Perbandingan Heap Sort dan Quick Sort Dalam Proses Pengurutan Data

Simon H S[1], Amalia Rahmah[2], Yudhie Hatmadji S[3]

Laboratorium Ilmu dan Rekayasa Komputasi
Departemen Teknik Informatika, Institut Teknologi Bandung
Jl. Ganesha 10, Bandung

E-mail : if14056@students.if.itb.ac.id[1], if14065@students.if.itb.ac.id[2], if14098@students.if.itb.ac.id[3]

Abstrak

Pengurutan data merupakan salah satu algoritma dasar yang cukup sering digunakan. Algoritma ini digunakan sebagai dasar untuk membangun sebuah aplikasi yang melibatkan pemrosesan data. Algoritma *Quick Sort* dan *Heap Sort* merupakan dua buah algoritma pengurutan data yang terbaik saat ini dengan kelebihannya masing-masing. Namun, algoritma apapun pastinya memiliki kelemahan. Dalam makalah ini, kami akan mencoba memaparkan keunggulan dan kelemahan dari algoritma *Heap Sort* dan *Quick Sort* dalam pemrosesan data-data berjumlah besar.

***Kata kunci**: heap sort, quick sort, pengurutan data, simpul, pohon biner*

1. Pendahuluan

Pengurutan data adalah algoritma yang meletakkan elemen pada sebuah list atau tabel dengan urutan tertentu. Algoritma pengurutan data saat ini telah demikian banyaknya, mulai dari yang sederhana sampai yang kompleks. Untuk saat ini, algoritma heap sort dan quicksort merupakan dua buah algoritma yang dianggap terbaik.
Penggunaan algoritma *heap sort* dan *quick sort* dalam pengurutan data berbeda. Perbedaan ini dipengaruhi oleh keunggulan dan kelemahan masing-masing algoritma pengurutan.

2. Algoritma Heap Sort dan Quick Sort

2.1 Heap Sort

Heapsort merupakan salah satu bentuk dari *selection sort* yang memiliki kompleksitas algorima O(n log(n)) yang menggunakan struktur data heap.

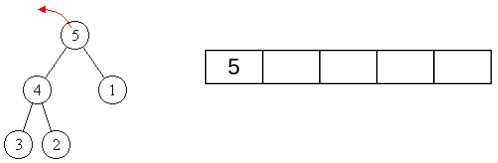
Algoritma ini bekerja dengan menentukan elemen terbesar (atau terkecil) dari sebuah daftar elemen, dan diletakkan pada akhir (atau awal) dari daftar tersebut. *Heap sort* menyelesaikan sebuah pengurutan menggunakan struktur data yang disebut *heap*.

Heap merupakan sebuah pohon biner hampir lengkap dimana isi dari simpul ayah selalu lebih besar dari isi simpul anak-anaknya sehingga simpul akar selalu merupakan elemen terbesar.

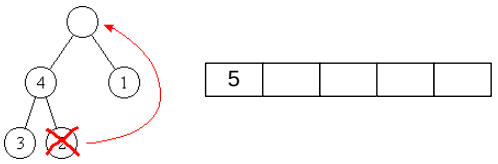
Kumpulan elemen yang akan diurutkan disimpan sebagai isi dari pohon biner. Pada implementasinya, pointer tidak dibutuhkan untuk merepresentasikan pohon, karena pohon biner yang hampir lengkap ini sudah dapat disimpan dalam sebuah tabel secara efisien.

P (pohon) dapat disebut *heap* apabila semua simpulnya memiliki properti heap.

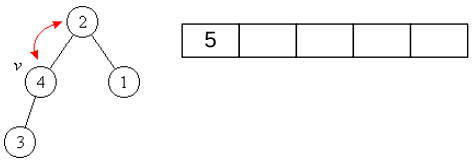
Cara kerja algoritma *heap sort* untuk pengurutan data menurun :
Bangun pohon biner yang hampir sempurna.
Deklarasikan tabel penyimpanan untuk integer.



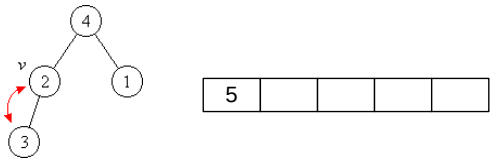
(a) Nilai dari akar dimasukkan ke ke dalam tabel penyimpanan.



