Analisa Perbandingan Kompleksitas Algoritma *Bubble Sort, Cocktail Sort* Dan *Comb Sort* Dengan Bahasa Pemrograman C++

Devy Kumalasari
Program Studi Manajemen Informatika, AMIK BSI Pontianak
devy2610@bsi.ac.id

Abstract *- As the development of advances in technology and computer science, demand to find a method of solving problems in a fast, effective and powerful become a necessity, especially for classical problems, one of which is the sorting (sorting). Data sorting (sorting) plays an important role in many applications that the question refers to the amount of data and is often a problem that should be considered so that the whole problem can be solved well and quickly. Sorting algorithms used in this study is the Bubble Sort algorithm, Cocktail sort and Comb sort. Bubble Sort algorithm is an algorithm comparison by swapping adjacent elements if the element was smaller than the element afterwards. This algorithm is described briefly explanation and comparison of data sorting time complexity with integer elements. Data that will be tested with a range of 1000 to 100000 where the data at the time of compiling the program will seek otomasis then sorted. Of the three algorithms used to be inferred where the algorithm has the fastest time complexity. Based on the testing algorithm is applied to the programming language C ++ can be concluded that the algorithm has the fastest time complexity of the algorithm is the Comb sort.*
Keywords: Bubble Sort Algorithm, Cocktail Sort, Sort Comb, Sorting, Execution Time.

Abstrak - Sebagai pengembangan dari kemajuan teknologi dan ilmu komputer, permintaan untuk menemukan metode pemecahan masalah secara cepat, efektif dan kuat menjadi kebutuhan, terutama untuk masalah klasik, salah satunya adalah pemilahan (sorting). Data menyortir (sorting) memainkan peran penting dalam banyak aplikasi bahwa pertanyaan mengacu pada jumlah data dan sering merupakan masalah yang harus dipertimbangkan sehingga seluruh masalah dapat diselesaikan dengan baik dan cepat. Algoritma pengurutan yang digunakan dalam penelitian ini adalah algoritma Bubble Sort, Cocktail Sort dan Comb Sort. Algoritma Bubble Sort adalah perbandingan algoritma dengan menukar elemen yang berdekatan jika elemen lebih kecil dari elemen sesudahnya. Algoritma ini dijelaskan penjelasan singkat dan perbandingan data menyortir kompleksitas waktu dengan elemen integer. Data yang akan diuji dengan berbagai 1000-100,000 dimana data pada saat kompilasi program akan mencari otomasis kemudian diurutkan. Dari ketiga algoritma yang digunakan untuk disimpulkan mana algoritma memiliki kompleksitas waktu tercepat. Berdasarkan algoritma pengujian yang diterapkan pada bahasa pemrograman C ++ dapat disimpulkan bahwa algoritma memiliki kompleksitas waktu tercepat dari algoritma ini adalah Comb Sort.
Kata Kunci: ***Bubble Sort Algorithm, Cocktail Sort, Sort Comb, Sorting, Waktu Eksekusi.***

1. PENDAHULUAN

Perkembangan ilmu pengetahuan dan teknologi semakin pesat salah satunya adalah teknologi komputer. Pengerjaan secara komputerisasi dapat membantu manusia untuk mempermudah pekerjaan baik itu aktivitas disekolah, perusahaan maupun instansi dimana peran komputer dapat bekerja lebih teliti dan akurat dalam pengolahan data.

Pengolahan data tidak dapat dilepaskan begitu saja dari kehidupan kita dan komputer pada umumnya digunakan untuk mengolah data. Hasil olahan data dapat dijadikan hasil untuk mengambil suatu keputusan, akan tetapi pengolahan data secara komputerisasi dibutuhkan algoritma sebagai langkah dan proses penyelesaian suatu masalah dimana akan ada masukan dan keluaran.

Algoritma adalah kumpulan instruksi yang dibuat secara jelas terhadap penyelesaian suatu masalah. Kumpulan instruksi tersebut dirancang ke dalam bahasa pemograman yang bisa dimengerti oleh komputer dengan tujuan akan mengahsilkan suatu *output* dan dpat menyelesaikan permsalahan secara efektif dan efisien.

