Studi Mengenai Perbandingan Sorting Algorithmics Dalam Pemrograman dan Kompleksitasnya

Ronny - 13506092
Jurusan Teknik Informatika Institut Teknologi Bandung
Email : if16092@students.if.itb.ac.id

1. Abstract

Makalah ini mengulas tentang algoritma sorting atau pengurutan yang biasa digunakan di dalam kehidupan yang berhubungan dengan dunia informatika. Algoritma pengurutan ini banyak kegunaannya di dunia informatika. Algoritma pengurutan adalah algoritma yang menyimpan suatu list pada suatu urutan tertentu, biasanya membesar atau mengecil, biasanya digunakan untuk mengurutkan angka ataupun huruf. Pada makalah ini akan diulas mengenai pengurutan angka, algoritma, teknik serta efisiensinya. Efisiensi pada pengurutan ini diperlukan untuk mengoptimalkan algoritma-algoritma yang lain yang akan digunakan. Semakin efisien suatu algoritma, maka pada saat dieksekusi dan dijalankan akan menghabiskan waktu yang lebih cepat dan bisa menerima lebih banyak masukan dari user. Karena itu efisiensi atau kemangkusan algoritma ini termasuk hal yang penting pada pemrograman.Pada algoritma ini, masukan yang diterima adalah list yang belum terurut(mungkin juga terurut) dan keluarannya berupa list yang sudah terurut.

Ada beberapa algoritma pengurutan yang akan dibahas pada makalah ini yaitu bubble sort, insertion sort, merge sort, dan quick sort. Setiap algoritma tersebut termasuk menarik untuk dibahas karena ada kelebihan dan kekurangan masing-masing. Melalui makalah ini diharapkan setiap pembaca mampu untuk memahami cara-cara menggunakan algoritma sorting yang efisien. Pada makalah in hasil dan pembahasan akan langsung dibahas saat algoritma tersebut diulas,

Kata kunci : *sorting ,algoritma*

2. Pendahuluan

Pada saat membuat sebuah program sering kali kita perlu untuk mengurutkan suatu list of integer yang akan digunakan untuk fungsi-fungsi lainnya. Misalnya kita ingin melakukan suatu searching nilai pada sebuah list, maka dengan mengurutkan list tersebut mulai dari kecil sampai besar, kita tinggal melakukan pencarian selama nilai tersebut lebih kecil atau sama dengan nilai yang ditelusuri pada list. Bila nilai ditelusuri sudah lebih besar dari nilai yang dicari, maka sudah bisa dipastikan bahwa nilai yang dicari tersebut tidak ada. Ini jauh lebih efektif dibandingkan menelusuri nilai-nilai pada list tersebut dari awal sampai akhir, apalagi jika kita sudah terlibat dengan data yang jumlahnya mencapai jutaan atau milyaran.

Pada kehidupan sehari-hari juga kita sering melakukan pengurutan dengan teknik-teknik tertentu. Misalnya saat kita bermain kartu remi, kita akan mengambil kartu tersebut dan mengurutkannya dengan cara-cara tertentu. Kita bisa mengurutkannya dengan mengambil kartu tersebut satu-per-satu dari tumpukannya dan setiap mengambil kita langsung mengurutkannya. Pada algoritma pengurutan, cara tersebut adalah implementasi dari insertion sort. Cara lain untuk mengurutkannya adalah dengan mengambil semua kartu tersebut dan mengurutkannya dengan menyimpan kartu paling kecil di paling kiri, dan seterusnya. Cara ini adalah implementasi dari selection sort.

Algoritma-algoritma pengurutan ini biasanya diklasifikasi berdasarkan

- Kompleksitas perbandingan antar elemen (terkait dengan kasus terbaik dan terburuk) dinotasikan dengan $O(n \log n)$ untuk pencarian yang baik, dan $\Omega(n^2)$ 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 menyimpa list
- Rekursif.
- Metode-metode penggunaanya, seperti exchange, insertion, partition, merging, dan selection.

3. Sorting Algorithmics

3.1 Bubble sort

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

- Algoritma dimulai dari elemen paling awal.
- 2 buah elemen pertama dari list dibandingkan.
- Jika elemen pertama lebih besar dari elemen kedua,dilakukan pertukaran.
- Langkah 2 dan 3 dilakukan lagi terhadap elemen kedua dan ketiga, seterusnya sampai ke ujung elemen
- Bila sudah sampai ke ujung dilakukan lagi ke awal sampai tidak ada terjadi lagi pertukaran elemen.
- Bila tidak ada pertukaran elemen lagi, maka list elemen sudah terurut.

