APLIKASI SIMULASI PENGURUTAN DATA MENGGUNAKAN ALGORITMA *HEAP SORT*

Ardi Wijaya[1], Noris Feter[2]

[1,2]Program Studi Informatika, Fakultas Teknik, Universitas Muhammadiyah Bengkulu
Jl. Bali PO BOX 118. Telp (0736) 227665, Fax (0736) 26161, Bengkulu 38119

[1]ardi_wijaya18@yahoo.co.id
[2]noris_feter63@yahoo.com

Abstrak: Untuk memecahkan masalah pengurutan dalam membangun suatu program aplikasi, dibutuhkan algoritma pengurutan. Metode-metode pengurutan data pun ada berbagai jenis. Mulai dari *binary sort, insertion sort, merge sort, Heap Sort* dll. *Heap Sort*, algoritma pengurutan, merupakan salah satu metode pengurutan yang sering digunakan. Algoritma *Heap Sort* termasuk algoritma *sorting* yang susah dipahami karena banyak langkah-langkah dalam mengurutkan data. Dikembangkannya perangkat lunak simulasi *Heap Sort*, diharapkan dapat membantu dalam pemahaman algoritma ini. Dalam penelitian ini penulis mengembangkan sebuah program aplikasi simulasi pengurutan data menggunakan algoritma *Heap Sort*. Program aplikasi kompresi yang dibuat dapat digunakan pada sistem operasi *Windows 7* dan *XP*, dengan menggunakan bahasa pemrograman *Microsoft Visual Basic 6.0*. Tujuan akhir dari implementasi perangkat lunak ini adalah diharapkan agar pembaca dan pengembang sistem dimasa yang akan datang dapat mengembangkan dan memperbaiki kekurangan serta keterbatasan yang terdapat pada perangkat lunak ini.
Kata kunci: Program aplikasi, pengurutan data, *Algoritma Heap Sort, Microsoft Visual Basic 6.0*

Abstract: To solve the problem of sorting in building an application program, it needs sorting algorithms. Methods of data sorting also there are various kinds. Starting from the binary sort, insertion sort, merge sort, Heap Sort etc. Heap Sort, sorting algorithms, is one of the sorting method that is often used. Heap Sort algorithm, including sorting algorithm that is difficult to understand because a lot of steps to sort the data. Development of simulation software Heap Sort, is expected to help in the understanding of this algorithm. In this study, the authors developed a simulation application program using the data sorting algorithms sort heap. Compression application program created can be used on the operating system Windows 7 and XP, using Microsoft Visual Basic 6.0. The ultimate purpose of implementing this software is expected that the reader and system developers in the future be able to develop and improve the shortcomings and limitations contained in this software.
Keywords : Program applications, sorting, Heap Sort algorithm, Microsoft Visual Basic *6.0*

I. PENDAHULUAN

Untuk memecahkan masalah pengurutan dalam membangun suatu program aplikasi, dibutuhkan algoritma pengurutan. Di dalam bidang Teknik Informatika terdapat banyak sekali jenis-jenis algoritma pengurutan yang dapat digunakan untuk memecahkan masalah pengurutan. Dalam membangun sebuah aplikasi, orang-orang sering kali dihadapi pada masalah pengurutan data pada aplikasi tersebut. Contoh sederhana ialah pada aplikasi yang berhubungan dengan data yang dapat diurutkan, seperti Nomor Pokok Mahasiswa (NPM), nilai, Nomor ID sebuah barang inventaris, dan lain sebagainya

Pengurutan data (*data sorting*) merupakan bagian dari pengolahan data informasi. Dari data-

data yang telah didapat, ada kalanya data tersebut harus diurutkan terlebih dahulu berdasarkan aturan yang lebih dulu ditentukan. Berdasarkan nilai maupun *alphabet* misalnya.

Metode-metode pengurutan data pun terdiri atas berbagai jenis, yakni *binary sort*, *insertion sort*, *merge sort*, *Heap Sort* dll. Penggunaan metode yang akan dipakai tergantung dari jenis maupun kuantitas data yang diolah. *Heap Sort*, algoritma pengurutan, merupakan salah satu metode pengurutan yang sering digunakan.