Dalam analisis suatu algoritma terdapat bagian-bagian yang dapat dianalisis yaitu kecepatan waktu, kapasitas biaya dan kapistas ruang. Banyak metode algoritma yang digunakan pada proses pengolahan data salah satunya adalah pengurutan (*sorting*), baik itu pengurutan naik (*ascending*) maupun pengurutan menurun (*descending*). Pengurutan (*sorting*) adalah salah satu operasi yang digunakan dalam menyusun suatu obyek sehingga tertata dengan baik dapat berupa nilai, objek atau nama.

Pada pembahasan algoritma banyak algoritma pengurutan yang disajikan untuk mengurutkan data dan masing-masing algoritma memiliki kelebihan dan kekurangan masing-masing. Beberapa algoritma yang dimakasud diantaranya *Bubble sort*, *Selection*

sort, *Insertion Sort, Merge sort, Quick sort.* Dari algoritma yang disebutkan merupakan algoritma perbandingan (*Comparison*). Sedangkan yang termasuk algoritma non perbandingan diantaranya *Bucket sort* dan *Radix sort.*

Adapun tujuan yang ingin dicapai dalam penelitian ini adalah membandingkan algoritma yang digunakan dalam proses pengurutan data antara algoritma *Bubble sort* dan algoritma pengembangan dari *Bubble sort* yaitu *Cocktail sort* dan *Comb sort* untuk menentukan mana pengurutan yang memiliki kompleksitas waktu paling baik untuk pengurutan data 100 sampai 100000.

Rumusan masalah yang akan dibahas dalam penelitian ini adalah bagaimana menganalisis perbandingan kompleksitas waktu yang terpakai antara algoritma *Bubble sort*, *Cocktail sort* dan *Comb sort* pada saat pengurutan data dengan menggunakan bahasa pemrograman C++.

Batasan masalah dari penelitian ini adalah tipe struktur data yang digunakan berupa array dinamis, data yang akan diurutkan merupakan bilangan integer dengan range data 1000 smpai dengan 100.000 dan ketentuan proses pengurutan dilakukan hanya memasukkan jumlah data yang akan diurutkan sedangkan data yang akan diurutkan adalah data acak.

2. LANDASAN TEORI

2.1 Algoritma

Menurut Rinaldi (2007:4) Algoritma merupakan prosedur komputasi yang terdefinisi dengan baik yang menggunakan beberapa nilai sebagai masukan dan menghasilkan beberapa nilai yang disebut keluaran atau lebih detailnya disimpulkan bahwa algortima adalah deretan langkah komputasi yang mentransformasikan masukan menjadi keluaran.

Menurut Suarga (2012:1) algoritma merupakan susunan langkah yang pasti, yang bila diikuti maka akan mentransformasi data input dan output berupa informasi.

Menurut Donald Kunth dalam buku Surga (2012:4) Sebuah algoritma juga harus memenuhi kriteria sebagai berikut:

1. *Input*

Suatu algoritma memiliki input atau kondisi awal sebelum dilaksanakan, bisa berupa nilai-nilai peubah yang diambil dari himpunan kusus.

2. *Output*

Suatu algoritma akan mengahsilkan *output* setelah dilaksanakan atau algoritma akan mengubah kondisi awal menjadi kondisi akhir, dimana nilai output diperoleh dari nilai input yang telah diproses melalui algoritma.

3. *Definiteness*

Langkah-langkah yang dituliskan dalam algoritma terdefinisi dengan jelas sehingga mudah dilaksanakan oleh pengguna algoritma.

4. *Finiteness*

Suatu Algoritma harus memberi kondisi akhir setelah jumlah langkah yang terbatas jumlahnya dilakukan terhadap setiap kondisi awal atau input yang diberikan.

5. *Effectiveness*

Setiap langkah dalam algoritma bisa dilaksanakan dalam suatu selang waktu tertentu sehingga pada akhirnya didapatkan suatu yang diharapkan.

