Generated by Foxit PDF Creator © Foxit Software
http://www.foxitsoftware.com For evaluation only.

Algoritma Pengurutan Dalam Pemrograman

Muhammad Oky Erzandi - 13507098
Jurusan Teknik Informatika Institut Teknologi Bandung
Email : if17098@students.if.itb.ac.id

Abstract *– Makalah ini menjabarkan tentang algoritma pengurutan yang umum digunakan di dalam dunia informatika. Mulai dari yang sederhana hingga yang kompleks. Algoritma pengurutan member manfaat yang sangat banyak di dunia informatika. Algoritma pengurutan adalah algoritma untuk menyimpan sebuah list tertentu pada suatu urutan tertentu, biasanya membesar atau mengecil. Umumnya digunakan untuk mengurutkan huruf atau angka. Pada makalah ini akan diulas mengenai pengurutan, algoritma, teknik, dan efisiensinya. Dalam suatu algoritma perlu diperhatikan efisiensinya. Karena efisiensi mampu mengoptimalkan algoritma-algoritma yang lain yang akan digunakan. Semakin efisien suatu algoritma, maka pada saat dieksekusi dan dijalankan akan menghabiskan waktu yang lebih cepat. Karena itu efisiensi algoritma termasuk hal yang penting pada pemrograman. Pada algoritma pengurutan ini, masukan yang diterima adalah list tertentu yang belum pasti terurut dan keluarannya berupa list yang sudah terurut.*

Algoritma pengurutan yang akan dibahas pada makalah ini yaitu bubble sort, insertion sort, merge sort, dan quick sort. Algoritma di atas menarik untuk dibahas karena ada kelebihan dan kekurangan masing-masing sesuai tingkatan masing-masing. Makalah ini diharapkan memberi pembaca pemahaman cara-cara menggunakan algoritma sorting yang efisien. Diharapkan pembaca mampu melihat kelebihan dan kekurangan tiap algoritma secara menyeluruh.

Kata kunci *: algoritma, urut, efisiensi*

1. PENDAHULUAN

Pada saat kita membuat sebuah program sering kali kita menghadapi permasalahan yang memerlukan pengrutan suatu nilai integer baik secara langsung atau pun tidak. Misalnya kita melakukan mencari sebuah nilai pada suatu list, permasalahan akanlebih mudah diselesaikan jika kita mengurutkan terlebih dahulu list tersebut dari kecil ke besar, kita tinggal melakukan pencarian nilai tersebut selama nilai tersebut lebih kecil atau sama dengan nilai yang ditelusuri pada list. Jika nilai dari dalam list sudah lebih besar dari nilai yang kita cari berarti sudah pasti nilai yang dicari tersebut tidak ada. Ini jauh lebih efektif dibandingkan mengecek semua nilai pada list tersebut dari awal sampai akhir jika nilai itu tidak ada, ini sangat tidak efektif/ bayangkan jika kita harus mencari satu nilai dalam data yang jumlahnya mencapai jutaan atau milyaran, bahkan triliunan.

Sadar atau tidak manusia sering melakukan pengurutan dengan teknik-teknik tertentu dalam kehidupan sehari-hari. Misalnya saat kita bermain kartu remi, kita akan mengambil kartu tersebut dan mengurutkannya dengan cara-cara tertentu. Bila kita mengambil kartu tersebut satu-per-satu dari tumpukannya dan setiap mengambil kita langsung mengurutkannya dalam algoritma pengurutan, cara tersebut adalah implementasi dari insertion sort. Namun bila kartu dibagikan semuanya terlebih dahulu kemudian baru kita kelompokan menurut jenisnya. Kemdian barulah kita urutkan dari paling kecil ke paling besar maka itulah yang disebut selection sort.