Struktur data *heap* adalah sebuah objek *array* yang dapat divisualisasikan dengan sebuah *complete binary tree*. Hubungan antara elemen dari *array* dan *node* pada pohon merupakan hubungan korespondensi satu satu. Pohon diisi secara penuh pada semua level, kecuali kemungkinan terkecil, di mana di sisi dari kiri sampai ke sebuah titik. Semua *node* dari *heap* juga memenuhi relasi bahwa nilai kunci pada setiap *node* minimal sama besar dengan nilai dari *node* anaknya [1].

Struktur data dari algoritma *Heap Sort* adalah sebuah pohon *biner* sempurna yang memenuhi properti *heap*. *Node* akar (*root node*) memiliki data terbesar atau terkecil yang terdapat pada pohon. Demikian juga pada *subtree*-nya, dimana *node* induk (*parent*) memiliki data yang paling besar atau paling kecil dibandingkan dengan data pada kedua anaknya (*child node* sebelah kiri atau sebelah kanan). Algoritma *Heap Sort* termasuk algoritma *sorting* yang susah dipahami karena banyak langkah-langkah dalam mengurutkan data.
Berdasarkan uraian di atas, maka diambil judul "Aplikasi Simulasi Pengurutan Data menggunakan Algoritma *Heap Sort*". Software yang dirancang akan mampu untuk menjelaskan prosedur kerja dari algoritma *Heap Sort.*

II. LANDASAN TEORI

A. *Struktur Data*

Struktur berarti susunan / jenjang, dan data berarti sesuatu simbol / huruf / lambang angka yang menyatakan sesuatu. Struktur data berarti susunan dari simbol / huruf / lambang angka untuk menyatakan sesuatu hal [2]. Sebagai contoh, struktur program Pascal dapat didefenisikan seperti berikut,

(i) Deklarasi Nama Fungsi / Prosedur.
(ii) Deklarasi Tipe Data.
(iii) Deklarasi Konstanta (untuk variabel bernilai nilai statis).
(iv) Deklarasi Variabel.
(v) Deklarasi Label.
(vi) Badan Program (*Begin … End*.)

Gabungan dari algoritma dan struktur data akan membentuk suatu program. Adapun manfaat dari struktur data adalah sebagai berikut,

a. Mengefisiensikan program.

Program yang dibuat dengan menerapkan konsep–konsep yang berlaku pada struktur data akan lebih efisien dibandingkan dengan program yang dibuat dengan mengabaikan konsep struktur data.

b. Modifikasi

Sesuatu program harus dapat dimodifikasi apabila diperlukan, hal ini dapat dilakukan jika fasilitas yang diperlukan dibuat (disertakan) walaupun pada tahap awal belum dipakai.

c. Memilih metode yang tepat

Misalkan suatu *plaza* pada hari – hari tertentu mengalami antrian yang panjang pada kasir, hal ini dapat diatasi dengan metode,

- Pemasukan data tidak melalui *keyboard* lagi, melainkan melalui *barcode*.
- Membuat pemberitahuan pada kasir – kasir

.

B. *Pohon Biner*

Pohon (*tree*) merupakan struktur data nonlinier yang banyak digunakan dalam aplikasi sehari-hari. Contoh aplikasi pohon yang dapat kita lihat sehari-hari adalah pengelolaan *file* dalam direktori penyimpanan. Pohon merupakan struktur data yang memiliki suatu struktur hirarki pada sekumpulan elemen, dan memiliki hubungan satu ke banyak (*one to may relationship*) seperti yang kita lihat dalam struktur organisasi sebuah perusahaan atau daftar isi sebuah buku.

