Jurnal Mahajana Informasi, Vol.1 No 2, 2016
e-ISSN: 2527-8290

SIMULASI PENGURUTAN DATA DENGAN ALGORITMA *HEAP SORT*

Harold Situmorang
Program Studi Sistem Informasi Universitas Sari Mutiara Indonesia
Haroldsitumorang@gmail.com

ABSTRAK

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). 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, dimana diisi 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. Setelah menyelesaikan perangkat lunak bantu pemahaman Heap Sort, 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. Algoritma pengurutan Heap Sort merupakan salah satu metode pengurutan tercepat setelah Merge Sort dan Quick Sort dengan kompleksitas O(n log n). Kompleksitas prosedur Build-Heap adalah O(n), kompleksitas prosedur Heapify adalah O(log n), sehingga kompleksitas algoritma Heap Sort adalah O(n log n).

Kata Kunci : *Heap Sort*, *Merge Sort*, *Quick Sort*, kompleksitas O(n log n).

I. PENDAHULUAN

1.1 Latar Belakang

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, dimana diisi 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.

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). Perangkat lunak yang dirancang akan mampu untuk menjelaskan prosedur kerja dari algoritma Heap Sort.

1.2 Perumusan Masalah

Berdasarkan latar belakang pemilihan judul, maka yang menjadi permasalahan adalah menampilkan prosedur kerja dari algoritma Heap Sort.

1.3 Batasan Masalah

Ruang lingkup permasalahan dalam merancang perangkat lunak ini dibatasi sebagai berikut :

1. Angka yang di-*input* bertipe data bilangan *integer* positif dengan batasan maksimal 3 digit.
2. Jumlah data yang di-*input* dibatasi maksimal 50 buah.
3. Perangkat lunak memiliki fasilitas *pause* (menghentikan sementara) untuk proses kerja algoritma pengurutan Heap Sort dan *resume* untuk melanjutkan proses kerja algoritma pengurutan Heap Sort.
4. Perangkat lunak akan menggambarkan visualisasi heap sebagai pohon biner lengkap (*complete binary tree*).

1.4 Tujuan Penelitian

Tujuan penelitian ini adalah untuk merancang suatu perangkat lunak yang mampu untuk menjelaskan dan menampilkan prosedur kerja dari algoritma Heap Sort.

1.5 Manfaat Penelitian

Manfaat dari penelitian ini, yaitu :

1. Untuk membantu pemahaman mengenai algoritma Heap Sort.
2. Perangkat lunak juga dapat digunakan sebagai fasilitas pendukung dalam proses belajar mengajar, terutama pada mata kuliah Struktur Data.

II LANDASAN TEORI

2.1 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. 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.
b. 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.
c. Modifikasi

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

e. 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.

2.1.1 Array

Array (larik) adalah suatu tipe data terstruktur yang dapat menampung (berisikan) suatu data yang sejenis. Komponen – komponen dari *Array* antara lain,

a. Nama *Array*
b. Nilai *Array*
c. Index *Array*
d. Jenis (tipe) *Array*

Deklarasi *Array* dapat dibuat pada,

a. Bagian Tipe

Contoh,

Type A = Array [1..5] of Char;

b. Bagian Variabel

Contoh,

Var A = Array [1..5] of Char;

Array terdiri dari beberapa jenis antara lain,

a. Array 1 dimensi

Contoh misalkan diketahui sebuah *Array* A dengan jenis data *Char*,

A	B	C	D	E	F	G	H
1	2	3	4	5	6	7	8

A =

A[1] = ‘A’

A[2] = ‘B’

A[3] = ‘C’

A[4] = ‘D’

A[5] = ‘E’

A[6] = ‘F’

A[7] = ‘G’

A[8] = ‘H’

b. *Array* Multi dimensi

Contoh, misalkan diketahui sebuah *Array* dua dimensi B dengan jenis data *Integer*,