Algoritma-algoritma pengurutan ini biasanya dibedakan berdasarkan:

- Kompleksitas perbandingan antar elemen (terkait dengan kasus terbaik dan terburuk) dinotasikan dengan O(n log n) untuk pencarian yang baik, dan (n²) sebagai kasus yang buruk.
- Kompleksitas pertukaran elemen, terkait dengan cara yang digunakan elemen setelah dibandingkan.
- Penggunaan memori. Ada beberapa jenis algoritma yang memerlukan memori sementara untuk menyimpan list
- Rekursif.
- Metode-metode penggunaanya, seperti *exchange, insertion, partition, merging,* dan *selection.*

2. ALGORITMA PENGURUTAN

2.1. *Bubble Sort*

Bubble sort adalah salah satu metodepengurutan exchanging yang bersifat langsung dan termasuk jenis pengurutan yang paling sederhana. Nama *bubble sort* sendiri berasal dari sifat nilai elemen terbesar yang selalu naik ke atas (ke akhir dari list) seperti gelembung udara(*bubble*). Ide dari bubble sort adalah sebagai berikut :

1. pengecekan dimulai dari elemen paling awal.
2. Elemen ke-1 dan ke-2 dari list

Generated by Foxit PDF Creator © Foxit Software
http://www.foxitsoftware.com For evaluation only.

dibandingkan.

3. Jika elemen pertama lebih besar dari elemen kedua, dilakukan pertukaran.
4. Langkah 2 dan 3 dilakukan lagi terhadap elemen kedua dan ketiga, seterusnya sampai elemen terakhir.
5. Bila sudah sampai di elemen terakhir dilakukan pengulangan lagi dari awal sampai tidak ada terjadi lagi pertukaran elemen.
6. Bila tidak ada pertukaran elemen lagi, maka elemen list terurut.

Contoh psuedocode untuk algoritma *bubble sort* dengan urutan membesar :

```
procedure bubblesort( A : list of integer )
var
  temp,i : integer
  tukar :boolean
Algoritma
do
    tukar := false
    for i = 1 to length( A ) - 1 do
            if A[ i ] > A[ i + 1 ] then
                temp:=A[i]
                A[i]:=A[i+1]
                A[i+1]:=temp
                tukar := true
            end if
    end for
while tukar
```

Pada setiap pengulangan (*loop*) dilakukan pengecekan terhadap tiap elemen mulai elemen pertama dan kedua, elemen kedua dan ketiga, dan seterusnya sampai elemen sebelum terakhir. Bila masih terjadi pertukaran (tukar = true) dilakukan pengecekan lagi sampai tidak terjadi pertukaran (tukar = false) yang berarti semua elemen dalam list tersebut sudah terurut membesar. Contoh:

5 3 8 7 9 1 awal (belum terurut)
3 5 7 8 1 9 pengulangan ke-1
3 5 7 1 8 9 pengulangan ke-2
3 5 1 7 8 9 pengulangan ke-3
3 1 5 7 8 9 pengulangan ke-4
1 3 5 7 8 9 pengulangan ke-5 (terurut)

Salah satu kelebihan algoritma *bubble sort*, terjadi saat semua elemen sudah terurut (kompleksitas = O(n)) di mana hanya terjadi pengecekan pada setiap elemen, sehingga penelusuran hanya dilakukan satu kali saja. Ini merupakan kasus terbaik yang mungkin terjadi pada algoritma ini.

Kelebihan lain dari algoritma ini adalah dapat dieksekusi dan dijalankan dengan cukup cepat dan efisien untuk sebuah kasus yang hanya mengurutkan list yang urutannya sudah hampir benar. Selain kasus terbaik tersebut, kompleksitas untuk algoritma ini akan menjadi $O(n^2)$. Karenanya algoritma ini sangat tidak efisien untuk dipergunakan dalam dunia pemrograman yang sesungguhnya, apalagi jika pengurutan dilakukan terhadap elemen yang brjumlah sangat besar.

