	143	882864	919670	876879
5	168	920438	934561	971234
6	139	982597	946838	971981
7	29	1021149	1031245	1123515
8	54	1418127	1026527	1142120
9	25	1072482	1045245	1211117
10	86	1090502	1224458	1353073
15	139	1221175	1229695	1356038
20	111	1894305	1894305	2073239
50	168	3029436	2779823	2843658
100	86	4012381	3949594	4292026

Tabel 4.2 Data pengujian(2) dengan *Linear Congruential Generator*

Inversive Congruential Generator:

(1) Masukan:
a:21
b:16
x0:6
m:18
Hasil:

n	Xn	Waktu generasi (nanosec)		
1	0.49275	890934	844412	828701
2	14	874513	851713	876236
3	14	882462	860240	883511
4	14	921692	920914	891205
5	14	950734	932118	968724
6	14	937832	997814	981835
7	14	962351	1111721	1072491
8	14	1165407	1012867	1001205
9	14	1176342	1024781	1056785
10	14	1138412	1160133	1150914
15	14	1221715	1696026	1239962
20	14	1921341	1886553	1789712
50	14	2375930	2314749	2922298
100	14	4484369	4832668	3914254

Tabel 4.3 Data pengujian(1) dengan *Inversive Congruential Generator*

(2) Masukan:
a:213
b:167
x0:90
m:142
Hasil:

n	Xn	Waktu generasi (nanosec)		
1	1	926445	929797	912362
2	3	995264	1214540	1012863
3	7	1063677	1166768	1059562
4	15	1067244	1081981	1076812
5	31	1407791	1368051	1348985
6	31	958912	991914	930123
7	31	1000616	1003759	961505
8	31	978458	1022546	1022546
9	31	1077651	1106566	1045175
10	31	1159365	1258610	1098254
15	31	1393682	1740724	1348355
20	31	1422108	1536597	1415124
50	31	2935359	2521270	2925581
100	31	4493798	4740407	4952806

Tabel 4.4 Data pengujian(2) dengan *Inversive Congruential Generator*

Blum Blum Shub:

Pengujian ini dialkukan hanya sampai n=31 karena terdapat batasan panjang tipe integer. Apabila menggunakan tipe BigInteger atau sejenisnya maka perhitungan dipastikan akan berlangsung lebih lama.

(1) Masukan:
p:11
q:23
s:3
Hasil:

n	Xn	Waktu generasi (nanosec)		
1	1	1176616	967581	935593
2	2	996216	981131	985111
3	4	1099511	1007670	1011650
4	9	1051251	1051251	1044965
5	18	1079746	1075765	1198895
6	37	1037613	1126819	1128775

7	74	1273974	1132451	1143162
8	148	1208813	1336204	1269156
9	296	1600134	1276908	1375804
10	592	1540559	1370356	1455912
15	18957	1513041	1692674	1749314
20	606643	1990547	1815804	1715162
25	19412594	2450731	2002000	2226610
31	1242406048	2264743	2594883	2484959

Tabel 4.5 Data pengujian(1) dengan *Blum Blum Shub*

(2) Masukan:
p:19
q:23
s:13
Hasil:

n	Xn	Waktu generasi (nanosec)		
1	0	969537	937200	926095
2	1	973493	983417	989246
3	2	989352	1004621	1020555
4	4	946930	1052421	1057351
5	8	1149216	1182015	1045803
6	17	1068323	1182461	1102160
7	35	1171254	1124617	1484122
8	70	1250419	1214159	1143969
9	141	1294787	1571150	1282426
10	282	1301911	1335017	1352336
15	9024	1518489	1562561	1254145
20	288790	1912745	1834584	1919462
25	9241290	1999765	2124670	2223620
31	591442594	2930749	2516720	2712416

Tabel 4.6 Data pengujian(2) dengan *Blum Blum Shub*

Logistic Map:

(1) Masukan:
r:4.0
X0:0.456
Hasil:

n	Xn	Waktu generasi (nanosec)		
1	0.9922560000000001	1082610	1195473	1054883
2	0.030736121855999457	1129613	1140648	1272089
3	0.1191656506	1294857	1114559	1273765
	7701043			
4	0.4198607935029406	1277848	1236610	1264146
5	0.9743108303280866	1287663	1333480	1325937
6	0.1001169449339240 6	1395569	1290806	1369587
7	0.3603741690840867	1364210	1370984	1481194
8	0.9220185093641632	1598667	1386489	1385729
9	0.2876015110161988 5	1485371	1484051	1498714
10	0.8195475275095924	1544609	1524286	1631410
15	0.9905115518348093	2125968	2083086	2198184
20	0.0003827329955062 7024	3292736	2890032	2371740
50	0.5062101991 81858	4544432	5336782	6253798
100	0.9966049820458703	17354301	9504769	24231990

Tabel 4.7 Data pengujian(1) dengan *Logistic Map*

(2) Masukan:
r:2.5
X0:0.36
Hasil:

n	Xn	Waktu generasi (nanosec)		
1	0.576	1114984	1177175	1076813
2	0.61056	1112712	1122210	1105448
3	0.594441216	1387956	1153568	1278844
4	0.6027021418011034	1261962	1231442	1209861
5	0.5986	1287663	1378997	1261962

	306751736651			
6	0.6006799747869674	1326355	1387956	1330266
7	0.5996588566922391	1342477	1406457	1482451
8	0.6001702807069894	1449201	1522565	1715581
9	0.5999147871577074	1693581	1472953	1537765
10	0.6000425882680751	1712509	1458146	1534902
15	0.5999986689 21837	2064438	2270959	1901080
20	0.6000000415960023	2056755	2368089	1941620
50	0.6	5077880	4687188	4632292
100	0.9966049820458703	11167900	7698744	10602116

Tabel 4.8 Data pengujian(2) dengan *Logistic Map*

Arnold's Cat Map:

(1) Masukan:
x:0.6
y:0.2
Hasil:

n	Xn	Waktu generasi (nanosec)		
1	0.1999999999999996	1157192	1096261	1357161
2	0.3999999999999999	1049613	1156783	1273789
3	0.7999999999999998	1330617	1112559	1299683
4	0.5999999999999996	1302235	1220337	1218171
5	0.1999999999999993	1303791	1269366	1436077
6	0.3999999999999986	1383898	1447334	1340534
7	0.7999999999999972	1457822	1460921	1458496
8	0.5999999999999943	1517721	1398032	1531041
9	0.1999999999998863	1419779	1445505	1430483
10	0.3999999999997726	1653562	1513321	1697207
15	0.7999999999992724	2288490	2212852	2056521
20	0.5999999999767169	2808806	2465467	1973016
50	0.375	3287848	3803976	4394623
100	1	5704915	5980439	6514795

Tabel 4.9Data pengujian(1) dengan *Arnold's Cat Map*

(1) Masukan:
x:0.123
y:0.456
Hasil:

n	Xn	Waktu generasi (nanosec)		
1	0.246	1124375	1287244	1034527
2	0.492	1049351	1195822	1134186
3	0.984	1330617	1112559	1299683
4	0.968	1212934	1222705	1191981
5	0.9359999999999999	1273137	1226234	1290861
6	0.8719999999999999	1342254	1326013	1379784
7	0.7439999999999998	1530572	1460921	1458496
8	0.4879999999999955	1517721	1529537	1387396
9	0.9759999999999991	1620508	1447740	1432451
10	0.9519999999999982	1623041	1532531	1561023
15	0.4639999999	1839898	1747428	1712508

	999418			
20	0.8479999999981374	2156419	1977347	1973924
50	0.75	3891416	3803976	3962096
100	1	6277962	6503200	6148407

Tabel 4.10Data pengujian(2) dengan *Arnold's Cat Map*

IV. PERHITUNGAN DAN ANALISIS

Berikut tabel dari hasil pengujian di atas:

No. tabel	Rata-rata	Standar Deviasi	Batasan nilai sesuai algoritma
4.1	8	4.17	0-16
4.2	101.9286	44.73967	0-170
4.3	13.0352	3.478649	0-17
4.4	24	11.43303	0-141
4.5	90174673	319610186.9	0-231
4.6	42927304	152149193.4	0-231
4.7	0.537121	0.383709326	0-1
4.8	0.6271	0.102727709	0-1
4.9	0.526786	0.247210199	0-1
4.10	0.765714	0.236543994	0-1