B[1,1] = 1 B[1,2] = 2 B[2,1] = 3

Proses - proses yang mungkin dilakukan pada *Array* adalah,

a. Proses memasukkan data
b. Proses mencari data
c. Proses menghapus data
d. Proses mencetak data
e. Proses menyisip data
f. Proses mencari posisi
g. Proses mengurutkan data
h. Dan sebagainya

2.2 Algoritma

Kata algoritma berasal dari nama seorang ahli matematika berkebangsaan Persia yang hidup pada abad ke – 9 yang bernama Abu Abdullah Muhammad bin Musa Al-Khawarizmi. Pada awalnya, kata 'algorism' diartikan sebagai aturan-aturan untuk melakukan proses aritmatika menggunakan numerik Arab. Kata 'algorism' diubah menjadi kata 'algorithm' pada abad ke – 18. Sekarang, pengertian dari kata ini mencakup semua prosedur terhingga untuk menyelesaikan problema atau melakukan pekerjaan.

Penerapan pertama dari algoritma yang ditulis untuk sebuah komputer adalah '*Ada Byron's notes on the analytical engine*' yang ditulis pada tahun 1842, dimana Ada Byron dianggap oleh kebanyakan orang sebagai *programmer* pertama di dunia. Walaupun, sejak Charles Babbage tidak menyelesaikan *analytical engine*-nya, algoritma ini tidak pernah diimplementasikan lagi.

2.2.1 Definisi Algoritma

Algoritma adalah suatu kumpulan terhingga (*finite set*) dari instruksi yang terdefinisi dengan baik (*well-defined instructions*) untuk menyelesaikan beberapa pekerjaan dimana diberikan *state* awal (*initial state*) dan akan dihentikan pada saat ditemukan *state* akhir (*end-state*) yang dikenal. Algoritma dapat diimplementasikan dalam pembuatan program komputer. Kesalahan dalam merancang algoritma untuk menyelesaikan suatu problema dapat menyebabkan program gagal dalam implementasinya.

Konsep dari suatu algoritma sering diilustrasikan dengan mengambil contoh sebuah resep, walaupun banyak algoritma yang jauh lebih kompleks. Algoritma sering memiliki beberapa langkah perulangan (iterasi) atau memerlukan pengambilan keputusan seperti logika (*logic*) atau perbandingan (*comparison*) sampai pekerjaan diselesaikan. Menerapkan suatu algoritma secara benar belum tentu dapat menyelesaikan problema. Hal ini dikarenakan adanya kemungkinan algoritma tersebut rusak atau cacat, atau penerapannya tidak cocok (tidak tepat) untuk menyelesaikan problema. Sebagai contoh, sebuah algoritma hipotesis untuk membuat sebuah salad kentang akan gagal jika tidak terdapat kentang.

Suatu pekerjaan dapat diselesaikan dengan menggunakan algoritma yang berbeda dengan kumpulan instruksi (*set of instructions*) yang berbeda dengan perbedaan waktu akses, efisiensi tempat, usaha dan sebagainya. Sebagai contoh, diberikan dua buah resep yang berbeda untuk membuat salad kentang, resep pertama mengupas kulit kentang terlebih dahulu sebelum memasak kentang tersebut, sementara resep lainnya dilakukan dengan langkah yang terbalik, dan kedua resep akan mengulangi kedua langkah tersebut dan akan dihentikan pada saat salad kentang siap untuk dimakan.

Algoritma adalah hal yang mendasar untuk komputer dalam memproses informasi, karena

sebuah program komputer adalah sebuah algoritma yang memberitahukan kepada komputer langkah – langkah spesifik yang akan dijalankan (dalam urutan spesifik) untuk melakukan pekerjaan tertentu, misalnya menghitung gaji karyawan atau mencetak rapor murid. Oleh karena itu, algoritma dapat dianggap sebagai beberapa operasi sekuensial (terurut) yang dapat dijalankan oleh sebuah sistem lengkap *Turing*.

Hukum di beberapa negara seperti Amerika Serikat, mengizinkan beberapa algoritma untuk mendapatkan hak paten secara efektif, walaupun kemungkinan tersedianya algoritma tersebut dalam sebuah perwujudan fisik (*physical embodiment*). Sebagai contoh, sebuah algoritma perkalian mungkin diwujudkan dalam unit aritmatika dalam sebuah mikroprosesor.