Kelebihan lain *bubble sort* adalah kemudahan untuk dimengerti. Umumnya algoritma ini sering digunakan untuk mengenalkan algoritma pengurutan dalam dunia komputer karena kesederhanaan idenya. Namun Owen Astrachan,seorang peneliti, mengutarakan sebaiknya algoritma *bubble sort* ini tidak diajarkan lagi di dunia komputer//

Posisi setiap elemen pada bubble sort akan sangat menentukan performa saat eksekusi. Bila elemen yang terbesar disimpan di awal, maka tidak akan menimbulkan persoalan sebab elemen tersebut secara cepat akan ditukar langsung ke elemen paling terakhir. Sebaliknya jika elemen terkecil disimpan di bagian paling akhir elemen, maka akan mengakibatkan elemen tersebut akan bergerak sebanyak hanya satu pergeseran setiap masuk ke loop. Ini berarti harus dilakukan pengecekan sebanyak n kali dalam satu loop dan loop akan dijalankan sebanyak n kali juga. Kedua jenis ini biasa disebut rabbit dan turtle. Untuk menghilangkan masalah rabbit dan turtle ini, algoritma ini dikembangkan dengan menciptakan algoritma cocktail sort dan comb sort. Cocktail sort cukup baik untuk mengatasi permasalahan ini namun untuk kasus terburuk kompleksitasnya sama dengan bubble sort yaitu $O(n^2)$. Comb sort cukup baik untuk mempercepat turtle pada elemen list dan juga memiliki kompleksitas yang cukup baik, yaitu n log n, namun comb sort pun memiliki kelemahan, yaitu tidak stabil pada saat pengurutan. Kedua algoritma di atas tidak akan dibahas pada makalah ini.

Kelemahan yang lain adalah bubble sort berinteraksi dengan buruk pada computer modern saat ini. Penulisanya menghabiskan tempat dua kali lebih banyak dari insertion sort dan juga sering melakukan cache misses dan lebih banyak terjadi branch missprediction. Penelitian yang dilakukan oleh Astrachan pada pengurutan string di java juga membuktikan bahwa bubble sort lima kali lebih lambat dari insertion sort. Karenanya pada implementasinya bubble sort jarang digunakan, meskipun banyak juga algoritma lain yang dikembangkan dari bubble sort ini. Dari analisis tersebut, algoritma ini sebaiknya tidak diimplementasikan sebab

Generated by Foxit PDF Creator © Foxit Software
http://www.foxitsoftware.com For evaluation only.

termasuk tidak efisien penggunaannya, hanya baik digunakan untuk mengurutkan list yang sudah hampir terurut. Selain itu pengurutan jenis ini sangat tidak efisien dan memakan banyak waktu saat dieksekusi. Namun karena algoritma ini termasuk sederhana membuatnya cukup mudah untuk diajarkan sebagai dasar dari algoritma pengurutan.

2.2. *Insertion Sort*

Algoritma insertion sort adalah sebuah algoritma sederhana yang cukup efisien untuk mengurutkan sebuah list yang hampir terurut. Algorima ini juga biasa digunakan sebagai bagian dari algoritma yang lebih canggih. Cara kerja algoritma ini adalah dengan mengambil elemen list satu-per-satu dan memasukkannya di posisi yang benar seperti namanya. Pada array, list yang baru dan elemen sisanya dapat berbagi tempat di array, meskipun cukup rumit. Untuk menghemat memori, implementasinya menggunakan pengurutan di tempat yang membandingkan elemen saat itu dengan elemen sebelumnya yang sudah diurut, lalu menukarnya terus sampai posisinya tepat. Hal ini terus dilakukan sampai tidak ada elemen tersisa di input. Seperti sudah dibahas di bagian pendahuluan, salah satu implementasinya pada kehidupan sehari-hari adalah saat kita mengurutkan kartu remi. Kita ambil kartu satu-per-satu lalu membandingkan dengan kartu sebelumnya untuk mencari posisi yang tepat. Variasi pada umunya yang dilakukan terhadap array pada
insertion sort adalah sebagai berikut :

- Elemen awal di masukkan sembarang, lalu elemen berikutnya dimasukkan di bagian paling akhir.
- Elemen tersebut dibandingkan dengan elemen ke (x-1). Bila belum terurut posisi elemen sebelumnya digeser sekali ke kanan terus sampai elemen yang sedang diproses menemukan posisi yang tepat atau sampai elemen pertama.
- Setiap pergeseran akan mengganti nilai elemen berikutnya, namun hal ini tidak menjadi persoalan sebab elemen berikutnya sudah diproses lebih dahulu.

