

Algoritma Pengurutan Data (*Sorting*) Dengan Metode *Insertion Sort* dan *Selection Sort*

Endang Retnoningsih^{1,*}

¹ Sistem Informasi; STMIK Bina Insani; Jl. Siliwangi No.6 Rawa Panjang Bekasi Bekasi Timur 17114 Indonesia, Telp. (021) 824 36 886 / (021) 824 36 996. Fax. (021) 824 009 24; e-mail: endang.retnoningsih@binainsani.ac.id

* Korespondensi: e-mail: endang.retnoningsih@binainsani.ac.id

Diterima: 17 November 2018; Review: 21 November 2018; Disetujui: 29 November 2018

Cara sitasi: Retnoningsih E. 2018. Algoritma Pengurutan Data (*Sorting*) Dengan Metode *Insertion Sort* dan *Selection Sort*. Information Management For Educators And Professionals. 3 (1): 95 - 106.

Abstrak: Pengurutan merupakan hal yang tidak bisa dipisahkan dari dunia komputer. Adanya kebutuhan terhadap proses pengurutan memunculkan bermacam-macam metode pengurutan yang bertujuan untuk memperoleh metode pengurutan yang optimal. Dengan menggunakan algoritma yang baik dapat menghasilkan program yang efisien dari segi waktu dan hasil yang dicapai. Data terkadang berada dalam bentuk yang tidak berpola ataupun dengan pola tertentu yang diinginkan. Secara umum ada dua jenis pengurutan data yaitu modelurut naik (ascending) yang mengurutkan data dari yang mempunyai nilai terkecil sampai terbesar dan modelurut turun (descending) yang mengurutkan data dari yang mempunyai nilai terbesar sampai terkecil. Perbandingan pengurutan data (sorting) dalam sebuah array (L) menggunakan dua algoritma dengan prinsip kerja yang berbeda. Pada penelitian ini membandingkan algoritma metode selection sort menggunakan prinsip pertukaran elemen dalam proses sorting, dan metode insertion sort menggunakan prinsip geser dan sisip elemen dalam proses sorting. Tidak ada algoritma terbaik untuk semua keadaan, kadang kala sebuah algoritma sangat efisien ketika jumlah datanya sedikit, namun kinerjanya menjadi berkurang ketika jumlah data ditambahkan atau meningkat. Hasil dari penelitian adalah metode *insertion sort* lebih unggul pada jumlah data yang sedikit, sedangkan metode *selection sort* lebih unggul pada jumlah data yang lebih banyak.

Kata kunci: Algoritma, Larik, Insertion Sort, Selection Sort, Visual Basic.

Abstract: Ordering is something that cannot be separated from the computer world. The need for the sorting process gives rise to a variety of sorting methods that aim to obtain the optimal sorting method. Using a good algorithm can produce programs that are efficient in terms of time and results achieved. Data sometimes is in a form that is not patterned or with a certain pattern desired. In general there are two types of sorting data, namely the ascending model which sorts data from the smallest to the largest and the descending sequence model which sorts data from the largest to the smallest. Comparison of sorting in an array (L) uses two algorithms with different working principles. In this study comparing algorithms the selection sort method uses the principle of element exchange in the sorting process, and the insertion sort method uses the principle of sliding and inserting elements in the sorting process. There is no best algorithm for all situations, sometimes an algorithm is very efficient when the amount of data is small, but its performance decreases when the amount of data is added or increased. The results of the study are that the insertion sort method is superior to the small amount of data, while the selection sort method is superior to a larger amount of data.

Keywords: Algorithms, Array, Insertion Sort, Selection Sort, Visual Basic.

1. Pendahuluan

Algoritma merupakan bagian terpenting dan tidak dapat dipisahkan dari pemrograman. Sebelum membuat suatu program aplikasi, hal pertama yang perlu dipahami adalah algoritma atau prosedur pemecahan masalah. Beberapa hal yang perlu diperhatikan dalam merancang algoritma antara lain 1) Algoritma berisi deskripsi langkah-langkah penyelesaian masalah, 2) Setiap algoritma ditulis dalam bentuk *pseudocode*, 3) *Pseudocode* yang dibuat mempunyai kemiripan dengan bahasa pemrograman umum, 4) *Pseudocode* dalam bentuk notasi algoritmik harus diterjemahkan ke dalam notasi bahasa pemrograman yang dipilih, 4) Algoritma yang dibuat diterjemahkan kedalam notasi bahasa pemrograman seperti pendeklarasian variabel, pemilihan tipe data, aturan sintaks, dan lainnya [Ramadhani, 2015].