6. *Generality*

Setiap langkah alhoritma berlaku untuk setiap himpunan input yang sesuai dengan persoalan yang diberikan, tidak hanya untuk himpunan tertentu.

Algoritma merupakan susunan atau struktural yang diaplikasikan ke dalam bahasa komputer atau pemrograman dengan tujuan membantu dalam menyelesaikan permasalahan dimana akan ada data sebagai masukan dan keluaran sebagai hasil dari proses yang dilakukan.

2.2. Kompleksitas Algoritma

Kompleksitas dari suatu algoritma merupakan ukuran seberapa banyak komputasi yang dibutuhkan algoritma untuk menyelesaikan masalah. Algoritma yang dapat menyelesaikan suatu permasalahn dalam waktu yang singkat memiliki kompleksitas yang rendah, sementara algoritma yang membutuhkan waktu lama untuk menyelesaikan masalah mempunyai kompleksitas yang tinggi.

Setiap algoritma memiliki dua buah ciri khas yang dapat digunakan sebagai parameter pembanding, yaitu jumlah proses yang dilakukan dan jumlah memori yang digunakan untuk melakukan proses. Jumlah proses ini dikenal sebagai kompleksitas waktu yang disimbolkan dengan T(n), diukur dari jumlah tahapan komputasi yang dibutuhkan untuk menjalankan algoritma sebagai fungsi dari ukuran masukan n, dimana ukuran masukan (n) merupakan jumlah data yang diproses oleh sebuah algoritma. Sedangkan jumlah memori dikenal sebagai kompleksitas ruang yang disimbolkan dengan S(n), diukur dari memori yang digunakan oleh struktur data yang terdapat didalam algoritma sebagai fungsi dari masukan .(Ryan Rheinadi (2010)).

Dengan menggunakan kompleksitas waktu atau kompleksitas ruang, dapat ditentukan laju peningkatan waktu atau ruang yang diperlukan algoritma, seiring dengan meningkatnya ukuran masukan (n).

Dalam setiap algoritma, terdapat berbagai jenis operasi, di antaranya :
a. Operasi baca tulis.
b. Operasi aritmatika (+ - / *)
c. Operasi pengisian nilai (assignment)
d. Operasi pengaksesan elemen larik
e. Operasi pemanggilan fungsi ataupun prosedur

Berdasarkan kodisi dari kompleksitas algortima ada 3 kondisi yang dilihat dari beberapa kasus yaitu :
a. Kondisi *Best Case*
Waktu tempuh yang bernilai minimum dari suatu fungsi F(N) untuk setiap input atau disebut juga suatu keadaan yang terbaik dari suatu proses di dalam algoritma.
b. Kondisi *Worst Case*
Waktu tempuh yang bernilai maksimum dari suatu fungsi F(N) untuk setiap input atau keadaan ini disebut juga dengan keadaan analog terburuk.
c. Kondisi *Avarage Case*
Merupakan suatu keadaan dari suatu waktu tempuh yang equivalen dengan nilai eksperimentasi dari fungsi F(N) untuk setiap input.

2.3. Notasi Asimtotik

Nilai n cukup besar bahkan tidak terbatas, dilakukan analisis efisiensi asimtotik dari suatu algoritma untuk menentukan kompkesitas waktu yang sesuai atau disebut juga dengan kompleksitas waktu. asimptotik yang dinotasikan dengan "O" (baca : "O-besar"). Kompleksitas waktu asimptotik ini diperoleh dengan mengambil term terbesar dari suatu persamaan kompleksitas waktu. Sebagai contoh, dapat dilihat pada persamaan di bawah ini :

$T(n)=4n^3+5n^2+7n+3$……. (1)
O(n3)…… (2)

Dari persamaan (1) di atas diperoleh persamaan (2). Dapat dilihat bahwa nilai O adalah term terbesar dari T(n), tanpa faktor pengalinya. Berikut ini adalah daftar dari beberapa kelompok algoritma berdasarkan nilai O nya.