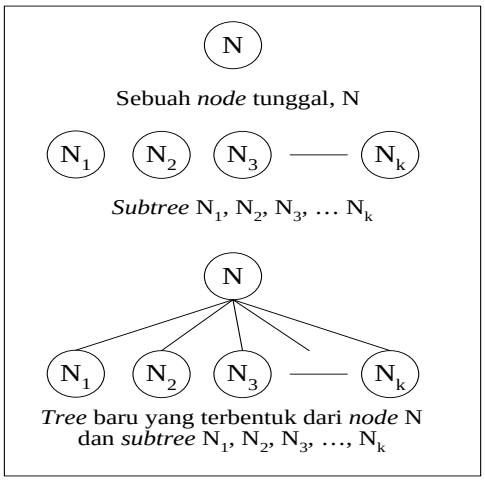
Dalam struktur organisasi, kita dapat melihat bahwa ada level atas biasanya hanya ada satu pimpinan tertinggi. Pada level berikutnya diisi oleh beberapa orang dengan jabatan yang berbeda tetapi dalam tingkatan yang sama. Selanjutnya dapat dipecah lagi ke level berikutnya sampai struktur dapat memenuhi fungsi dan tujuan organisasi. Biasanya satu atasan memiliki beberapa bawahan yang berada dalam ruang lingkup wewenang dan tugas atasan. Begitu juga dalam daftar isi buku, dimana satu buku terdiri dari beberapa bab dan setiap terdiri dari beberapa sub bab, satu sub bab terdiri dari beberapa sub sub bab dan seterusnya. Dengan demikian hirarki dapat kita anggap sebagai "terdiri dari" atau "bawahan" atau "diawasi" dari atas ke bawah. Salah satu keuntungan pohon dibandingkan dengan struktur data linier adalah waktu cari sebuah *node* maksimum (dapat) lebih kecil dari n jika jumlah data = n.

Sebuah *tree* dapat mempunyai hanya sebuah simpul tanpa sebuah sisi pun. Dengan kata lain, jika $G = (V, E)$ adalah tree, maka V tidak boleh berupa himpunan kosong, namun E boleh kosong. *Tree* juga seringkali didefinisikan sebagai graf tak-berarah dengan sifat bahwa hanya terdapat sebuah lintasan unik antara setiap pasang simpul. Selain itu, di dalam *tree* jumlah sisinya adalah jumlah simpul dikurangi satu.

Secara sederhana, sebuah *tree* bisa didefenisikan sebagai kumpulan dari elemen – elemen yang disebut dengan *node / vertex* (simpul) dimana salah satu *node* disebut dengan *root* (akar), dan sisa *node* lain terpecah menjadi himpunan yang saling tidak berhubungan satu sama lain dan disebut dengan *subtree* (pohon bagian). Jika dilihat pada setiap *subtree* maka *subtree* juga mempunyai *root* dari *subtree*-nya masing – masing.

Dengan melihat istilah dasar di atas, maka sebuah *tree* secara rekursif dapat didefenisikan sebagai berikut :

1. Sebuah *node* tunggal adalah sebuah *tree*.
2. Jika terdapat sebuah *node* N dan beberapa *subtree* $N_1, N_2, N_3, \ldots, N_k$ maka dari *node* N dan *subtree* yang ada dapat dibentuk sebuah *tree* yang mempunyai *root* pada *node* N.



Gambar 1. Contoh pembentukan *Tree*

Binary Tree (pohon *biner*) didefenisikan sebagai suatu kumpulan *node* yang mungkin kosong atau mempunyai *root* dan paling banyak dua *subtree* (anak) yang saling terpisah yang disebut dengan *left subtree* (pohon bagian kiri / anak kiri / cabang kiri) dan *right subtree* (pohon bagian kanan / anak kanan / cabang kanan). *Subtree* bisa disebut juga dengan istilah *branch* (cabang).

Binary tree merupakan tipe yang sangat penting dari struktur data *tree*, dan banyak dijumpai dalam berbagai terapan. Lebih lanjut, dalam *binary tree* akan dibedakan antara *left subtree* dengan *right subtree*, sementara dalam struktur *tree* secara umum urutan ini tidak penting. Jadi *binary tree* merupakan bentuk *tree* yang beraturan. Karakteristik lain adalah bahwa dalam *binary tree* dimungkinkan tidak mempunyai *node*. Gambar 2 berikut ini menunjukkan contoh suatu *binary tree*.

Gambar 2. Contoh binary tree

Prosedur dasar yang terdapat dalam *heap tree* adalah:

1. *Agoritma Heapify*.