2.2.2 Kompleksitas Algoritma

Suatu algoritma memiliki kemungkinan terbaik (*best case*) dan kemungkinan terburuk (*worst case*). *Best Case* maksudnya adalah waktu eksekusi tercepat dari algoritma, sedangkan *Worst Case* maksudnya adalah waktu eksekusi terlama dari algoritma. Waktu eksekusi dari algoritma ini biasanya disebut dengan kompleksitas algoritma. Kompleksitas algoritma biasanya dinyatakan dengan notasi O (*Big-O*).

Suatu algoritma dikatakan memiliki kompleksitas $O(n^2)$ berarti bahwa waktu eksekusi terlama dari algoritma tersebut adalah sebesar kuadrat dari banyaknya elemen data yang akan diproses.

Semakin kecil kompleksitas suatu algoritma maka algoritma tersebut dikatakan lebih efisien dan efektif. Algoritma semacam inilah yang lebih disukai orang untuk digunakan dalam penyelesaian masalah.

Dalam analisis matematika khususnya analisis algoritma, untuk menentukan pertumbuhan dari fungsi, dapat digunakan *asymtotic notations* (notasi asimtot). Knuth mengutarakan beberapa notasi antara lain,

- *Big-O*
- *Big-* t *(Omega)*
- *Big-*n *(Theta)*

Notasi – notasi di atas tidak memiliki sifat matematika konvensional dan tidak berlaku ketentuan – ketentuan matematika di dalamnya. Suatu fungsi f(x) memiliki notasi O(x) dan fungsi lainnya g(x) juga memiliki notasi O(x), namun ini tidak berarti bahwa f(x) sama dengan g(x).

Notasi *Big-O* adalah suatu cara untuk membandingkan fungsi dan sangat berguna untuk menghitung kompleksitas dari algoritma, misalnya banyaknya waktu yang diperlukan oleh komputer untuk menjalankan program.

Contoh,

$3x^3 + 5x^2 – 9 = O(x^3)$

Persamaan di atas tidak berarti bahwa ada suatu fungsi $O(x^3)$ yang sama dengan fungsi $3x^3 + 5x^2 – 9$, namun persamaan tersebut berarti bahwa $3x^3 + 5x^2 – 9$ memiliki big-O x^3 atau dapat dikatakan bahwa $3x^3 + 5x^2 – 9$ didominasi secara asimtot oleh x^3. Notasi asimtot mencerminkan kelakuan

dari fungsi untuk nilai yang besar sehingga notasi asimtot kadang kala tidak berlaku untuk nilai yang kecil.

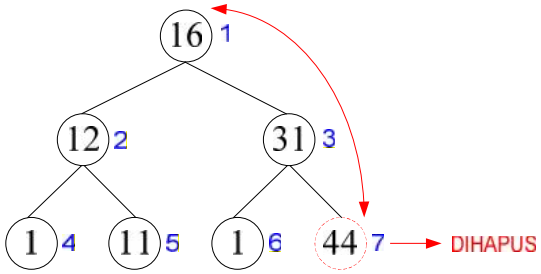
2.3 HEAP-SORT

Pengurutan dengan *heap-sort* mengambil data dari pohon *heap* satu per satu. Data yang diambil adalah data pada *node* 1, data pada *node* 1 kemudian ditukarkan dengan data pada *node* terakhir dan *node* terakhir dihapus.

a. Ambil angka '44' pada *node* 1, sehingga deretan angka menjadi:

[12, 31, 1, 11, 1, 16], 44

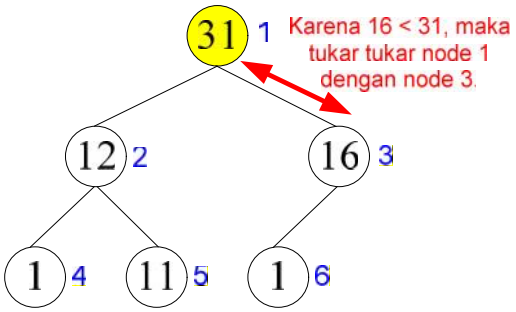
Tukar *node* 1 dengan *node* 7 (*node* terakhir) dan hapus *node* terakhir.