Notasi O dinyatakan *running time* dari suatu algoritma untuk kemungkinan kasus terburuk. Notasi O memiliki beberapa bentuk diantaranya:
a. Bentuk O(1)
Bahwa algoritma yang sedang dianalisis merupakan algoritma konstan. Hal ini mengindikasikan bahwa running time algoritma tersebut tetap, tidak bergantung pada n.
b. O(n)
Bahwa algoritma tersebut merupakan algoritma kuadratik artinya bila n menjadi 2n maka running time algoritma akan menjadi dua kali *running time* semua.
c. $O(n^2)$
Merupakan algoritma kuadratik, algoritma kuadratik biasa hanya digunakan untuk kasusu dengan n yang berukuran kecil. Sebab, bila n dinaikkan dua kali semula, maka running time algoritma akan menjadi empat kali semula.
d. $O(n^3)$
Merupakan Algoritma kubik. Pada algoritma ini bila n di naikkkan menjadi dua kali semula, maka *running time* algoritma akan menjadi delapan kali semula.
e. Bentuk $O(2^{n)}$
Algorima tersebut termasuk algoritma eksponensial. Pada kasus ini, bila n dinaikkan menjadi dua kali semula, maka running time algoritma menjadi kuadrat kali semula.
f. O(log n)
Merupakan algoritma logaritmik. Pada kasus ini, laju pertumbuhan waktu lebih lambat dari pada pertumbuhan n. algoritma ini memecahkan persoalan besar dengan mentransformasikan menjadi beberapa persoalan lebih kecil denga ukuran yang sama.
g. Bentuk O(n log n)
Algoritma ini membagi persoalan menjadi beberapa persoalan lebih kecil, menyelesaikan setiap persoalan secara independen, kemudian menggabungkan solusi masing-masing persoalan.

2.4. Pengurutan (*Sorting*)

Menurut Ema (2005:219) Pengurutan (sorting) diartikan sebagai proses penyusunan kembali sekumpulan elemen kedalam urutan tertentu. Tujuannya adalah adalah untuk memudahkan dalam pencarian dari suatu himpunan, selain itu juga dapat megetahui data terbesar dan terkecil.

Menurut Saniman dan Fathoni (2010 : 375) Pengurutan (*sorting*) adalah proses pengurutan data yang sebelumnya disusun secara acak sehingga menjadi susunan secra teratur menurut suatu aturan tertentu.

Berdasarkan dua teori diatas dapat disimpulkan bahwa pengurutan (sorting) merupakan susunan data yang diurutkan berdasarkan aturan untuk mempermudah dalam pencarian.

Pada dasarnya ada dua jenis pengurutan yang dilakukan pada proses pengurutan yaitu :
a. Urut Naik (*Ascending*)
Mengurutkan data dari yang paling kecil sampai paling besar.
b. Urut Turun (*Descending*)

Mengurutkan data dari yang paling besar ke paling kecil.

2.5. *Bubble* Sort

Menurut Yahya (2014:136) *Bubble sort* adalah metode pengurutan yang membandingkan elemen yang sekarang dengan elemen sesudahnya yaitu jika elemen pertama lebih besar dari elemen kedua maka dilakukan swap tapi jika tidak maka tidak perlu dilakukan swap. Proses sort dilakukan tahap per tahap, misalnya untuk n = 7 maka akan dilakukan (n – 1) = 6 tahap (mulai dari 0 sampai dengan n – 2).

Menurut Ema (2005: 231) Metode *Bubble sort* merupakan proses yang terjadi pada pengurutan dengan membandingkan dua data yang berdekatan. Diberikan nama "*Bubble*" karena konsep dari algoritmanya diibaratkan seperti gelembung air untuk elemen struktur data yang seharusnya pada posisi awal. *Bubble sort* mengurutkan data dengan cara membandingkan elemen pertama dengan elemen kedua, dimana cara kerjanya adalah dengan berulang-ulang melakukan proses *looping* (perulangan) terhadap elemen-elemen struktur data yang belum diurutkan. Nilai dari masing-masing elemen akan dibandingkan secara terus menerus sampai pengurutan selesai. Seringkali dikatakan metode ini kurang efisien namun sangat mudah dipahami. (Desisuryani (2013:3)).