Algoritma *Heapify* adalah membangun sebuah *heap* dari bawah ke atas, secara berturut – turut berubah ke bawah untuk membangun *heap*. Permasalahan pertama yang harus kita pertimbangkan dalam melakukan *Heapify* adalah dari bagian mana kita harus memulai. Bila kita mencoba operasi *Heapify* dari akar maka akan terjadi operasi runut –naik seperti algoritma bouble sort yang akan mnyebabkan kompleksitas waktu [3]. Berikut adalah algoritma prosedur *Heapify:*

HEAPIFY (*A*, *i*)

1. $l \leftarrow$ left [i]
2. $r \leftarrow$ right [i]
3. if $l \leq$ *heap*-size [A] and $A[l] > A[i]$
4. then largest $\leftarrow l$
5. else largest $\leftarrow i$
6. if $r \leq$ *heap*-size [A] and $A[i] >$ A[largest]
7. then largest $\leftarrow r$
8. if largest $\neq i$
9. then exchange $A[i] \leftrightarrow A$[largest]
10. *Heapify* (*A*, largest)

2. *Build-Heap.*

Apabila diberi *input* sebuah pohon *biner* atau sekumpulan data *array*, maka untuk menjadikan pohon *biner* ini pohon *heap*, kita harus memastikan semua data pada pohon *biner* memenuhi properti *heap*. Berikut adalah algoritma *Build-Heap*:

BUILD-HEAP (*A*)

1. *heap*-size (*A*) $\leftarrow$ length [*A*]
2. For $i \leftarrow$ floor(length[*A*]/2) down to 1 do
3. *Heapify* (*A, i*)

3. *Heap Sort*

Algoritma *heapsort* adalah algoritma pengurutan yang memiliki kompleksitas waktu terbaik. Selain itu juga, *heapsort* menerapkan teknik yang unik di dalam memecahkan masalah pengurutan, yaitu dengan menggunakan *heaptree* [4] Prosedur *Heap Sort* mengurutkan sekumpulan data pada sebuah *array* atau pohon *heap*. Cara kerjanya adalah, *Heap Sort* akan mengambil data pada *node* akar (*index array* = 1) dan menggantinya (*exchange*) dengan data pada *node* paling akhir (*index array* = *index* paling maksimum dari pohon *heap*). Setelah itu, *node* terakhir dihapus dan *Heap Sort* memanggil

prosedur *heapify* dengan tujuan agar setelah proses penggantian data, pohon masih memenuhi properti *heap*. Data-data yang dikeluarkan merupakan data yang terurut, baik menaik (*ascending*) maupun menurun (*descending*).

III. METODOLOGI PENELITIAN

Jenis penelitian yang digunakan dalam penelitian ini adalah penelitian terapan. Penelitian terapan adalah penyelidikan yang hati-hati, sistematik dan terus menerus terhadap suatu masalah dengan tujuan untuk digunakan dengan segera untuk keperluan tertentu [6].

Teknik pengumpulan data pada penelitian terapan ini menggunakan teknik studi pustaka (*Library research*). yaitu dengan mempelajari konsep-konsep dasar mengenai *heapsort* yang terdapat pada beberapa sumber literatur. Sumber literatur dapat berupa buku teks, *paper*, *website*, *blog*, laporan penelitian, karangan-karangan ilmiah, tesis dan disertasi dan sumber-sumber tertulis baik tercetak maupun elektronik yang berhubungan dengan penelitian.

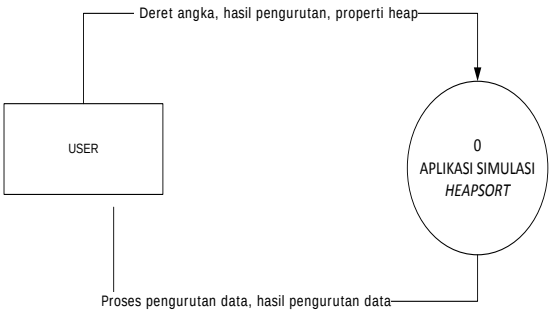
IV. ANALISA DAN PERANCANGAN

Algoritma ini berfungsi untuk mengurutkan sekumpulan data pada *list array* A. Cara kerjanya adalah, *Heap Sort* akan mengambil data pada *node* akar (*index array* = 1) dan menggantinya (*exchange*) dengan data pada *node* paling akhir (*index array* = *index* paling maksimum dari pohon *heap*). Setelah itu, *node* terakhir dihapus dan *Heap Sort* memanggil prosedur *heapify* untuk *node* ke-1 dengan tujuan supaya setelah proses penggantian data, pohon masih memenuhi properti *heap*. Data-data yang dikeluarkan merupakan data yang terurut, baik menaik (*ascending*) maupun menurun (*descending*).