Berikut adalah contoh psudocode untuk bubble sort dengan urutan membesar :

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

Di dalam 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 (swapped = true) dilakukan pengecekan lagi sampai tidak terjadi pertukaran (swapped = false) yang berarti semua elemen dalam list tersebut sudah terurut membesar. Contoh penerapannya :

8 6 3 7 9 1	belum terurut
6 3 7 8 1 9	setelah loop pertama
3 6 7 1 8 9	setelah loop kedua
3 6 1 7 8 9	seteleh loop ketiga
3 1 6 7 8 9	seteleh loop keempat
1 3 6 7 8 9	setelah loop kelima (terurut)

Pada algoritma ini, kasus terbaik terjadi saat semua elemen sudah terurut (kompleksitas = O(n)) di mana hanya terjadi pengecekan pada setiap elemen, sehingga penelusuran hanya dilakukan satu kali saja. Keuntungan lain dari algoritma ini adalah dapat dijalankan dengan cukup cepat dan efisien untuk mengurutkan list yang urutannya sudah hampir benar. Selain kasus terbaik tersebut, komleksitas untuk algoritma ini adalah O(n^2). Karenanya algoritma ini termasuk sangat tidak efisien untuk dilakukan, apalagi jika pengurutan dilakukan terhadap elemen yang banyak jumlahnya. Biasanya bubble sort digunakan untuk mengenalkan konsep dari sorting algoritma pada pendidikan komputer karena idenya yang cukup sederhana. Namun owen Astrachan,seorang peneliti, mengatakan bahwa sebaiknya algoritma bubble sort ini tidak diajarkan lagi di pendidikan ilmu 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 dikembankan dari bubble sort ini.

Dari analisis tersebut, algoritma ini sebaiknya tidak diimplementasikan sebab termasuk tidak efisien penggunaannya, hanya baik digunakan untuk mengurutkan list yang sudah hampir terurut. Selain itu pengurutan jenis ini sangat tidak efisien dan memakan

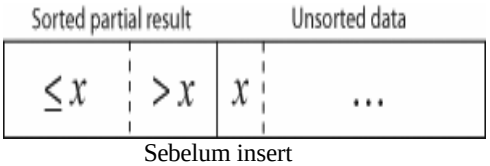
banyak waktu saat dieksekusi. Namun karena algoritma ini termasuk sederhana membuatnya cukup mudah untuk diajarkan sebagai dasar dari algoritma pengurutan.

3.2 Insertion Sort

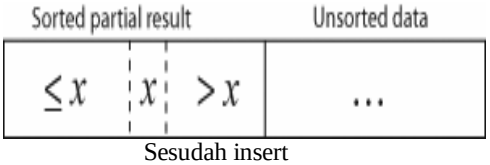
Algoritma insertion sort adalah sebuah algoritma sederhana yang cukup efisien untuk mengurutkan sebuah list yang hampir terurut. Algorima ini juga biaa digunakan sebagai bagian dari algoritma yang lebih canggih. Cara kerja algoritma ini adalah dengan mengambil elemen list satu-per-satu dan memasukkannya 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.

Ilustrasinya adalah sebagai berikut :



Sebelum insert



Sesudah insert

Berikut adalah contoh psudocode untuk bubble sort dengan urutan membesar :

```
procedure insertionSort(A : array of
integer)
    for i = 1 to length[A]-1 do
        value = A[i]
        j = i-1
        while j >= 0 and A[j] >
value do
            A[j + 1] = A[j]
            j = j-1
        end while
        A[j+1] = value
          end for
end procedure
```

Pertukaran yang berulang terjadi di loop while yang akan berhenti saat elemen sebelumnya sudah lebih kecil. Loop 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 loop iterate. 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 loop dalamnya sangat cepat.

Bila dibandingkan 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 :

- Implementasi yang sederhana
- Paling efisien untuk data berukuran kecil
- Merupakan *online algorithmic*, yang berarti bisa langsung melakukan sort setiap ada data baru
- Proses di tempat (memerlukan O(1) memori tambahan)
- Stabil.