Algoritma pengurutan (*sorting*) meletakkan elemen data kedalam kumpulan data urutan tertentu, proses pengurutan yang sebelumnya data disusun acak menjadi tersusun teratur menurut aturan tertentu [Saputro and Khasanah, 2018]. Pengurutan merupakan hal yang tidak bisa dipisahkan dari dunia komputer. Sekarang ini *Google* dikenal sebagai mesin pencari terbesar di dunia. Dalam hitungan detik dapat diperoleh informasi yang diinginkan. Adanya kebutuhan terhadap proses pengurutan memunculkan bermacam-macam metode pengurutan yang bertujuan untuk memperoleh metode pengurutan yang optimal [Sitorus and Sembiring, 2012].

Pengurutan data (*sorting*) didefinisikan sebagai proses untuk menyusun kembali himpunan objek dengan menggunakan aturan tertentu. Tujuannya adalah untuk mendapatkan kemudahan dalam pencarian dari suatu himpunan. Kelebihan suatu data yang terurut adalah mudah untuk dicek apabila ada data yang hilang.

Larik (*Array*)

Array adalah kumpulan data untuk menyimpan item bertipe data sama, biasanya pada pengurutan data adalah data dengan tipe sama. Setiap data disimpan dalam alamat memori yang berbeda yang disebut elemen *array*. Komponen-komponen dari *array* antara lain nama *array*, nilai *array*, indeks *array*, jenis *array* [Situmorang, 2016]. Adapun proses yang dapat dilakukan dalam sebuah *array* antara lain memasukkan elemen data, mencari elemen data, menghapus elemen data, mencetak atau menampilkan hasil dari proses pengolahan data, menyisipkan elemen data, mencari posisi elemen data tertentu dan proses mengurutkan elemen data [Situmorang, 2016].

Pengurutan Data (*Sorting*)

Data terkadang berada dalam bentuk yang tidak berpola ataupun dengan pola tertentu yang diinginkan. Tidak ada algoritma terbaik untuk semua keadaan, kadang kala sebuah algoritma sangat efisien ketika jumlah datanya sedikit, namun kinerjanya menjadi berkurang ketika jumlah data ditambahkan atau meningkat. Meskipun memiliki kemampuan komputasi yang lebih tinggi, namun jika menggunakan algoritma yang kurang efisien, maka akan membutuhkan waktu lebih lama. Sehingga untuk memecahkan permasalahan diperlukan sebuah algoritma yang efektif dan efisien agar persoalan komputasi serta terbatasnya alokasi memori dapat diatasi.

Secara umum ada dua jenis pengurutan data yaitu [Ramadhani, 2015]: 1) Model urut naik (*ascending*) yang mengurutkan data dari yang mempunyai nilai terkecil sampai terbesar. 2) Model urut turun (*descending*) yang mengurutkan data dari yang mempunyai nilai terbesar sampai terkecil. Jika N buah data disimpan didalam sebuah *array* Nilai, maka pengurutan *ascending* berarti menyusun elemen *array* sedemikian hingga:

$$\text{NILAI}[0] \leq \text{NILAI}[1] \leq \text{NILAI}[2] \leq \dots \leq \text{NILAI}[N-1] \quad (1)$$

Sumber: Sitorus and Sembiring (2012)

Sedangkan pengurutan *descending* berarti menyusun elemen *array* sedemikian hingga:

$$\text{NILAI}[0] \geq \text{NILAI}[1] \geq \text{NILAI}[2] \geq \dots \geq \text{NILAI}[N-1] \quad (2)$$

Sumber: Sitorus and Sembiring (2012)

Algoritma pengurutan data yang sering ditemukan dalam literatur komputer antara lain *bubble sort*, *selection sort*, *insertion sort*, *heap sort*, *shell sort*, *quick sort*, *merge sort*, *radix sort*,

dan *tree sort* [Munir, 2011]. Semua algoritma pengurutan selalu melakukan operasi perbandingan data untuk menemukan posisi urutan yang tepat. Berdasarkan tempat penyimpanan data, *sorting* dibedakan antara *external sort* dan *internal sort* [Sjukani, 2012]. *External sort* bila datanya berada dalam media *external*, atau *external storage* seperti *hardisk*. *Internal sort* bila datanya ada dalam *internal storage* atau *memory* komputer.

Algoritma Insertion Sort

Algoritma *insertion sort*, adalah metode pengurutan dengan cara menyisipkan elemen data pada posisi yang tepat. Pencarian posisi yang tepat dilakukan dengan melakukan pencarian berurutan didalam barisan elemen, selama pencarian posisi yang tepat dilakukan pergeseran elemen [Sitorus and Sembiring, 2012]. Pengurutan *insertion sort* sangat mirip dengan konsep permainan kartu, bahwa setiap kartu disisipkan secara berurutan dari kiri ke kanan sesuai dengan besar nilai kartu tersebut, dengan syarat apabila sebuah kartu disisipkan pada posisi tertentu kartu yang lain akan bergeser maju atau mundur sesuai dengan besaran nilai yang dimiliki [Ramadhani, 2015]. Proses pengurutan data dalam sebuah *array* menggunakan algoritma *insertion sort* tersebut dapat dilihat pada gambar 1, berikut:



Sumber: Ramadhani (2015).