Algoritma *Heap-Sort(A)* adalah sebagai berikut:

1. Panggil prosedur Build-Heap(A).
2. Set Urut = “”.
3. Untuk j = ukuran *array* A sampai 2 dengan pengurangan nilai 1 setiap *looping*, lakukan algoritma di bawah ini.
    a. Jika (TipeHeap = “S” dan TipeUrutan = “A”) Atau (TipeHeap = “L” dan TipeUrutan = “D”) maka:
        i. Jika Urut <> “” maka set Urut = Urut & “”.
        ii. Set Urut = Urut & A(1).
    b. Jika tidak, maka set Urut = A(1) & IIf(Urut <> "", ",", "") & Urut.
    c. Tukarkan data pada A(1) dan A(j).
    d. Kurangi ukuran *array* A dengan 1.
    e. Jika Tipe*Heap* = “S”, maka panggil prosedur *Heapify*S(A, 1).
    f. Jika Tipe*Heap* = “L”, maka panggil prosedur *HeapifyL*(A, 1).
4. Jika (Tipe*Heap* = “S” dan TipeUrutan = “A”) Atau (Tipe*Heap* = “L” dan TipeUrutan = “D”) maka:
    a. Jika Urut <> “” maka set Urut = Urut & “”.
    b. Set Urut = Urut & A(1).
5. Jika tidak, maka set Urut = A(1) & IIf(Urut <> "", ",", "") & Urut.

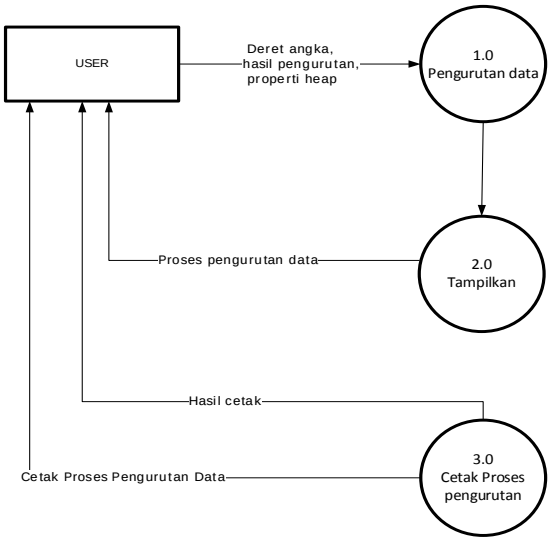
Untuk menggambarkan hubungan sistem dengan lingkungan luarnya diperlukan diagram konteks. Diagram konteks dari sistem yang akan dirancang adalah seperti Gambar 3 di bawah ini.



Gambar 3. Diagram Konteks

Diagram konteks menunjukan bahwa sistem terbentuk dari satu sistem besar yaitu aplikasi simulasi *heapsort*. Diagram konteks juga menggambarkan tahap utama system

Diagram konteks pada Gambar 3 dapat diperinci menjadi DFD level-1. Proses-proses pada DFD level-1 merupakan dekomposisi dari proses pada diagram konteks. Proses-proses tersebut dapat dilihat pada Gambar 4.



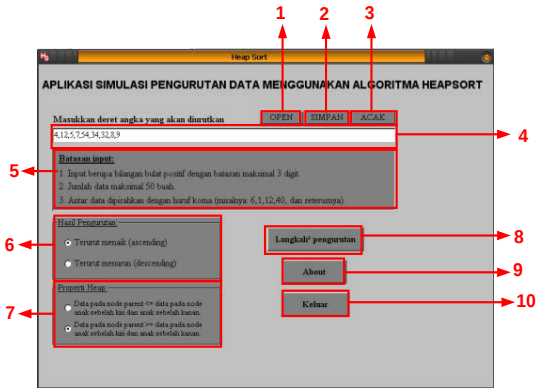
Gambar 4. DFD Level 1

IV. HASIL DAN PEMBAHASAN

Pembuatan perangkat lunak bantu pemahaman *Heap Sort* ini mencakup beberapa bagian penting, yaitu:

1. Cara memasukkan *input* ke dalam perangkat lunak.
2. Cara kerja perangkat lunak.
3. Proses penggambaran pohon *biner*.
4. Proses pengurutan *Heap Sort*, mencakup 3 (tiga) buah prosedur, yaitu:
   a. Prosedur *Heapify*.
   b. Prosedur *Build-Heap*.
   c. Prosedur *HeapSort*.
5. Perangkat lunak dimulai dari *Form* Main. Pada *form* ini, *user* dapat meng-*input* barisan angka yang akan diurutkan atau menghasilkan barisan angka secara acak. Setelah itu, tekan tombol ’Langkah-Langkah Pengurutan’ akan membuka *Form* Pengurutan. Pada *form* ini, perangkat lunak akan menjelaskan dan menampilkan proses kerja algoritma *Heap Sort* dalam melakukan pengurutan. Proses pengurutan dapat dihentikan sementara (*pause*) dan dilanjutkan kembali (*resume*).