Penyajian Psedoucode algoritma *Bubble sort* dapat dilihat dibawah ini :

```
for i=1 to (n-1) do
    for j=1 to (n-i) do
        if A[j] > A[j+1] then
        temp=A[j]
        A[j]=A[j+1]
       A[j+1]=temp
     end if
  end for
end for
```

Ciri khas dari *Bubble sort* adalah cepatkan data besar menempati posisi yang tepat dan lamanya data kecil berada pasa posisi yang tepat. Gerakan ini dianalogikan seperti Kura-kura dan Kelinci. Ketidak efektifan dari Algoritma *Bubble sort* maka munculah variasi atau pengembangan dari *Bubble sort* diantaranya *Cocktail sort* dan *Comb sort.*

2.5.1. *Cocktail Sort*

Cocktail sort merupakan pengembangan algoritma *Bubble sort* dengan Ide dasar data terkecil dan data terbesar berada di tempat yang tepat pada iterasi 1, kemudian proses akan melakukan perbandingan atau penukaran data tanpa mengganggu data awal dan akhir , setelah itu lakukan penukaran sampai data terurut.

Proses kompleksitas algoritma lebih baik dari *Bubble sort* karna tidak memakan proses yang panjang namun untuk kasus dengan kondisi *worst case* masih sama dengan *Bubble sort* dengan kompleksitas algoritma $O(n^2)$. *Pseudocode* dari algoritma *Cocktail sort* sebagai berikut :

```
Procedure cocktailSort( A :list of sortable items ) defined as:
  do
    swapped := false
    for eachi in0 tolength( A ) - 2
 do:
     ifA[ i ] > A[ i + 1 ]
     swap( A[ i ], A[ i + 1 ] ) //
     swapped := true
   end if
end for
      if swapped = false then
break do-while loop
end if
    swapped := false
    for eachi inlength( A ) - 2 to0
 do:
   ifA[ i ] > A[ i + 1 ] then
   swap( A[ i ], A[ i + 1 ] )
   swapped := true
 end if
 end for
end procedure
```

2.5.2. *Comb Sort*

Comb sort merupakan pengembangan dari *Bubble sort* dengan ide dasar penukaran data dilakukan tidak dengan sebelahnya tetapi berdasarkan gap. Gap didapat dari jumlah data dibagi dengan factor *shrink* yaitu 1.3. Jika gap nya sudah di dapat maka data akan ditukar berdasarkan dengan jumlah data. Komplesitas algoritma dari kondisi terburuk atau *worst case* pada algoritma ini adalah O(n log n). *Pseudocode* dari algoritma *Comb sort* sebagai berikut :

```
Function combsort(array input)  gap := input.size //initialize gap size loop untilgap <= 1 and swaps = 0
gap := int(gap / 1.25)
i := 0
swaps := 0
loop untili + gap >= input.size
       if input[i] > input[i+gap]
       swap(input[i],
      input[i+gap])
  swaps := 1
end if
  i := i + 1
```

end loop
end loop
end function

2.6. Bahasa Pemrograman C++

Menurut Joni dan Raharjo (2011:3) mengemukakan bahwa bahasa C merupakan bahasa yang powerful dan fleksibel yang telah terbukti dapat menyelesaikan program-program besar seperti pembuatan system operasi, pengolah kata, pengolahan gambar (seperti pembuatan game) dan juga pembuatan kompilator untuk bahasa pemrograman baru.

Bahasa C merupakan bahasa yang sudah populer dan banyak digunakan oleh programmer berpengalaman sehingga kemungkinan besar library (pustaka) dan aksesoris program lainnya yang diperlukan dalam pemrograman telah banyak disediakan oleh pihak luar/lain dan dapat diperoleh dengan mudah (Joni dan Raharjo, 2011:3).

Bahasa C/C++ merupakan bahasa yang sangat ketat dalam pemakaian type data maupun penulisanya yang case sensitive, untuk itu perlu perlu kedisiplinan dalam penulisan program.