Gambar 1. Insertion Sort

Pada gambar 1, terlihat pergeseran *array* dilakukan dari $i=1$ yang kemudian dibandingkan dengan *array* yang berada disebelah kiri. Apabila *array* kedua lebih kecil dari *array* pertama, akan dilakukan penukaran. Selanjutnya dengan *looping* secara perlahan indeks dari *array*

sebelah kanan dibandingkan lagi dengan *array* sebelah kiri yang sudah tersusun rapi sehingga apabila ditemukan nilai sebelah kanan lebih kecil dari nilai sebelah kiri yang sudah tersusun rapi, pergeseran dan penukaran akan dilakukan.

Algoritma *Selection Sort*

Algoritma *selection sort* sering juga disebut dengan metode maksimum atau minimum. Metode maksimum karena didasarkan pada pemilihan data atau elemen maksimum sebagai dasar pengurutan. Konsepnya dengan memilih elemen maksimum kemudian mempertukarkan elemen maksimum tersebut dengan elemen paling akhir untuk urutan *ascending* dan elemen pertama untuk *descending*.

Algoritma *selection sort* disebut juga dengan metode minimum karena didasarkan pada pemilihan elemen minimum sebagai dasar pengurutan. Konsepnya dengan memilih elemen minimum kemudian mempertukarkan elemen minimum dengan elemen paling akhir untuk urutan *ascending* dan elemen pertama untuk urutan *descending*. Proses yang dilakukan oleh algoritma *selection sort* adalah mengambil nilai terbesar dari susunan data dan menggantikannya dengan data yang paling kanan [Ramadhani, 2015].

Proses pengurutan data dalam sebuah *array* menggunakan algoritma *selection sort* tersebut dapat dilihat pada gambar 2, berikut:



Sumber: Ramadhani (2015)

Gambar 2. Metode *Selection Sort* (*Ascending*)

Pada gambar 2, terlihat proses yang dilakukan oleh algoritma *selection sort* adalah mengambil nilai terbesar dari susunan *array* dan menggantikannya dengan *array* yang paling kanan. Algoritma *selection sort* selalu menganggap nilai pada index pertama sebagai nilai maksimum ($max=0$), kemudian membandingkan nilai dari indeks pertama tersebut ($array[max]$) dengan nilai-nilai pada indeks berikutnya. Apabila nilai dari indeks yang dibandingkan lebih tinggi dari nilai indeks pertama, nilai tersebut diambil sebagai nilai max ($max=i$). Perbandingan tersebut dilakukan sebanyak ukuran *array* atau sejumlah elemen data yang akan diurutkan.

Visual Basic .Net merupakan *core* dari pembuatan aplikasi berbasis .Net, model untuk *development* dimana *platform* dan aplikasi bisa dibuat dan dijalankan tanpa bergantung pada alat (*device*) yang dipakai [Bernardo et al., 2015]. Pada penelitian ini Visual Basic .Net digunakan sebagai antar muka dalam perbandingan algoritma *insertion sort* dan *selection sort*.

2. Metode Penelitian

Perbandingan pengurutan data (*sorting*) dalam sebuah *array* (L) menggunakan dua algoritma dengan prinsip kerja yang berbeda. Metode *selection sort* menggunakan prinsip pertukaran elemen dalam proses *sorting*, sedangkan metode *insertion sort* menggunakan prinsip geser dalam proses *sorting* [Munir, 2011].

Metode *insertion sort*, secara *ascending* urutan langkah secara garis besar untuk setiap $pass\ i=2, \dots, n$ yaitu $y \leftarrow L[i]$, sisipkan y pada tempat yang sesuai diantara $L[1] \dots L[i]$. Selanjutnya langkah rincian pada setiap *Pass* yaitu a) **Pass2**, Elemen $y = L[2]$ harus cari tempat yang tepat di dalam $L[1..2]$ dengan cara menggeser elemen $L[1..1]$ ke kanan (atau ke bawah) bila $L[1..1]$ lebih besar daripada $L[2]$. b) **Pass3**, Elemen $y = L[3]$ harus cari tempat yang tepat di dalam $L[1..3]$ dengan cara menggeser elemen $L[1..2]$ ke kanan (atau ke bawah) bila $L[1..2]$ lebih besar daripada $L[3]$. c) Seterusnya sampai dengan **Pass n** , Elemen $y = L[n]$ harus cari tempat yang tepat di dalam $L[1..n]$ dengan cara menggeser elemen $L[1..n-1]$ ke kanan (atau ke bawah) bila $L[1..n-1]$ lebih besar daripada $L[n]$.