Gambar 2.28 Proses penukaran angka pada *node* 1 dan *node* 7 dihapus

Kemudian lakukan proses Heapify(A, 1).

Prosedur Heapify(A,1)



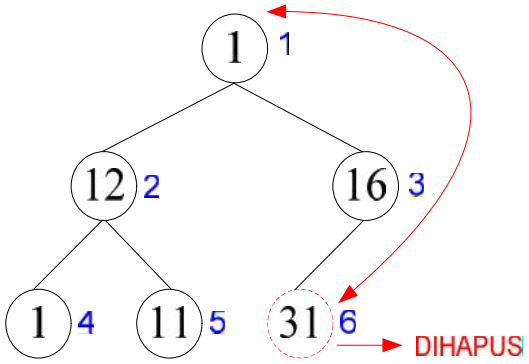
Gambar 2.29 Proses penukaran angka pada *node* 1 dengan *node* 3

Proses Heapify berlanjut pada *node* 3 (*node* yang ditukar dengan *node* 1), tetapi karena *node* 3 memiliki data terbesar dibandingkan dengan *node* anaknya, maka struktur pohon tidak berubah dan prosedur Heapify berakhir.

b. Ambil angka '31' pada *node* 1, sehingga deretan angka menjadi:

[12, 1, 11, 1, 16], 31, 44

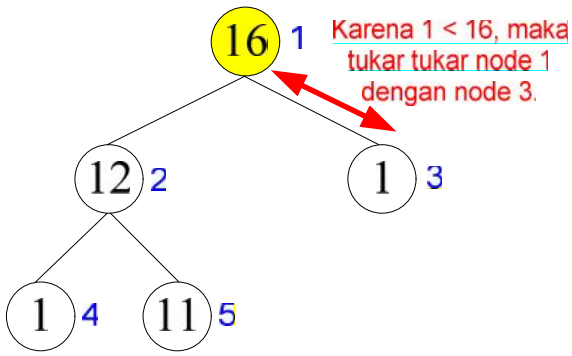
Tukar *node* 1 dengan *node* 6 (*node* terakhir) dan hapus *node* terakhir.



Gambar 2.30 Proses penukaran angka pada *node* 1 dan *node* 6 dihapus

Kemudian lakukan proses Heapify(A, 1).

Prosedur Heapify(A,1)



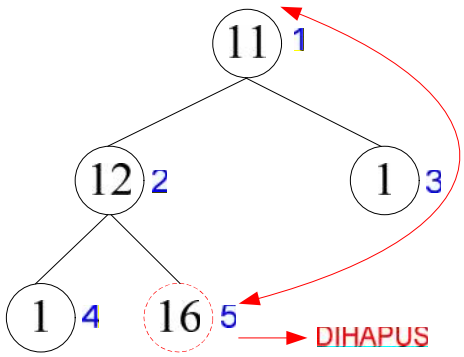
Gambar 2.31 Proses penukaran angka pada *node* 1 dengan *node* 3

Proses Heapify berlanjut pada *node* 3 (*node* yang ditukar dengan *node* 1), tetapi karena *node* 3 tidak memiliki anak, maka struktur pohon tidak berubah dan prosedur Heapify berakhir.

c. Ambil angka '16' pada *node* 1, sehingga deretan angka menjadi:

[12, 1, 11, 1], 16, 31, 44

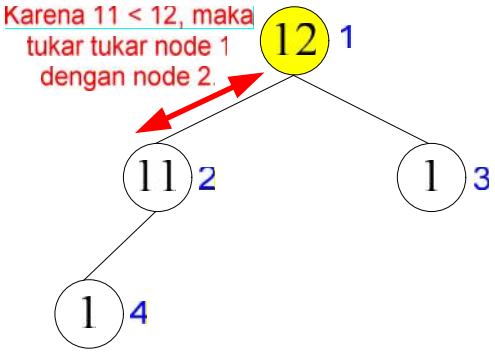
Tukar *node* 1 dengan *node* 5 (*node* terakhir) dan hapus *node* terakhir.