Sebelum *insert*

Sorted partial result		Unsorted data	
$\leq x$	$> x$	x	...

Sesudah *insert*

Sorted partial result			Unsorted data
$\leq x$	x	$> x$	...

Contoh psuedocode untuk bubble sort dengan urutan membesar :

```
procedure insertionsort(A : list of integer)
var
  Nilai,I,j : integer
Algoritma
for i = 1 to length[A]-1 do
    nilai = A[i]
    j = i-1
    while (j >= 0) and (A[j] > nilai) do
        A[j + 1] = A[j]
        j = j-1
    end while
   A[j+1] = nilai
end for
```

Pertukaran yang berulang terjadi di pengulangan *while* yang akan berhenti saat elemen sebelumnya sudah lebih kecil. Pengulangan *for* berguna untuk melakukan insert elemen selanjutnya. Kasus terbaik pada algoritma ini adalah saat semua elemen sudah terurut. Pengecekan tiap elemen hanya dilakukan 1 kali sehingga hanya terjadi n kali pengulangan *iterate* (komplesitas = O(n)). Sedangkan kasus terburuk adalah saat list ada dalam kondisi terbalik yang membutuhkan n buah pertukaran terhadap n buah elemen, sehingga kompleksitasnya sama dengan $O(n^2)$. kompleksitas ini sama dengan kompleksitas rata-ratanya. Ini berarti untuk menghitung jumlah elemen yang sangat besar algoritma ini kurang efisien untuk digunakan. Namun untuk melakukan sorting terhadap elemen yang sedikit, algoritma ini termasuk algoritma tercepat eksekusinya. Hal ini disebabkan pengulangan di dalamnya sangat cepat.

Jika kita membandingkan dengan *bubble sort*, keduanya memiliki kompleksitas yang sama untuk kasus terburuk, namun menurut Astrachan keduanya sangat berbeda dalam jumlah pertukaran yang diperlukan. Karenanya sekarang ini cukup banyak *text book* yang merekomendasikan *insertion sort* disbanding bubble sort.

Insertion sort ini memiliki beberapa keuntungan:

1. Implementasi yang sederhana
2. Paling efisien untuk data berukuran kecil
3. Merupakan online algorithmic, yang berarti bisa langsung melakukan sort setiap ada data

Generated by Foxit PDF Creator © Foxit Software
http://www.foxitsoftware.com For evaluation only.

baru

4. Proses di tempat (memerlukan O(1) memori tambahan)
5. Stabil.

Pada tahun 2004 Bender, Farach-Colton, and Mosteiro menemukan pengembangan baru dari algoritma ini, disebut library sort atau gapped insertion sort yang menggunakan beberapa gap kosong di sepanjang array. Dengan algoritma ini, pergeseran elemen dilakukan sampai gap tersebut dicapai. Algoritma ini cukup baik dengan kompleksitas O(n log n).

2.3. *Merge Sort*

Merge sort ini memanfaatkan sebuah fungsi merge dengan spesifikasi mengurutkan 2 buah list yang elemen tiap list sudah terurut. Dengan ide ini list yang akan diproses dibagi-bagi dulu menjadi list yang lebih kecil hingga tingal satu elemen. Setelah itu digabung kembali dari dua list menjadi satu, lalu digabung kembali terus sampai menjadi 2 list besar yang setelah dimerge akan menghasilkan list yang sudah terurut. Sorting jenis ini sangat berguna saat kita akan memproses jumlah elemen yang sangat banyak.