A. *Form Main*



Gambar 5. *Form Main*

1. *Open* (1)
   Tombol ini berfungsi untuk me*load file* deret angka yang berekstensi (.hps).
2. *Button* “*Simpan*” (2)
   Tombol ini berfungsi untuk menyimpan deret angka yang di*input*kan pada sistem. Deret angka tersebut disimpan dalam ekstensi (.hps).

3. *Button* "*Acak*" (3)

   Tombol ini berfungsi untuk mengacak deret angka yang di*input*kan.

4. *Input* Deret Angka (4)

   Menu ini untuk tempat meng*input*kan data deret angka yang akan diurutkan, untuk meng*input* deret angka yang akan diuruttkan ada 2 cara yaitu dengan meng*input*kan file berekstensi (.hps) yang telah disimpan sebelumnya atau meng*input*kan langsung pada menu ini.

5. Batasan *input (5)*

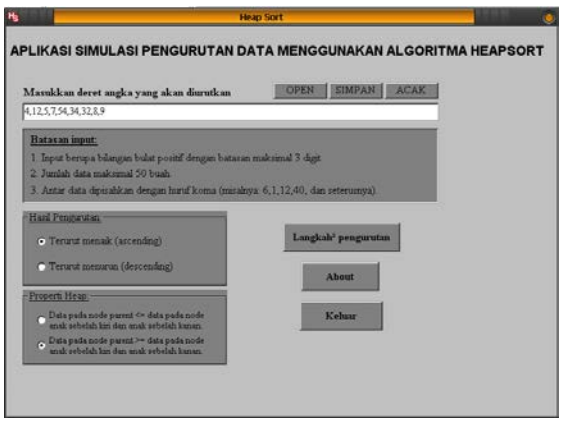
   Batasan *input* adalah syarat-syarat dalam meng*input*kan data deret angka agar dapat diproses oleh sistem.

6. Menu Hasil Pengurutan (6)

   Pada menu hasil pengurutan teerdapat 2 pilihan yaitu descending (terurut menurun) dan ascending (terurut menaik), pilihan ini untuk menentukan hasil penguruan deret angka yang di*input*kan kedalam sistem.

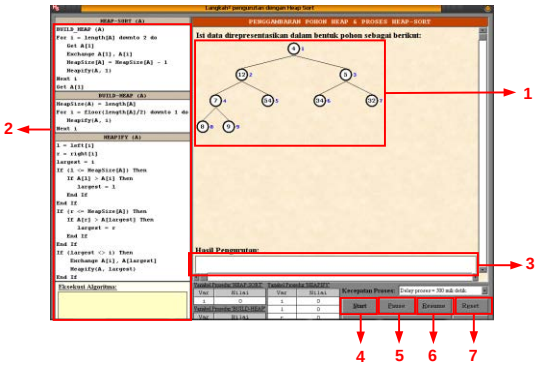
B. Pengujian Program

Sebagai contoh pengujian program, misalkan *input* barisan data yang akan diurutkan = 4,12,5,7,54,34,32,8,9. Hasil pengurutan yang diinginkan adalah terurut menaik (*ascending*) dan properti *heap* yang dipilih adalah properti dengan data pada *node parent* >= data pada anak sebelah kiri atau anak sebelah kanan.



Gambar 6. Tampilan *Form Main* dengan Data = 4,12,5,7,54,34,32,8,9

C. *Form T*ampilan Pengurutan Data



Gambar 7. Tampilan *Form* Pengurutan dengan Data

1. Gambar Pohon *Heapsort* (1)

   Gambar pohon *heapsort* yang terbentuk setelah di*input*kan data deret angka ke dalam sistem.

2. Eksekusi Algoritma (2)

   Proses algoritma yang berjalan secara terstruktur akan ditampilkan pada menu ini.