Gambar 2.32 Proses penukaran angka pada *node* 1 dan *node* 5 dihapus

Kemudian lakukan proses Heapify(A, 1).

Prosedur Heapify(A,1)



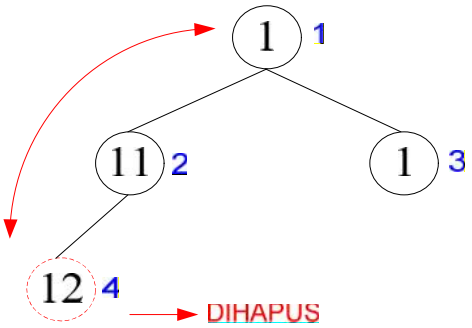
Gambar 2.33 Proses penukaran angka pada *node* 1 dengan *node* 2

Proses Heapify berlanjut pada *node* 2 (*node* yang ditukar dengan *node* 1), tetapi karena *node* 2 tidak memiliki anak, maka struktur pohon tidak berubah dan prosedur Heapify berakhir.

d. Ambil angka '12' pada *node* 1, sehingga deretan angka menjadi:

[1, 11, 1], 12, 16, 31, 44

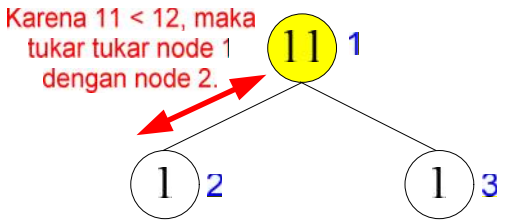
Tukar *node* 1 dengan *node* 4 (*node* terakhir) dan hapus *node* terakhir.



Gambar 2.34 Proses penukaran angka pada *node* 1 dan *node* 4 dihapus

Kemudian lakukan proses Heapify(A, 1).

Prosedur Heapify(A,1)



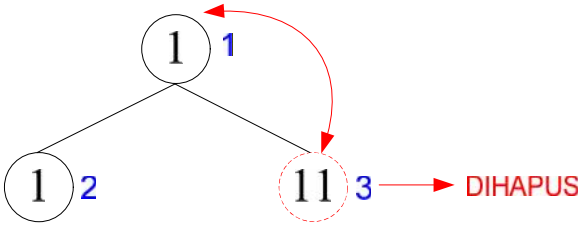
Gambar 2.35 Proses penukaran angka pada *node* 1 dengan *node* 2

Proses Heapify berlanjut pada *node* 2 (*node* yang ditukar dengan *node* 1), tetapi karena *node* 2 tidak memiliki anak, maka struktur pohon tidak berubah dan prosedur Heapify berakhir.

e. Ambil angka '11' pada *node* 1, sehingga deretan angka menjadi:

[1, 1], 11, 12, 16, 31, 44

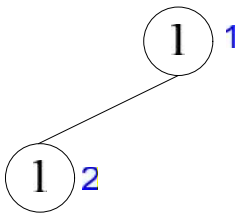
Tukar *node* 1 dengan *node* 3 (*node* terakhir) dan hapus *node* terakhir.



Gambar 2.36 Proses penukaran angka pada *node* 1 dan *node* 3 dihapus

Kemudian lakukan proses Heapify(A, 1).

Prosedur Heapify(A,1)



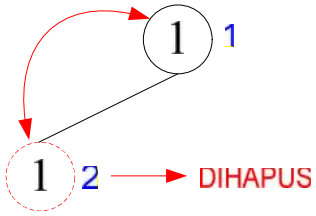
Gambar 2.37 Proses penukaran angka pada *node* 1 dengan *node* 2

Proses Heapify tidak mengubah struktur pohon karena *node parent* tidak lebih besar daripada *node* anak.

f. Ambil angka '1' pada *node* 1, sehingga deretan angka menjadi:

[1], 1, 11, 12, 16, 31, 44

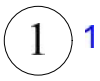
Tukar *node* 1 dengan *node* 2 (*node* terakhir) dan hapus *node* terakhir.