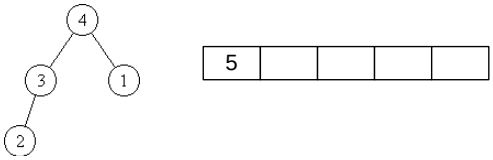
(b) Nilai daun dari level terdalam dimasukkan ke akar, hapus daun tersebut



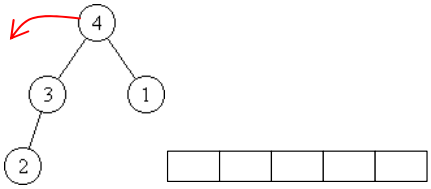
(c) Bandingkan nilai simpul akar dengan nilai simpul anak kiri dan kanannya, bila nilai akar kiri atau kanan lebih besar, maka nilai simpul akar dan nilai simpul yang lebih besar tadi ditukar.



(d) Bandingkan nilai simpul di level berikutnya dengan anak kiri dan kanannya. Bila lebih besar, tukar nilai di simpul ayah dan anak.



(e) Terbentuk pohon biner hampir lengkap kembali

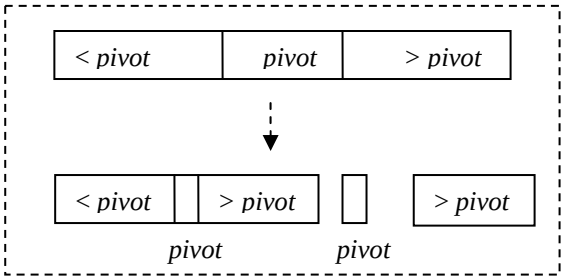


(f) Ulangi langkah (a) dan seterusnya

Gambar a-b-c-d-e: Menghapus elemen terbesar (namun tetap menjaga struktur heap) dan kemudian disimpan di-array yang baru (terurut menurun)

2.2 Quick Sort

Quick sort adalah algoritma yang menggunakan metode *divide and qonquer* yaitu dengan mempartisi tabel dengan acuan elemen tabel yang dijadikan sebagai *pivot*. Tabel dipartisi menjadi tabel dengan elemen *pivot*, < *pivot* dan > *pivot*, hingga elemen tersebut terurut.



Salah satu hal yang menentukan keoptimalan algoritma *quick sort* adalah pemilihan elemen tabel yang akan dijadika *pivot*.

Cara pemilihan *pivot* :
1. *Pivot* = elemen pertama/elemen terakhir/elemen tengah tabel
2. *Pivot* dipilih secara acak dari salah satu elemen tabel.
3. *Pivot* = elemen median tabel

Kompleksitas Algoritma:

1. *Kasus terbaik (best case)*
Kasus terbaik terjadi bila *pivot* adalah elemen median sedemikian sehingga kedua upatabel berukuran relatif sama setiap kali pempartisian.

Penyelesaian :

$$T(n) = 2T(n/2) + cn = na + \text{cn } {}^2\log n = O(n\ {}^2\log n).$$

2. *Kasus terburuk (worst case)*
Kasus ini terjadi bila pada setiap partisi *pivot* selalu

elemen maksimum (atau elemen minimum) tabel.

Kompleksitas waktu pengurutan:

Penyelesaian:

$$T(n) = T(n-1) + cn = O(n^2).$$

3. *Kasus rata-rata (average case)*
Kasus ini terjadi jika *pivot* dipilih secara acak dari elemen tabel, dan peluang setiap elemen dipilih menjadi *pivot* adalah sama.

Kompleksitas waktunya: $T_{avg}(n) = O(n\ {}^2\log n)$.

3. Perbandingan Heap Sort dan Quick Sort

Setiap algoritma pastinya memiliki kelebihan dan kekurangannya masing-masing. Dalam menganalisa kelebihan dan kekurangan dari 2 (dua) buah algoritma sorting, dapat dilakukan dengan meneliti beberapa hal berikut ini :

- kompleksitas waktu
- penggunaan memori
- kestabilan

Untuk itu kami akan mencoba membahas kedua buah algoritma ini dari ketiga segi ini.

3.1. Kompleksitas waktu

Untuk algoritma pengurutan data, kompleksitas waktu yang paling ideal adalah O(n), yaitu algoritma

pengurutan hanya dengan 1 kali pass saja. Namun sampai saat ini, belum ada algoritma yang mampu untuk mencapai keadaan ini.

Kompleksitas waktu dari kedua algoritma ini, secara rata-rata adalah sama, yaitu O(n log n). Saat ini, kompleksitas waktu O(n log n) merupakan kompleksitas algoritma sorting yang paling mendekati algoritma pengurutan ideal yaitu O(n). Hal inilah yang menyebabkan kedua buah algoritma ini dipandang sebagai yang terbaik saat ini.