Metode *selection sort*, secara *ascending* urutan langkah secara garis besar yaitu $JumlahPass=n-1$, untuk setiap $Pass\ i=1, 2, \dots, JumlahPass$ lakukan pencarian elemen, pertukaran max dengan elemen $ke-n$, mengurangi nilai n satu. Selanjutnya langkah rincian pada setiap *Pass* yaitu a) **Pass1**, cari elemen maksimum di dalam $L[1..n]$, pertukarkan elemen maksimum dengan elemen $L[n]$, ukuran larik yang belum terurut = $n-1$. b) **Pass2**, cari elemen maksimum di dalam $L[1..n-1]$, pertukarkan elemen maksimum dengan elemen $L[n-1]$, ukuran larik yang belum terurut = $n-2$. c) **Pass3**, cari elemen maksimum di dalam $L[1..n-2]$, pertukarkan elemen maksimum dengan elemen $L[n-2]$, ukuran larik yang belum terurut = $n-3$. d) Seterusnya sampai dengan **Pass $n-1$** , cari elemen maksimum di dalam $L[1..2]$, pertukarkan elemen maksimum dengan elemen $L[2]$, ukuran larik yang belum terurut = 1. e) Setelah *Pass $n-1$* , elemen yang tersisa adalah $L[1]$, tidak perlu diurutkan lagi karena hanya satu-satunya.

3. Hasil dan Pembahasan

Perbandingan pengurutan data (*sorting*) dalam penelitian dilakukan dengan menentukan jangkauan (*range*) data yang akan diuji adalah sejumlah sepuluh elemen data, lima puluh elemen data dan lima ratus elemen data.

3.1. Pseudocode Sorting

Pseudocode mempunyai hubungan dengan penulisan bahasa pemrograman sehingga proses penerjemahan lebih mudah ke kode program yang digunakan. Tidak ada aturan baku dalam membuat *pseudocode*. Pada setiap satu kali langkah n disebut juga dengan satu kali *pass n* , pengurutan terdapat proses mencari elemen data maksimum (*ascending*) atau elemen data minimum (*descending*) dan proses pertukaran dua buah elemen data dalam *array*.

Pseudocode Insertion Sort

Algoritma *insertion sort* proses mencari elemen data maksimum (*ascending*) selengkapnya seperti pada gambar 3 berikut:

```

Procedure InsertionSort1 (input/output L : LarikInt, input n:integer)
L[1] ≤ L[2] ≤ ... ≤ L[n]

DEKLARASI
i,j,y : integer

ALGORITMA
for i ← 2 to n do
  y ← L[i]
  j ← i-1
  while (j ≥ 1) and (not ketemu) do
    if y < L[j] then
      L[j+1] ← L[j]
      j ← j-1
    else
      ketemu ← true
    endif
  endwhile
  {j<1 or ketemu}
  L[j+1] ← y
endfor

```

Sumber: Hasil Penelitian (2018)

Gambar 3 Pseudocode Insertion Sort (Ascending)

Pseudocode Selection Sort

Algoritma *selection sort* proses mencari elemen data maksimum (*ascending*) selengkapnya seperti pada gambar 4 berikut:

```

Procedure SelectionSort1 (input/output L : LarikInt, input n:integer)
L[1] ≤ L[2] ≤ ... ≤ L[n]

DEKLARASI
i,j,imaks,maks,temp : integer

ALGORITMA
for i ← n downto 2 do
  imaks ← 1
  maks ← L[1]

  for j ← 2 to i do
    if L[j] > L[maks] then
      imaks ← j
      maks ← L[j]
    endif
  endfor

  temp ← L[i]
  L[i] ← L[maks]
  L[imaks] ← temp
endfor

```

Sumber: Hasil Penelitian (2018)

Gambar 4. Pseudocode Selection Sort (Ascending)