Gambar 2.38 Proses penukaran angka pada *node* 1 dan *node* 2 dihapus

Kemudian lakukan proses Heapify(A, 1).

Prosedur Heapify(A,1)



Gambar 2.39 Proses penukaran angka pada *node* 1 dengan *node* 2

Proses Heapify tidak mengubah struktur pohon karena *node* 1 tidak memiliki anak lagi.

g. Ambil angka '1' pada *node* 1, sehingga deretan angka menjadi:

1, 1, 11, 12, 16, 31, 44

Hasil akhir adalah berupa deretan angka terurut secara menaik (*ascending*). Untuk angka terurut menurun (*descending*), tempatkan angka yang dikeluarkan dari pohon *heap* di depan deretan.

2.4.8 Kompleksitas Algoritma Pengurutan

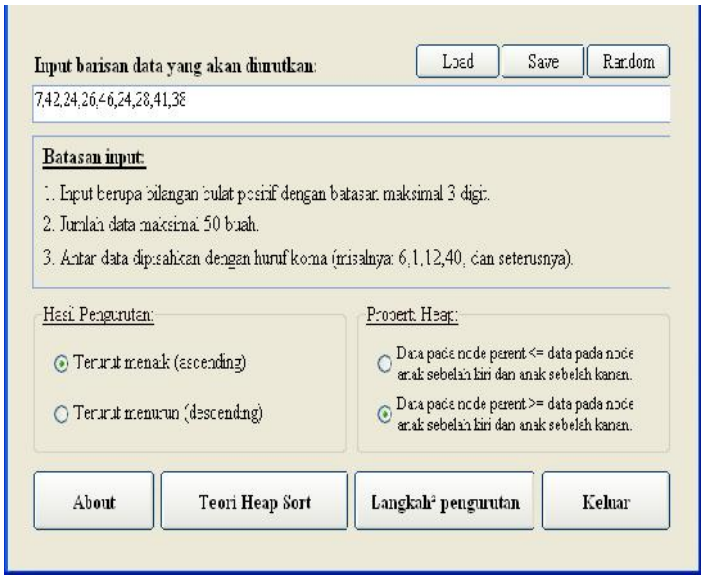
Berdasarkan kompleksitas algoritma (waktu eksekusi algoritma) untuk melakukan proses pengurutan, algoritma pengurutan dapat dibagi menjadi 2 bagian, yaitu:

1. Algoritma pengurutan yang memiliki kompleksitas O(n^2), terdiri atas Bubble Sort, Insertion Sort, Selection Sort dan Shell Sort.
2. Algoritma pengurutan yang memiliki kompleksitas O(n log n), terdiri atas Heap Sort, Merge Sort dan Quick Sort. Algoritma pengurutan ini lebih cepat dibandingkan algoritma pengurutan dengan kompleksitas O(n^2).