3. Menu "*Hasil Pengurutan*" (3)

   Hasil Pengurutan akan ditampilkan pada menu ini

4. *Button* " *Start*" (4)

   Tombol ini berfungsi untuk memulai proses pengurutan data.

5. *Button "pause"* (5)

   Tombol ini berfungsi untuk memberhentikan sementara proses pengurutan data.

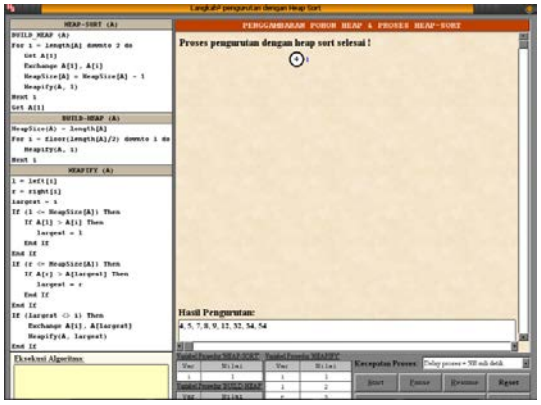
6. *Button "resume"* (6)

   Tombol ini berfungsi untuk melanjutkan proses pengurutan yang di pause.

7. *Button "reset"* (7)

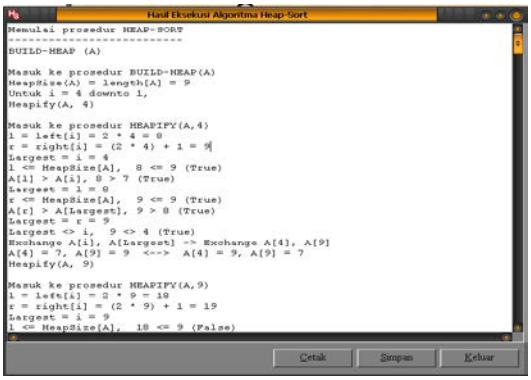
   Tombol ini berfungsi untuk mengulang kembali proses yang sedang berjalan.

Tampilan *Form* Pengurutan setelah proses pengurutan dapat dilihat pada gambar 8 di bawah ini.



Gambar 8. Tampilan *Form* Pengurutan Setelah Proses Pengurutan Dengan Data

D. *Form* Hasil Eksekusi



Gambar 9. Tampilan Hasil Eksekusi

E. *Form About*

Form ini adalah *form* about yaitu *form* yang menampilkan pembuat aplikasi Tampilan *form About* pada perangkat lunak seperti terlihat pada gambar 10 berikut.

Gambar 10. *Form About*

Dari pengujian di atas disimpulkan bahwa aplikasi simulasi pengurutan data menggunakan algoritma *Heap Sort* dapat diimplementasikan menggunakan vb 6.0, sehingga aplikasi ini nanti diharapkan dapat membantu dalam proses belajar mengajar mengenai algoritma *heapsort*.

VI. KESIMPULAN

Berdasarkan analisi, perancangan dan pembahasan terhadap algoritma *heapsort* ini, maka dapat diambil kesimpulan berupa:

1. Perangkat lunak menjelaskan algoritma pengurutan *Heap Sort* dan gambar keadaan pohon *biner* secara bertahap serta menampilkan *Form Teori*, sehingga dapat membantu pemahaman mengenai pengurutan dengan metode *Heap Sort.*
2. Penggunaan aplikasi ini dapat mempermudah dalam proses belajar mengajar.

REFERENSI

[1] Bambang Hariyanto, *Struktur Data : Memuat Dasar Pengembangan Orientasi Objek*, Edisi Kedua, Informatika Bandung, April 2003

[2] Robert L.Kruse, *Data Structures & Program Design, Second Edition*, 1991

[3] Firdi Mulia**,** *Penerapan Pohon Dalam Heap Sort***,** ITB Bandung

[4] Chalikdjen, Efendy dkk *Heap Tree dan Kegunaannya dalam Heap Sort*. Makalah STMIK. Institut Teknologi Bandung. [Online] tersedia:

[5] http://MakalahStemik08 .pdf. [15 April 2008]

[6] Umar, Husein. 2005. *Metode Penelitian Untuk Skripsi dan Tesis Bisnis*. Jakarta : PT. Raja Grafindo Persada