3.2 Pengurutan Data (Sorting) Menggunakan Visual Basic .Net

Visual Basic .Net merupakan pemrograman berorientasi objek (*OOP*) yang dapat digunakan untuk merancang antar muka dalam perbandingan algoritma. *OOP* bermaksud untuk memecahkan masalah pemrograman dengan pola pikir manusia. Program yang dibuat didesain dengan membuat prosedur-prosedur yang sederhana. Pada *OOP* terdapat objek yang memiliki informasi dari sebuah atribut data yang berada dalam sebuah *class*. Berikut adalah *class – class* dalam pengurutan data.


```

Public Class insertion
Dim stopwatch As New Diagnostics.Stopwatch
Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button2.Click
Me.Close()
End Sub

Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
Me.stopwatch.reset()
If data.Text = "" Then
MsgBox("Tolong Input Jumlah Data Terlebih Dahulu")
data.Focus()
Else
Me.stopwatch.Start()
hasil.Text = "Data : "
hasil2.Text = "Data : "
Dim input As Integer
input = (data.Text)
Dim inputLength As Integer
inputLength = Int(input)
Dim arr(inputLength) As Integer
Randomize()
If data.Text >= 100 Then
For index As Integer = 0 To inputLength - 1
arr(index) = CInt(Int((999 * Rnd()) + 1))
Next
ElseIf data.Text >= 1000 Then
For index As Integer = 0 To inputLength - 1
arr(index) = CInt(Int((9999 * Rnd()) + 1))
Next
ElseIf data.Text >= 10000 Then
For index As Integer = 0 To inputLength - 1
arr(index) = CInt(Int((99999 * Rnd()) + 1))
Next
ElseIf data.Text >= 100000 Then
For index As Integer = 0 To inputLength - 1
arr(index) = CInt(Int((999999 * Rnd()) + 1))
Next
ElseIf data.Text >= 1000000 Then
For index As Integer = 0 To inputLength - 1
arr(index) = CInt(Int((1300000 * Rnd()) + 1))
Next

```

Sumber: Hasil Penelitian (2018)

Gambar 5 (a) . Algoritma *Insertion Sort* Menggunakan Visual Basic .Net

Gambar 5 (a) dan gambar 5 (b) merupakan kesatuan penulisan algoritma metode *insertion sort* kedalam *class* pada menggunakan bahasa pemrograman Visual Basic .Net.

```

ElseIf data.Text < 100 Then
For index As Integer = 0 To inputLength - 1
arr(index) = CInt(Int((99 * Rnd()) + 1))
Next
Else
For index As Integer = 0 To inputLength - 1
arr(index) = CInt(Int((99999 * Rnd()) + 1))
Next
End If
If hasil.Text.Trim <> Nothing Then
For index2 As Integer = 0 To inputLength - 1
hasil.Text &= (arr(index2) & " ")
Next
Dim i, j As Integer
For i = 1 To inputLength - 1 Step 1
Dim pick_item As Integer = arr(i)
Dim inserted As Integer = 0
j = i - 1
While (j >= 0 And inserted <> 1)
If (pick_item < arr(j)) Then
arr(j + 1) = arr(j)
j -= 1
arr(j + 1) = pick_item
Else : inserted = 1
End If
End While
Next
For i = 0 To inputLength - 1
hasil2.Text &= (arr(i) & " ")
Next
Me.stopwatch.Stop()
End If
data.Clear()
data.Focus()
End If
waktu.Text = Me.stopwatch.Elapsed.Seconds.ToString("00") & "," &
Me.stopwatch.Elapsed.Milliseconds.ToString("000")
End Sub
End Class

```

Sumber: Hasil Penelitian (2018)

Gambar 5 (b) . Algoritma *Insertion Sort* Menggunakan Visual Basic .Net

```

Public Class selection
Dim stopwatch As New Diagnostics.Stopwatch
Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button2.Click
Me.Close()
End Sub

Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
Me.stopwatch.Reset()
If data.Text = "" Then
MsgBox("Tolong Input Jumlah Data Terlebih Dahulu")
data.Focus()
Else
Me.stopwatch.Start()
hasil.Text = "Data : "
hasil2.Text = "Data : "
Dim t, j As Integer
Dim input As Integer
input = (data.Text)
Dim inputLength As Integer
inputLength = Int(input)
Dim arr(inputLength) As Integer Randomize()
If data.Text >= 100 Then
For index As Integer = 0 To inputLength - 1
arr(index) = CInt(Int((999 * Rnd()) + 1))
Next
ElseIf data.Text >= 1000 Then
For index As Integer = 0 To inputLength - 1
arr(index) = CInt(Int((9999 * Rnd()) + 1))
Next
ElseIf data.Text >= 10000 Then
For index As Integer = 0 To inputLength - 1
arr(index) = CInt(Int((99999 * Rnd()) + 1))
Next
ElseIf data.Text >= 100000 Then
For index As Integer = 0 To inputLength - 1
arr(index) = CInt(Int((999999 * Rnd()) + 1))
Next
End If

```

Sumber: Hasil Penelitian (2018)