III. Hasil dan Pembahasan

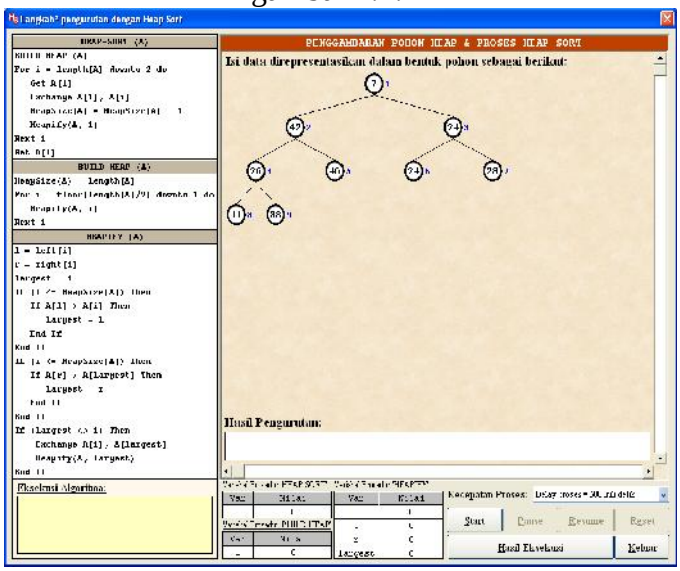
3.1 Pengujian Program

Sebagai contoh pengujian program, misalkan *input* barisan data yang akan diurutkan = 7, 42, 24, 26, 46, 24, 28, 41, 38. 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. Tampilan Form Main dapat dilihat pada gambar 3.1.



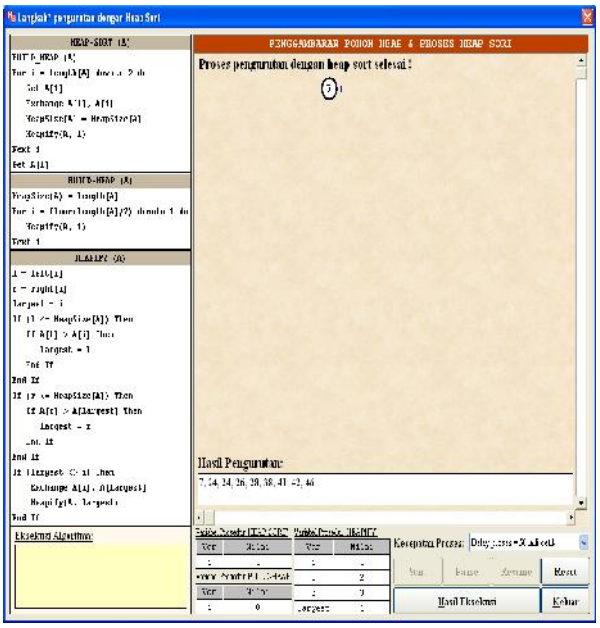
Gambar 3.1 Tampilan Form Main dengan data = 7,42,24,26,46,24,28,41,38.

Tampilan Form Pengurutan dapat dilihat pada gambar 4.2.



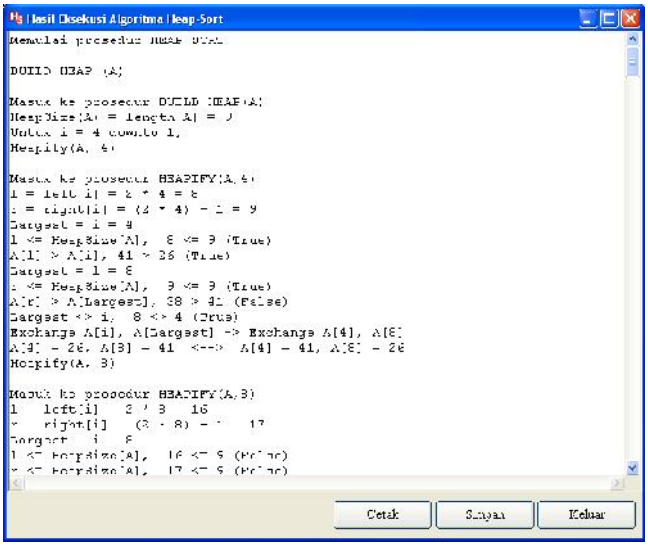
Gambar 3.2 Tampilan Form Pengurutan dengan data = 7,42,24,26,46,24,28,41,38.

Tampilan Form Pengurutan setelah proses pengurutan dapat dilihat pada gambar 4.3.