3. PEMBAHASAN

Pada pembahasan ini akan ditampilkan hasil eksekusi program dari masing-masing algoritma, namun ada beberapa yang harus diketahui sebelum mengimplementasikanya ke dalam perintah program.

3.1. Perangkat

Pada pengujian ini menggunakan compiler Borland C++ pada platform windows 8, adapun spesifikasi komputer yang digunakan adalah :

a. Intel ® Core 2 Processor T6600 2.20 GHz
b. RAM 2 GB
c. HDD 320

Spesifikasi dari perangkat yang digunakan untuk mengeksekusi program ini sangat berpengaruh dengan kecepatan waktu pengurutan algoritma yang digunakan.

3.2. Prosedur Waktu Eksekusi

Menguji kompleksita waktu pada algoritam maka aka nada penambahan instruksi pada *header*.

```
#include <time.h>
#include <windows.h>
```

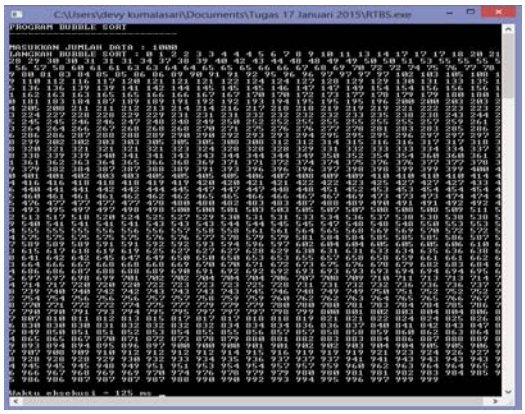
Kemudian tambahkan baris instruksi pencarian data secara acak.

```
srand(time(NULL));
{
 for(int x=0; x<jumlah; x++)
 jumlah+1;
}
double mulai = GetTickCount();
<<fungsi yang diuji>>
double selesai = GetTickCount();
```

3.3. Eksekusi Algoritma

3.3.1. *Bubble Sort*

Berdasarkan *pseduocode* algoritma *Bubble sort* yang ada maka dapat dibuatkan perintah program menggunakan Borland C++. Pada gambar.1 ditunjukkan hasil eksekusi algoritma Bubble Sort.



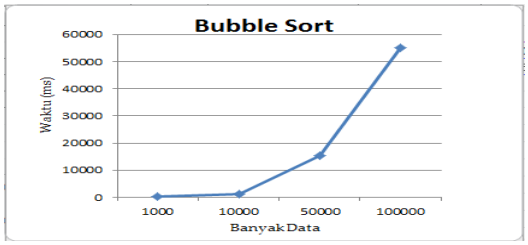
Gambar 1. Hasil Eksekusi Algoritma *Bubble sort*

Hasil eksekusi pada gambar diatas merupakan pengujian pengurutan dengan dengan algoritma *Bubble sort* data 1000. Selanjutnya akan diuji 10.000 data, 50.000 data dan 100.000 data. pada tabel dibawah adalah hasil eksekusi yang telah diuji dan pada grafik dapat dilihat siklus waktu eksekusi yang terpakai pada saat pengurutan data.

Tabel 1. Waktu Eksekusi *Bubble sort*

Banyak data	Waktu (ms)
1000	125
10000	1155
50000	15.273
100000	55.193

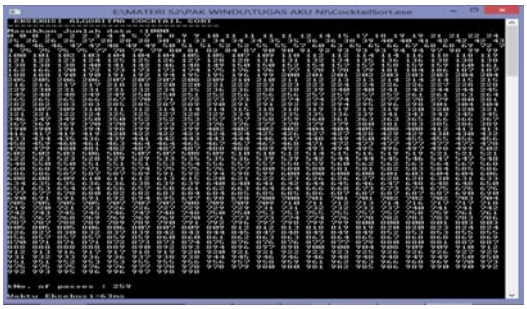
Sumber : Dokumentasi sendiri



Gambar 2. Grafik pengujian algoritma *Bubble sort*

3.3.2. *Cocktail Sort*

Berdasarkan *pseduocode* algoritma *Cocktail sort* yang ada maka dapat dibuatkan perintah program menggunakan Borland C++.