Gambar 6 (a). Algoritma *Selection Sort* Menggunakan *Visual Basic .Net*

Gambar 6 (a) dan gambar 6 (b) merupakan kesatuan penulisan algoritma metode *selection sort* kedalam *class* pada menggunakan bahasa pemrograman *Visual Basic .Net*.

```

ElseIf data.Text >= 1000000 Then
For index As Integer = 0 To inputLength - 1
arr(index) = CInt(Int((1300000 * Rnd()) + 1))
Next
ElseIf data.Text < 100 Then
For index As Integer = 0 To inputLength - 1
arr(index) = CInt(Int((99 * Rnd()) + 1))
Next
Else
For index As Integer = 0 To inputLength - 1
arr(index) = CInt(Int((99999 * Rnd()) + 1))
Next
If hasil.Text.Trim <> Nothing Then
For index2 As Integer = 0 To inputLength - 1
hasil.Text &= (arr(index2) & " ")
Next
For i = 0 To inputLength - 2
For j = i + 1 To inputLength - 1
If arr(i) > arr(j) Then
t = arr(i)
arr(i) = arr(j)
arr(j) = t
End If
Next
Next
For i = 0 To inputLength - 1
hasil2.Text &= (arr(i) & " ")
Next
End If
data.Clear()
data.Focus()
Me.stopwatch.Stop()
End If
waktu.Text = Me.stopwatch.Elapsed.Seconds.ToString("00") & "," &
Me.stopwatch.Elapsed.Milliseconds.ToString("000")
End Sub
End Class

```

Sumber: Hasil Penelitian (2018)

Gambar 6 (b). Algoritma *Selection Sort* Menggunakan *Visual Basic .Net*

3.3. Hasil Uji Coba Elemen Data Acak

Pada penelitian ini dilakukan pengujian terhadap *algoritma insertion sort* dan *selection sort* dengan memasukkan data yang akan *disorting* ke dalam *text box* sejumlah elemen data. Misalnya, sejumlah 10 elemen data, kemudian klik mulai maka akan ditampilkan data acak pada *text box* hasil data yang belum di sorting data pada *text box* hasil data yang sudah di sorting. Pada Pengujian juga ditampilkan pada *text box* waktu yang diperlukan oleh algoritma dalam mengurutkan elemen data (proses *sorting*).

Gambar 7 berikut menunjukkan pengujian menggunakan algoritma *insertion sort*, elemen yang diuji sejumlah sepuluh data secara acak, menghasilkan waktu 00.003 detik sehingga keseluruhan elemen data menjadi terurut secara menaik (*ascending*).

Insertion Sorting

Masukan data yang akan di sorting

Mulai

Tutup

Hasil data yang belum di sorting

Data : 96 65 51 65 74 87 50 97 21 43

Hasil data yang sudah di sorting

Data : 21 43 50 51 65 65 74 87 96 97

Lama Proses (Per-Detik) 00.003

Sumber: Hasil Penelitian (2018)

Gambar 7. Uji Coba Sepuluh Data Menggunakan *Insertion Sort*

Gambar 8 berikut menunjukkan pengujian menggunakan algoritma *insertion sort*, elemen yang diuji sejumlah sepuluh data secara acak, menghasilkan waktu 00.013 detik sehingga keseluruhan elemen data menjadi terurut secara menaik (*ascending*).

Insertion Sorting

Masukan data yang akan di sorting

Mulai

Tutup

Hasil data yang belum di sorting

Data : 63 63 27 44 85 25 81 83 79 33 20 13 90 27 73 62 33 52 68 89 41 82 40 16 58 69 84 89 41 27 21 80 92 25 93 87 22 19 90 79 24 5 86 61 90 87 16 64 40 91

Hasil data yang sudah di sorting

Data : 5 13 16 16 19 20 21 22 24 25 25 27 27 33 33 40 40 41 41 44 52 58 61 62 63 63 64 68 69 73 79 80 81 82 83 84 85 86 87 87 89 89 90 90 90 91 92 93

Lama Proses (Per-Detik) 00.013

Sumber: Hasil Penelitian (2018)

Gambar 8. Uji Coba Lima Puluh Data Menggunakan *Insertion Sort*

Gambar 9 berikut menunjukkan pengujian menggunakan algoritma *insertion sort*, elemen yang diuji sejumlah sepuluh data secara acak, menghasilkan waktu 01.495 detik sehingga keseluruhan elemen data menjadi terurut secara menaik (*ascending*).