Konsep dari merge sort sendiri adalah sebagai berikut :

1. Bagi list besar menjadi setengahnya
2. Lakukan hal ini secara rekursif sampai diperoleh list dengan satu elemen saja
3. List tersebut digabung lagi menjadi sebuah list besar yang sudah terurut.

Contoh pseudocode untuk merge sort :

```
function mergesort(m)
var
    kiri, kanan, hasil :list
    tengah: integer
algoritma
if length(m)     1 then
     return m
else
    tengah = length(m) div 2
    for x = m  to tengah do
        add x to kiri
    end for
    for  x = m after tengah do
         add x to kanan
    end for
    kiri = mergesort(kiri)
    kanan = mergesort(kanan)
    hasil = merge(kiri, kanan)
end if
return hasil
```

fungsi merge sendiri pseudocodenya contohnya:

```
function merge(kiri,kanan)
var
    hasil:list
algoritma
while length(kiri) > 0 and length(kanan) > 0 do
     if first(kiri)    first(kanan) then
            append first(kiri) to hasil
            kiri = rest(kiri)
    else
            append first(kanan) to hasil
            kanan = rest(kanan)
    end if
end while
if length(kiri) > 0  then
      append rest(kiri) to hasil
end if
if length(kanan) > 0 then
    append rest(kanan) to hasil
end if
return hasil
```

Merge sort memiliki kasus terburuk dan kasus rata-rata. Kasus terburuk adalah saat tiap 2 lemen dibandingkan selalu dilakukan pertukaran. Bila waktu yang diperlukan untuk melakukan merge sort adalah T(n) maka untuk saat rekursif waktu yang dihabiskan adalah T(n) = 2T(n/2) + n. T (n/2) adalah waktu yang diperlukan untuk merge setengah dari ukuran list, dan ditambah n sebagai langkah dari penggabungan list. Kompleksitas waktu terburuk dan rata-rata dari merge sort adalah O(n log n), sama dengan kompleksitas terbaik dari quick sort. Untuk mengurutkan data yang sangat besar, jumlah perbandingan yang diharapkan mendekati nilai n di mana

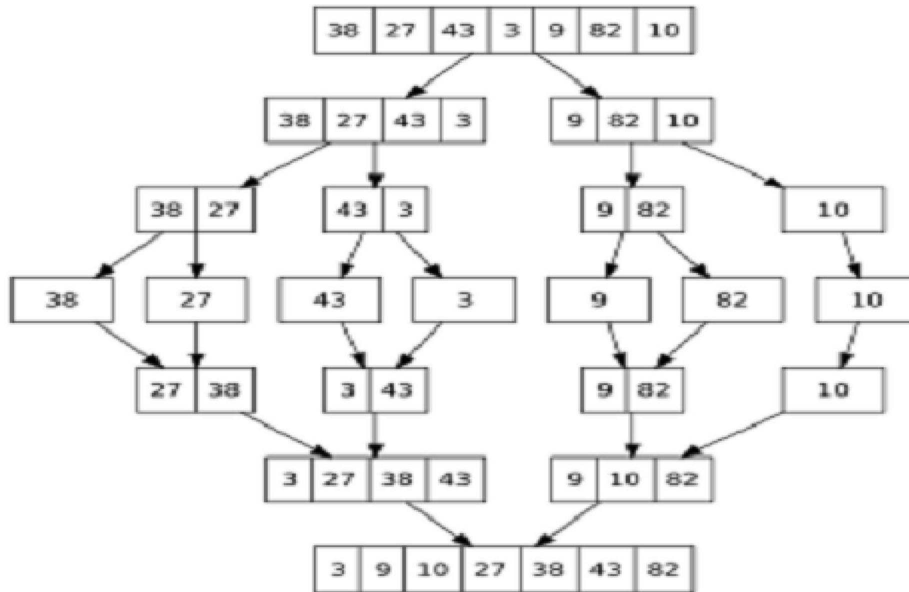
$$\alpha = -1 + \sum_{k=0}^{\infty} \frac{1}{2^k + 1} \approx 0.2645$$