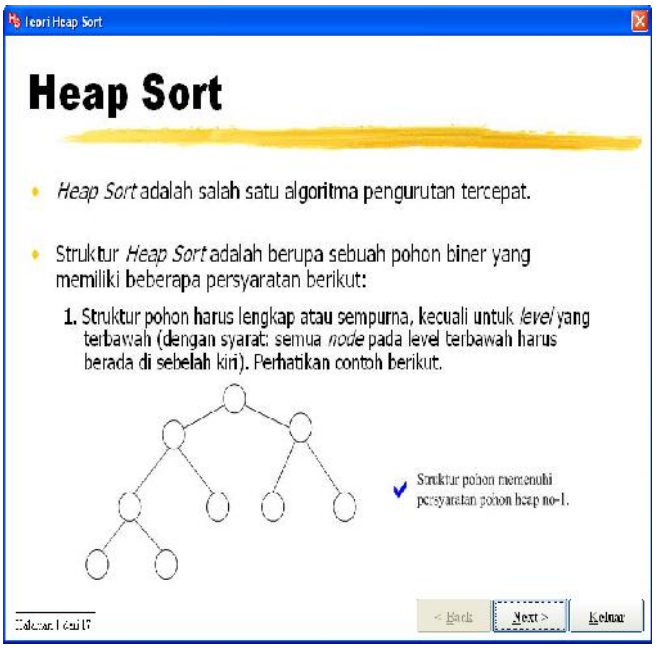
Gambar 4.3 Tampilan *Form* Pengurutan setelah proses pengurutan dengan data = 7,42,24,26,46,24,28,41,38.

Tampilan hasil eksekusi pada *form* 'Hasil' dapat dilihat pada gambar 4.4 berikut.



Gambar 4.4 Tampilan *Form* Hasil Eksekusi dengan data = 7,42,24,26,46,24,28,41,38.

Tampilan teori pada perangkat lunak seperti terlihat pada gambar 4.5 berikut.



Gambar 4.5 Contoh Tampilan *Form* Teori

IV. KESIMPULAN DAN SARAN

4.1 Kesimpulan

Setelah menyelesaikan perangkat lunak bantu pemahaman Heap Sort, penulis menarik kesimpulan sebagai berikut:

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. Algoritma pengurutan Heap Sort merupakan salah satu metode pengurutan tercepat setelah Merge Sort dan Quick Sort dengan kompleksitas O(n log n).
3. Kompleksitas prosedur Build-Heap adalah O(n), kompleksitas prosedur Heapify adalah

O(log *n*), sehingga kompleksitas algoritma Heap Sort adalah O(*n* log *n*).

4.2 Saran

Penulis ingin memberikan beberapa saran yang mungkin dapat membantu dalam pengembangan perangkat lunak ini yaitu :

1. Algoritma pengurutan Heap Sort yang terdapat dalam perangkat lunak dapat dikembangkan dengan *class object oriented* sehingga dapat dipisahkan secara independen dan dapat digunakan untuk membangun perangkat lunak lain yang membutuhkan algoritma pengurutan.
2. Penggambaran proses-proses yang terjadi pada pohon biner dapat ditingkatkan dengan menambahkan kualitas animasi yang lebih baik. Animasi yang baik bisa didapatkan dengan aplikasi Macromedia Flash.

DAFTAR PUSTAKA

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

Bambang Hariyanto, Struktur Data : Memuat Dasar Pengembangan Orientasi Objek, Edisi Kedua, Informatika Bandung, April 2003.

Thomas H.Cormen, Charles.E.Leiserson dan Ronald L.Riverst, Introduction to Algorithms, Mc GRAW-HILL, 1990.

Ralph Rinaldi Munir, Leoni Lidia, Algoritma dan Pemrograman, Edisi Kedua, 2002.

Agung Novian, Panduan MS. Visual Basic 6, Andi, Yogyakarta, 2004.

Arief Ramadhan, 36 Jam Belajar Komputer Visual Basic 6.0, PT. Elex Media Komputindo, Jakarta, 2004.

www-cse.uta.edu/~holder/courses/ cse2320/lectures/applets/sort1/heapsort.html.

www.webopedia.com/TERM/H/heap_sort.html.

www.cs.auckland.ac.nz/software/AlgAnim/heapsort.html.

http://www.stanford.edu/~rashimisu/Sorting.pdf.