Insertion Sorting

Masukan data yang akan di sorting

Mulai Tutup

Hasil data yang belum di sorting

Data : 809 6 813 638 111 787 657 786 872 132 989 944 189 64 798 955 88 814 449 538 979 467 761 48 362 839 337 492 771 634 136 798 988 820 310 329 492 280 275 646 156 698 44 526 210 699 798 230 438 244 726 14 634 264 92 896 304 555 565 675 616 921 805 192 610 559 282 846 646 702 356 366 107 509 611 817 355 210 426 873 932 489 806 330 16 130 463 870 117 907 137 82 40 724 180 718 80 5 384 220 480 334 57 478 133 348 349 847 532 875 836 416 978 331 346 519 30 348 32 503 207 677 705 743 947 324 416 909 803 938 274 481 900 457 532 512 640 993 913 648 647 978 188 390 980 550 1 611 583 197 953 323 981 644 929 691 246 999 670 298 187 140 608 662 813 353 940

Hasil data yang sudah di sorting

Data : 1 2 5 6 7 10 14 14 16 16 22 30 31 32 32 33 34 35 35 40 42 43 44 48 54 57 64 66 73 76 78 80 82 88 88 91 92 94 99 101 102 105 105 105 105 107 107 109 111 112 114 116 117 118 118 126 126 127 128 129 129 130 132 133 136 137 140 143 144 147 147 154 156 159 171 176 177 180 182 182 184 187 188 189 192 193 197 198 200 202 207 207 209 210 210 216 219 220 221 228 229 230 230 230 231 231 244 246 250 251 258 261 264 265 266 267 272 274 275 275 280 280 282 282 282 289 289 292 293 298 301 303 303 304 306 306 308 310 315 315 315 317 318 319 321 323 324 326 328 329 330 330 331 331 333 334 335 337 341 346 347 348 348 349 350 353 353 355 356 362 364 364 366 366 369 370 372 380 381 383 384 385 386 387 389 390 391 392 402 405 405 406 410 411 414 416 416 424 426 427 427 432 434 436 438 440 441 442 442 447 449 451 453 456 457 460 463 463 464 466 467 468 470 474 475 476 477 478 480 481 483 484 485 489 489 489 490 492 492 495 497 501 503 503 504 505 506 508 509 512 513 515 516 519 521 525 525 526 527 532 532 538 540 540 541 541 545 549 550 550

Lama Proses (Per-Detik) 01,495

Sumber: Hasil Penelitian (2018)

Gambar 9. Uji Coba Lima Ratus Data Menggunakan *Insertion Sort*

Gambar10 berikut menunjukkan pengujian menggunakan algoritma *selection sort*, elemen yang diuji sejumlah sepuluh data secara acak, menghasilkan waktu 00.018 detik sehingga keseluruhan elemen data menjadi terurut secara menaik (*ascending*).

Selection Sorting

Masukan data yang akan di sorting

Mulai Tutup

Hasil data yang belum di sorting

Data : 45 31 27 23 50 19 78 46 71 2

Hasil data yang sudah di sorting

Data : 2 19 23 27 31 45 46 50 71 78

Lama Proses (Per-Detik) 00,018

Sumber: Hasil Penelitian (2018)

Gambar 10. Uji Coba Sepuluh Data Menggunakan *Selection Sort*

Gambar 11 berikut menunjukkan pengujian menggunakan algoritma *selection sort*, elemen yang diuji sejumlah lima puluh data secara acak, menghasilkan waktu 00.014 detik sehingga keseluruhan elemen data menjadi terurut secara menaik (*ascending*).

Selection Sorting

Masukan data yang akan di sorting

Mulai

Tutup

Hasil data yang belum di sorting

Data : 69 10 59 43 60 97 37 13 73 22 5 13 81 90 43 86 16 41 46 33 37 30 55 9 25 42 16 59 4 37 61 65 87 5 20 22 48 64 49 38 10 68 91 67 73 49 60 55 58 79

Hasil data yang sudah di sorting

Data : 4 5 5 9 10 10 13 13 16 16 20 22 22 25 30 33 37 37 37 38 41 42 43 43 46 48 49 49 55 55 58 59 59 60 60 61 64 65 67 68 69 73 73 79 81 86 87 90 91 97

Lama Proses (Per-Detik) 00,014

Sumber: Hasil Penelitian (2018)

Gambar 11. Uji Coba Lima Puluh Data Menggunakan *Selection Sort*

Gambar 12 berikut menunjukkan pengujian menggunakan algoritma *selection sort*, elemen yang diuji sejumlah lima ratus data secara acak, menghasilkan waktu 01.483 detik sehingga keseluruhan elemen data menjadi terurut secara menaik (*ascending*).