Dibanding dengan algoritma lain, merge sort ini termasuk algoritma yang sangat efisien dalam penggunaannya sebab setiap list selalu dibagi-bagi menjadi list yang lebih kecil, kemudian digabungkan lagi sehingga tidak perlu melakukan banyak perbandingan. Merge sort ini merupakan algoritma terbaik untuk mengurutkan linked list, sebab hanya memerlukan memori tambahan sebesar (1).

Berdasarkan analisis tersebut, merge sort bisa dibilang sebagai salah satu algoritma terbaik terutama untuk mengurutkan data yang jumlahnya sangat banyak. Untuk data yang sedikit, algoritma ini sebaiknya tidak digunakan karena ada beberapa algoritma lain yang bisa bekerja lebih cepat dari merge sort.

Generated by Foxit PDF Creator © Foxit Software
http://www.foxitsoftware.com For evaluation only.

Ilustrasinya adalah sebagai berikut (implementasi dari merge sort terhadap 7 buah nilai):



Gambar 1. Ilustrasi algoritma pengurutan *merge sort*

2.3. *Quick Sort*

Quick sort merupakan divide and conquer algorithm. Algoritma ini mengambil salah satu elemen secara acak (biasanya dari tengah) lalu menyimpan semua elemen yang lebih kecil di sebelah kirinya dan semua elemen yang lebih besar di sebelah kanannya. Hal ini dilakukan secara rekursif terhadap elemen di sebelah kiri dan kanannya sampai semua elemen sudah terurut. Algoritma ini termasuk algoritma yang cukup baik dan cepat. Hal penting dalam algoritma ini adalah pemilihan nilai tengah yang baik sehingga tidak memperlambat proses sorting secara keseluruhan.

Ide dari algoritma ini adalah sebagai berikut :

1. Pilih satu elemen secara acak
2. Pindahkan semua elemen yang lebih kecil ke sebelah kiri elemen tersebut dan semua elemen yang lebih besar ke sebelah kanannya.
3. Elemen yang nilainya sama bisa disimpan di salah satunya. Ini disebut operasi partisi
4. Lakukan sort secara rekursif terhadap sublist sebelah kiri dan kanannya.

Berikut adalah psudocode untuk quicksort :

```
function quicksort(array)
var
   kecil,sama,besar :list
algoritma
if length(array)   1 then
   return array
end if
pivot{mengambil sebuah nilai}
   for each x in array
      if x < pivot then append x
to kecil  end if
      if x = pivot then append x
to sama end if
      if x > pivot then append x
to besar  end if
   end for
   return
concatenate(quicksort(kecil), sama ,
quicksort(besar))
```

Setiap elemen yang akan disort selalu diperlakukan secara sama di sini, diambil salah satu elemen, dibagi menjadi 3 list, lalu ketiga list tersebut disort dan digabung kembali. Contoh kode di atas menggunakan 3 buah list, yaitu yang lebih besar, sama dan lebih kecil nilainya dari pivot. Untuk membuat lebih efisien, bisa digunakan 2 buah list dengan mengeliminasi yang nilainya sama (bisa digabung ke salah satu dari 2 list yang lain).

Kasus terburuk dari algoritma ini adalah saat dibagi menjadi 2 list, satu list hanya terdiri dari 1 elemen dan yang lain terdiri dari n-2 elemen. Untuk kasus terburuk dan kasus rata-rata, algoritma ini memiliki kompleksitas sebesar O(n log n). Jumlah rata-rata perbandingan untuk quick sort berdasarkan permutasinya dengan asumsi bahwa nilai pivot diambil secara random adalah :

$$C(n) = n - 1 + \frac{1}{n}\sum_{i=0}^{n-1}\left(C(i) + C(n-i-1)\right) = 2n\ln n = 1.39n\log_2 n$$

Generated by Foxit PDF Creator © Foxit Software
http://www.foxitsoftware.com For evaluation only.