Tabel 5.1 Hasil perhitungan 1

No. tabel	Rata-rata Waktu Generasi	Rata-rata Waktu Generasi per n
4.1	1826693	130478.1
4.2	1434732	102480.9
4.3	1426871	101919.3
4.4	1516047	108289
4.5	1415240	101088.5
4.6	1395626	99687.57
4.7	2911516	207965.4
4.8	2309615	164972.5
4.9	2005501	143250.1
4.10	1955495	139678.2

Tabel 5.2 Hasil perhitungan 2

No. tabel	Persentase Persebaran	Pergesaran dari rata-rata
4.1	52%	0%
4.2	52%	9%
4.3	35%	74%
4.4	16%	80%
4.5	29%	95%
4.6	14%	98%
4.7	72%	3%
4.8	20%	12%
4.9	48%	2%
4.10	46%	26%

Tabel 5.3 Hasil perhitungan 3

Berikut hasil analisis dari perhitungan diatas dan data hasil uji:

Linear Congruential Generator:
(1) Memiliki waktu generasi yang cukup cepat
(2) Memiliki persebaran yang tinggi
(3) Memiliki persebaran yang berada di tengah
(4) Memiliki periode generasi
(5) Tingkat keacakan yang tidak bergantung pada n

Inversive Congruential Generator:
(1) Memiliki waktu generasi yang cukup cepat
(2) Memiliki persebaran yang sedang
(3) Memiliki persebaran yang tidak berada di tengah
(4) Periode sangat rendah
(5) Tingkat keacakan rendah saat n tinggi

Blum Blum Shub:
(1) Memiliki waktu generasi yang paling cepat
(2) Persebaran sulit diukur karena menggenerasikan secara unik
(3) Generasi lebih bergantung pada n daripada state awal
(4) Baik untuk generasi nilai yang sangat besar

Logistic Map:
(1) Memiliki waktu generasi yang sangat lambat
(2) Memiliki persebaran yang bergantung pada kondisi awal dengan nilai sedang ke tinggi
(3) Memiliki persebaran yang berada di tengah
(4) Sangat bergantung pada kondisi awal
(5) Tingkat keacakan rendah saat n tinggi

Arnold's Cat Map:
(1) Memiliki waktu generasi yang lambat
(2) Memiliki persebaran yang bergantung pada kondisi awal dengan nilai sedang ke tinggi
(3) Memiliki persebaran yang berada di tengah
(4) Sangat bergantung pada kondisi awal
(5) Tingkat keacakan rendah saat n tinggi

V. KESIMPULAN

1. Terdapat berbagai *pseudorandom number generator* yang dapat digunakan. Masing-masing terdapat kelebihan dan kekurangan.

2. Beberapa *pseudorandom number generator* memiliki keacakan yang bergantung pada state awal.

3. Tingkat keacakan pada algoritma *chaos*

theory semakin berkurang dengan jumlah n yang besar

4. *Chaos theory* sangat bergantung pada kondisi awal

5. *Linear Congruential Generator* sangat baik dipakai di aplikasi biasa sedangkan *Blum Blum Shub* sangat baik dipakai untuk aplikasi yang membutuhkan keamanan tinggi dalam generasi bialngan acaknya

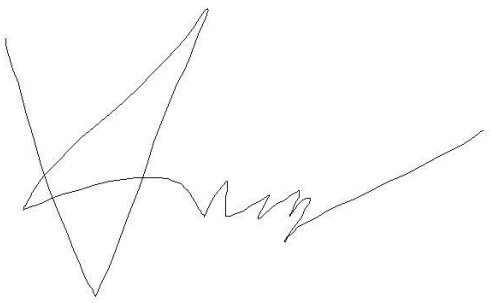
REFERENCES

[1] http://en.wikipedia.org/wiki/Pseudorandom_number_generator
[2] http://informatika.stei.itb.ac.id/~rinaldi.munir/Kriptografi/2010-2011/kripto10-11.htm
[3] http://en.wikipedia.org/wiki/List_of_pseudorandom_number_generators
[4] http://en.wikipedia.org/wiki/Linear_congruential_generator
[5] http://en.wikipedia.org/wiki/Inversive_congruential_generator
[6] http://en.wikipedia.org/wiki/Blum_Blum_Shub
[7] http://en.wikipedia.org/wiki/Chaos_theory
[8] http://en.wikipedia.org/wiki/Logistic_map
[9] http://en.wikipedia.org/wiki/H%C3%A9non_map

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 Mei 2012



Kevin Wibowo/13509065