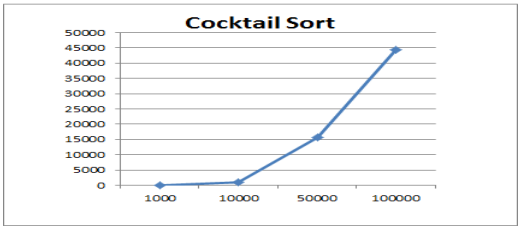
Gambar 3. Hasil eksekusi algoritma *Cocktail sort*

Hasil eksekusi pada gambar diatas merupakan pengujian pengurutan dengan dengan *cocktail sort* data 1000. Selanjutnya akan diuji 10.000 data, 50.000 data dan 100.000 data. Pada tabel dapat dilihat perbedaan waktu eksekusinya dan pada grafik dapat dilihat siklus waktu eksekusi yang terpakai pada saat pengurutan data.

Tabel 2. Waktu Eksekusi *Cocktail sort*

Banyak data	Waktu (ms)
1000	63
10000	1.014
50000	15.616
100000	44.429

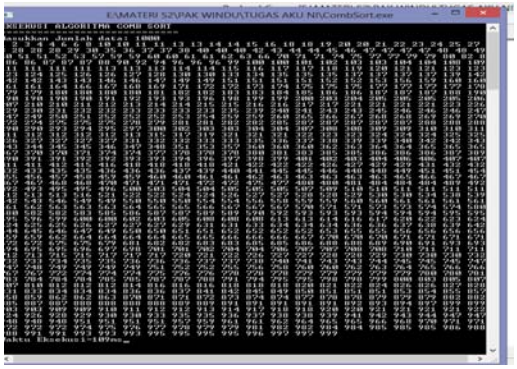
Sumber :Dokumentasi Sendiri



Gambar 4. Grafik pegujian algoritma *Cocktail sort*

3.3.3. *Comb Sort*

Berdasarkan *pseduocode* algoritma *Comb sort* yang ada maka dapat dibuatkan perintah program menggunakan Borland C++. Berikut adalah hasil eksekusi program pada C++ dengan 1000 data.



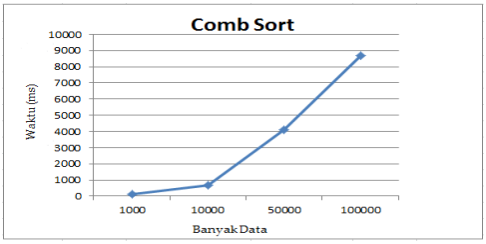
Gambar 5. Hasil Program algoritma *Comb sort*

Hasil eksekusi pada gambar diatas merupakan pengujian pengurutan dengan dengan algoritma *Comb sort* data 1000. Selanjutnya itu akan diuji 10.000 data, 50.000 data dan 100.000 data. Pada tabel dapat dilihat waktu eksekusinya dan pada grafik dapat dilihat siklus waktu eksekusi yang terpakai pada saat pengurutan data.

Tabel 3. Waktu Eksekusi *Comb sort*

Banyak data	Waktu (ms)
1000	109
10000	671
50000	4.087
100000	8.705

Sumber :Dokumentasi sendiri



Gambar 6. Grafik pengujian data algoritma *Comb sort*

3.4. Perbandingan Kompleksitas Waktu algoritma

Setelah dilakukanya eksekusi program menggunakan C++ *Bubble sort, Cocktail sort* dan *Comb sort,* maka langkah selanjutnya adalah membandingkan kompleksitas waktu ketiga algoritma yang ada.