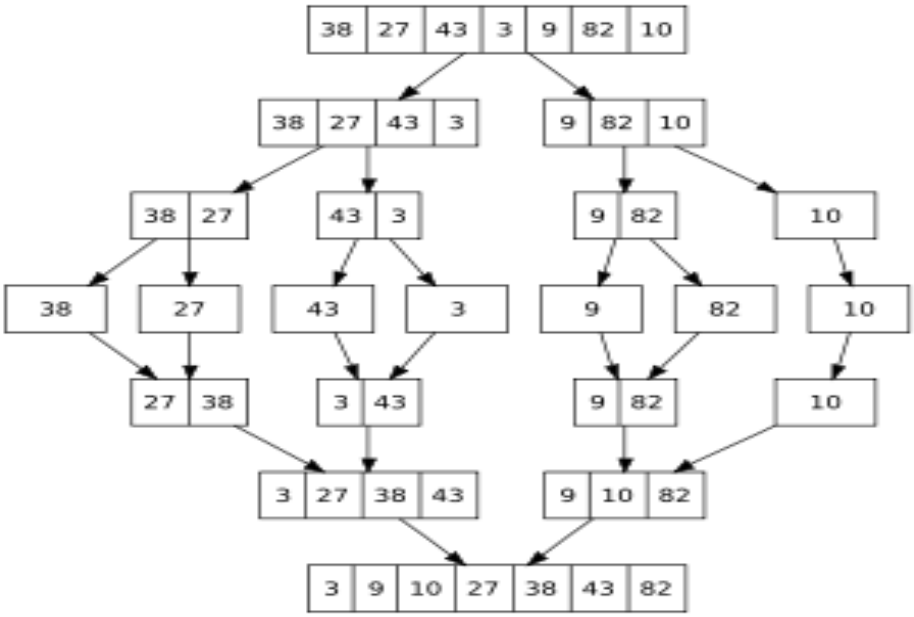
Pada tahun 2004 Bender, Farach-Colton, and Mosteiro menemukan pengembangan baru dari algoritma ini, disebut library sort atau *gapped insertion sort* yang menggunakan beberapa gap kosong di sepanjang array. Dengan algoritma ini, pergeseran elemen dilakukan sampai gap tersebut dicapai. Algoritma ini

cukup baik dengan kompleksitas O(n log n).

3.3 merge sort

Merge sort ini memanfaatkan fungsi merge yang mampu mengurutkan 2 buah list yang sudah terurut. List yang akan diproses dibagi-bagi dulu menjadi list yang lebih kecil. 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 baik untuk memproses jumlah elemen yang sangat banyak.

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



Konsep dari merge sort sendiri adalah sebagai berikut :

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

Berikut ini adalah pseudocode untuk merge sort :

```
function mergesort(m)
    var list kiri, kanan, hasil
    if length(m) ≤ 1
        return m
    else
        var tengah = length(m) / 2
        for x = m  to tengah
            add x to kiri
        end for
        for  x = m after tengah
            add x to kanan
        end for
        kiri = mergesort(kiri)
       kanan = mergesort(kanan)
       hasil = merge(kiri, kanan)
    end if
        return hasil
```

Untuk fungsi merge sendiri, ini merupakan salah satu contoh pseudocode

```
function merge(kiri,kanan)
    var list hasil
    while length(kiri) > 0 and
length(kanan) > 0
        if first(kiri) ≤
first(kanan)
            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
        append rest(kiri) to hasil
    end if
    if length(kanan) > 0
```

```
        append rest(kanan) to hasil
    end if
    return hasil
```

Merge sort memiliki kasus terburuk dan kasus rata-rata. Kasus terburuk adalah saat tiap 2 elemen 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 $\alpha \cdot n$ 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 Θ(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.

3.4. 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 :

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

Berikut adalah psudocode untuk quicksort :

```
function quicksort(array)
     var list kecil,sama,besar
     if length(array) ≤ 1
         return array
     end if
     pilih sebuah nilai, disebut
pivot
     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 :

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

Lalu bagaimana cara menentukan pivot sendiri? Kasus terbaik yang diharapkan diilustrasikan sebagai berikut :
Bagi sebuah list menjadi 4 buah. Lalu pilih 2 buah list sedemikian rupa sehingga setiap elemennya lebih besar dari 25 % elemen terkecil dan lebih kecil dari 25% elemen terbesar. Bila nilai pivot yang dipilih secara konstan terambil dari nilai ini maka hanya diperlukan pembagian list sebanyak $2\log_2 n$ kali.

Quick sort adalah pengembangan dari binary tree sort, yang lebih menghemat ruang. Cara perbandingannya mirip, hanya pengurutannya berbeda. Bila dibandingkan dengan merge sort, quick sort memiliki keuntungan di kompleksitas waktu sebesar $\Theta(\log n)$, dibanding dengan merge sort sebesar $\Theta(n)$. namun quick sort tidak mampu membandingkan *linked list* sebaik merge sort, karena ada kemungkinan pemilihan pivot yang buruk. Selain itu pada *linked list* merge sort memerlukan ruang yang lebih sedikit.

Berdasarkan analisis tersebut quick sort termasuk algoritma sorting yang cukup baik, namun kita pun harus bisa memilih nilai pivot yang baik agar penggunaannya bisa optmal.

4. Kesimpulan

Penggunaan algoritma pengurutan dalam ilmu computer 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 PUSTAKA

[1] Wikipedia, the free encyclopedia. (2006). Sorting algorithmic. *http://en.wikipedia.org*. Tanggal akses : 27 Desember 2007 pukul 10.00.

[2] *http://IlmuKomputer.com*. Tanggal Akses : 26 Desember 2006 pukul 15.04.

[3] Munir, Rinaldi. (2004). Diktat Kuliah IF2151 Matematika Diskrit Edisi Keempat. Departemen Teknik Informatika, Institut Teknologi Bandung.