Name	Average	Worst	Memory	Stable	Method
Bubble sort	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$	$\mathcal{O}(1)$	Yes	Exchanging
Cocktail sort	—	$\mathcal{O}(n^2)$	$\mathcal{O}(1)$	Yes	Exchanging
Comb sort	—	—	$\mathcal{O}(1)$	No	Exchanging
Gnome sort	—	$\mathcal{O}(n^2)$	$\mathcal{O}(1)$	Yes	Exchanging
Selection sort	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$	$\mathcal{O}(1)$	No	Selection
Insertion sort	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$	$\mathcal{O}(1)$	Yes	Insertion
Shell sort	—	$\mathcal{O}(n\log^2 n)$	$\mathcal{O}(1)$	No	Insertion
Binary tree sort	$\mathcal{O}(n\log n)$	$\mathcal{O}(n\log n)$	$\mathcal{O}(n)$	Yes	Insertion
Library sort	$\mathcal{O}(n\log n)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n)$	Yes	Insertion
Merge sort	$\mathcal{O}(n\log n)$	$\mathcal{O}(n\log n)$	$\mathcal{O}(n)$	Yes	Merging
In-place merge sort	$\mathcal{O}(n\log n)$	$\mathcal{O}(n\log n)$	$\mathcal{O}(1)$	No	Merging
Heapsort	$\mathcal{O}(n\log n)$	$\mathcal{O}(n\log n)$	$\mathcal{O}(1)$	No	Selection
Smoothsort	—	$\mathcal{O}(n\log n)$	$\mathcal{O}(1)$	No	Selection
Quicksort	$\mathcal{O}(n\log n)$	$\mathcal{O}(n^2)$	$\mathcal{O}(\log n)$	No	Partitioning
Introsort	$\mathcal{O}(n\log n)$	$\mathcal{O}(n\log n)$	$\mathcal{O}(\log n)$	No	Hybrid
Patience sorting	—	$\mathcal{O}(n^2)$	$\mathcal{O}(n)$	No	Insertion & Selection
Strand sort	$\mathcal{O}(n\log n)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n)$	Yes	Selection

Tabel 1. Perbandingan kompleksitas berbagai algoritma pengurutan

3. KESIMPULAN

Penggunaan algoritma pengurutan dalam ilmu komputer memang sangat diperlukan sebab kita tidak bisa membuat algoritma dengan prinsip “yang penting jalan”. Bila ingin mengurutkan data yang sedikit jumlahnya maka sebaiknya menggunakan *insertion sort*. Namun bila ingin mengurutkan data yang sangat banyak, *merge sort* dan *quick sort* akan menjadi pilihan yang baik. *Bubble sort* sendiri hanya sebuah algoritma sederhana yang sebaiknya tidak diimplementasikan lagi. Masih banyak algoritma pengurutan yang lain, dengan segala kelebihan dan kekurangannya. Karena itu pemilihan kompleksitas waktu dan ruang sangat penting di sini. Makalah ini tidak membahas semua algoritma pengurutan, karena untuk membahas satu algoritma secara mendalam pun akan sangat rumit dan mungkin menghabiskan satu makalah ini. Namun melalui tulisan ini, pembaca diharapkan mampu menganalisa penggunaan *sorting algorithmic* yang baik.

DAFTAR REFERENSI

[1] Wikipedia, the free encyclopedia. (2006). Sorting algorithmic. http://en.wikipedia.org/wiki/Sorting_algorithm Tanggal akses : 2 Januari 2009 pukul 20.00.
[2] Munir, Rinaldi. (2008). Diktat Kuliah IF2093 Struktur Diskrit Edisi Keempat. Departemen Teknik Informatika, Institut Teknologi Bandung.