Tabel 4. Perbandingan Kompleksitas Waktu Algoritma

Range Data	Bubble Sort	Cocktail Sort	Comb Sort
1000	125	63	109
10000	1.155	1.014	671

50000	15.273	15.616	4.087
100000	55.193	44.429	8.705

Sumber :Dokumentasi Sendiri

Pada tabel diatas dapat dilihat perbandingan waktu setelah dialakukan kompilasi sehingga menghasilkan output untuk setiap algoritmanya.

a. *Bubble Sort*

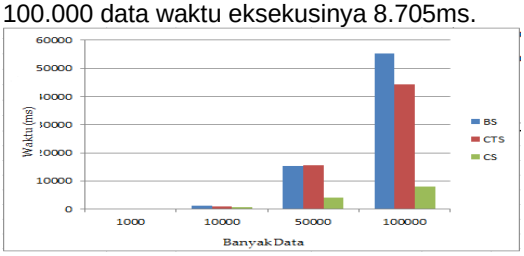
Dari data 1000 waktu eksekusinya adalah 125ms, 10000 data waktu eksekusinya 1.155ms, 50.000 data waktu eksekusinya 15.273ms dan 100.000 data waktu eksekusinya 55.193ms.

b. *Cocktail Sort*

Mulai 1000 data waktu eksekusinya adalah 63ms, 10000 data waktu eksekusinya 1.014ms, 50.000 data waktu eksekusinya 15.616ms dan 100.000 data waktu eksekusinya 44.429ms.

c. *Comb Sort*

Mulai 1000 data waktu eksekusinya adalah 109ms, 10000 data waktu eksekusinya 671ms, 50.000 data waktu eksekusinya 4.087ms dan 100.000 data waktu eksekusinya 8.705ms.



Gambar 7. Diagram Batang Perbandingan Waktu Eksekusi

4.1 Kesimpulan

Berdasarkan logika proses pengurutan data menggunakan algoritma *Bubble sort, Cocktail sort* dan *Comb sort* dengan bahasa pemrograman C++ mulai dari pembuatan program dan dilanjutkan dengan proses kompilasi serta melihat perbandingan waktu yang terpakai pada saat pengurutan data, maka dapat disimpulkan :

1. Pengujian program diketahui bahwa algoritma *Cocktail sort* lebih cepat pengurutannya dibandingkan *Bubble sort* dan *Comb sort* untuk data 1000 sedangkan untuk data > 10.000 *Comb sort* jauh lebih cepat dalam pengurutan data.
2. Meskipun dalam pengurutan *Bubble sort* memiliki waktu yang cukup lama dibanding *Cocktail sort* dan *Comb sort* tapi secara teknis *Bubble sort* lebih mudah dipahami instruksi algoritmanya.

4.2 Saran

Setiap algoritma memiliki kelebihan dan kekurangan dalam menyelesaikan proses pengurutan data. Pada penelitian ini penulis memberikan saran yaitu untuk mengurutkan data dengan jumlah data yang banyak sebaiknya jangan gunakan algoritma *Bubble Sort* karena proses pengurutanya sangat lama dan tidak efektif.

DAFTAR PUSTAKA

[1] Joni, I Made dan Budi Raharjo. 2011.
[2] Pemrograman C dan Implementasinya.
[3] Bandung: Informatika
[4] Ryan.2010.http://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2009-2010/Makalah0910/MakalahStrukdis0910-032.pdf
[5] Saniman, Fathoni. 2010. Konsep *Sorting* dalam Pemrograman.Jurnal SAINTIKOM Vol. VIII. 1 Januari 2010
[6] Suarga. 2012. Algoritma dan Pemrograman. Yokyakarta: Andi.
[7] Suryani. 2013. Perbandingan Metode *Bubble Sort* dan *Insertion Sort* terhadap Efisiensi Memori. Jurnal Teknologi Informasi & Pendidikan, Vol. 6 No. 1 Maret 2013.
[8] Utami, Sukrisno .2005. 10 Langkah Belajar Logika dan Algoritma, Menggunakan Bahasa C dan C++ di GNU/LINUX. Yokyakarta. Andi
[9] Yahya, Sofyansyah Yusari. 2014. Analisa Perbandingan Algoritma *Bubble Sort* dan *Selection Sort* Dengan Metode
[10] Perbandingan Eksponensial. Jurnal Pelita Informatika Budi Darma, Vol : VI, No : 3, April 2014 (http://pelita-informatika.com/berkas/jurnal/28.%20Sofyansyah.pdf, diakses 18 Januari 2016).