Walaupun memiliki kompleksitas algoritma yang sama yaitu O(n log n), dalam praktiknya, algoritma *quicksort* memiliki kecepatan pemrosesan yang lebih baik dari *heap sort*. Penyebabnya adalah algoritma *heapsort* membutuhkan tahapan-tahapan yang lebih banyak daripada *quicksort*.

Namun untuk *quicksort*, pada kasus terburuknya memiliki kompleksitas $O(n^2)$. Sedangkan, pada *heap sort*, kompleksitas untuk kasus terburuknya sama dengan kasus rata-rata, yaitu O(n log n). Jadi pada kasus terburuk, algoritma *heap sort* lebih baik daripada algoritma *quicksort*.

Untuk lebih jelasnya, berikut ini ditampilkan tabel kompleksitas untuk kedua algoritma :

Algoritma	Terbaik	Rata-rata	Terburuk
Heapsort	O(*n*log(*n*))	—	O(*n*log(*n*))
Quicksort	O(*n*log(*n*))	O(*n*log(*n*))	O(n^2)

3.2 Penggunaan Memori

Dalam penerapannya, seringkali penggunaan algoritma pengurutan memerlukan aras (*array*) yang sangat besar. Untuk komputer dengan memori yang cukup tinggi, penggunaan aras yang besar mungkin tidak terlalu menjadi masalah. Namun, untuk komputer dengan memori yang kecil, ketidakcukupan memori yang dimilikinya haruslah ditanggung dengan menggunakan memori dari disk atau *swap space*, sehingga pemrosesannya menjadi jauh lebih lambat.

Algoritma *quicksort*, memiliki kelemahan dalam hal ini. Algoritma ini memang memiliki performa yang cukup bagus ketika memori komputernya cukup. Namun karena metodanya yang menggunakan metoda rekursif untuk mengopi sebagian atau seluruh aras, menyebabkan algoritma ini menjadi tidak efisien ketika dijalankan di dalam komputer dengan memori yang relatif kecil.

Sebaliknya, algoritma pengurutan data dengan *heap sort* justru memiliki keunggulan. Struktur data *heap*, yang secara merupakan struktur data pohon biner dimana elemen parent selalu lebih besar dari kedua anaknya, merupakan sebuah struktur data yang tidak terlalu memakan memori, karena pemrosesan datanya dilakukan bagian per bagian.

3.3 Kestabilan

Stabil yang dimaksudkan di sini memiliki pengertian yang berbeda dari yang kita ketahui sehari-hari. Algoritma pengurutan data disebut stabil apabila dalam pemrosesannya, bila sebelum diurutkan terdapat dua atau lebih data dengan nilai pembandingnya yang sama, maka setelah diurutkan data-data tersebut haruslah berada dengan urutan yang sama.

Kedua buah algoritma ini, merupakan algoritma yang **tidak stabil**. Penyebabnya adalah dalam masing-masing algoritma, terdapat suatu proses dimana seluruh elemen diubah posisinya, sehingga setelah proses tersebut terjadi, urutan awal sudah tidak diketahui lagi.

4. Kesimpulan

Dari penjelasan yang kami sebutkan di atas, kami mengambil beberapa kesimpulan sebagai berikut:

a. Kedua algoritma pengurutan ini merupakan algoritma pengurutan terbaik karena memiliki kompleksitas algoritma O(n log n) yang mendekati kompleksitas waktu ideal.

b. Untuk pengurutan data di lingkungan implementasi yang memiliki memori berkapasitas kecil, *heap sort* lebih baik karena menggunakan memori yang lebih sedikit, karena pemrosesan datanya yang dibagi-bagi, tidak seperti *quicksort* yang mengambil langsung seluruh senarai.

5. Kamus

Pohon biner hampir sempurna : pohon biner yang hanya mengharuskan nilai pada simpul ayah lebih besar dari simpul anak, peletakan pada simpul anak kiri atau kanan tidak berpengaruh.

Isi simpul atau *nilai simpul :* nilai yang ada dalam simpul berupa elemen data yang akan diurutkan.

6. Referensi

[www.wikipedia.com]

[linux.wku.edu]

[http://www.inf.fh-flensburg.de]

[Rinaldi Munir. *Diktat Kuliah Strategi Algoritmik*. Departemen teknik Informatika ITB : 2004.]