Selection Sorting

Masukan data yang akan di sorting

Mulai

Tutup

Hasil data yang belum di sorting

Data : 870 924 372 139 697 883 430 416 772 195 136 624 415 714 488 85 726 949 525 998 168 328 526 834 391 712 468 976 311 164 727 607 440 960 34 853 463 861 670 173 65 623 930 999 11 765 820 629 470 585 11 364 603 424 38 902 378 183 762 976 657 87 70 247 524 704 819 442 856 208 56 745 161 892 452 444 647 943 25 809 557 445 74 245 743 653 398 676 496 509 737 187 623 402 486 913 777 187 637 186 33 453 991 915 715 35 607 239 481 778 506 407 642 559 622 922 745 548 12 937 402 723 299 887 365 639 384 387 857 441 393 366 149 130 884 464 386 80 301 667 669 802 670 203 381 960 562 677 764 640 286 467 751 857 353 362 39 332 168 450 420 497 992 265 361 769 142

Hasil data yang sudah di sorting

Data : 7 7 7 8 8 9 9 10 11 11 12 12 16 17 19 23 23 25 33 34 35 35 38 39 43 43 48 48 53 54 56 59 60 63 63 64 65 66 68 70 70 70 74 77 80 81 82 84 85 87 87 89 94 99 99 105 114 120 120 122 123 130 130 132 135 135 136 136 136 138 139 141 142 147 149 150 152 156 157 157 161 163 164 165 167 168 168 169 169 171 171 173 177 183 183 186 187 187 189 195 195 197 200 203 203 204 205 206 208 210 212 212 213 217 219 220 229 233 238 239 239 240 245 245 246 247 248 249 250 256 259 263 265 265 268 270 279 285 286 294 294 294 299 300 301 305 307 311 313 313 316 318 320 322 323 328 329 330 331 332 341 342 345 347 348 348 348 352 353 354 356 357 357 359 361 362 364 365 365 366 366 367 369 369 372 372 372 378 381 382 384 385 385 386 386 387 391 393 398 399 399 401 402 402 402 405 406 407 408 408 409 412 415 416 420 424 425 427 430 431 434 438 439 440 441 441 441 442 442 443 444 445 447 448 450 452 453 454 456 459 459 463 464 465 467 468 469 470 473 476 481 482 483 485 486 486 488 493 493 494 496 497 503 506 506 509 513 514 522 524 525 525 525 525

Lama Proses (Per-Detik) 01,483

Sumber: Hasil Penelitian (2018)

Gambar 12. Uji Coba Lima Ratus Data Menggunakan *Selection Sort*

Tabel 1 menunjukkan hasil keseluruhan pengujian terhadap elemen data dan waktu yang diperlukan oleh algoritma *insertion sort* dan *selection sort* dalam melakukan *sorting*.

Tabel 1. Waktu Yang Dihasilkan Pada Pengujian Algoritma

Algoritma	10 Elemen Data	50 Elemen Data	500 Elemen Data
Insertion Sort	00.003 detik	00.013 detik	01.495 detik
Selection Sort	00.018 detik	00.014 detik	01.483 detik

Sumber: Hasil Penelitian (2018)

4. Kesimpulan

Dengan algoritma yang baik dapat dihasilkan sebuah program yang efisien dari segi waktu dan hasil yang dicapai. Sebuah algoritma sangat efisien ketika jumlah datanya sedikit, namun kinerjanya menjadi berkurang ketika jumlah data ditambahkan atau meningkat. Hasil dari penelitian adalah metode *insertion sort* lebih unggul pada jumlah data yang sedikit yaitu pada sejumlah sepuluh elemen data membutuhkan waktu 00.003 detik, lima puluh elemen data membutuhkan waktu 00.013 detik, sedangkan metode *selection sort* lebih unggul pada jumlah data yang lebih banyak yaitu pada sejumlah lima ratus elemen data membutuhkan waktu 01.083 detik.

Referensi

- Bernardo, Mesterjon, Zulita LN. 2015. Implementasi Metode Selection Sort Untuk Menentukan Nilai Prestasi Siswa Kelas 3 dan Kelas 4 SD Negeri 107 Selumba. *Jurnal Media Infotama* 11(1): 91–100.
- Munir R. 2011. *Algoritma & Pemrograman Dalam Bahasa Pascal dan C*. Bandung: Informatika Bandung.
- Ramadhani. 2015. *Dasar Algoritma dan Struktur Data dengan Bahasa JAVA*. Yogyakarta: CV Andi Offset.
- Saputro FE, Khasanah FN. 2018. Teknik Selection Sort dan Bubble Sort Menggunakan Borland C ++. *Jurnal Mahasiswa Bina Insani*. 2 (2): 136–145.
- Sitorus L, Sembiring DJ. 2012. *Konsep dan Implementasi Struktur Data dengan C++*. Yogyakarta: CV Andi Offset.
- Situmorang H. 2016. Simulasi Pengurutan Data Dengan Algoritma Heap Sort. *Jurnal Mahajana Informasi* 1 (2): 20–30.
- Sjukani M. 2012. *Struktur Data (Algoritma dan Struktur Data 2)*, Edisi 5. Jakarta: Mitra Wacana Media.