Dari hasil implementasi ini, beberapa jenis *device* (Android versi 3 ke bawah, Blackberry OS7 ke bawah, dan Windows Phone) yang secara *default* tidak di-*support* oleh layanan *catch-up* TV jadi bisa menikmati layanan *catch-up* TV.

Kelanjutan dari penelitian ini kemungkinan akan berfokus ke kualitas *video streaming* dengan *bandwidth* yang lebih efisien. Di mana kita ketahui saat ini teknologi kompresi H.265 sudah mulai berkembang untuk menggantikan H.264. Keuntungan dari teknologi H.265 adalah efisiensi penggunaan *bandwidth* yang mencapai sekitar 40% dibandingkan H.264 [14].

VI. SIMPULAN

RTSP/RTMP merupakan protokol *streaming* yang sudah dipergunakan secara luas, tetapi secara *default* tidak mendukung aplikasi *catch-up* TV. Dengan modifikasi pada *platform* internet TV, menggunakan modul livestreamrecord pada Wowza 3.5 ke atas, dan pemrograman *shell-script* pada Linux *operating system*, maka aplikasi *catch-up* TV dapat dibuat. Aplikasi *catch-up* TV pada tulisan ini sudah diimplementasikan di layanan internet TV UseeTV.

UCAPAN TERIMA KASIH

Terima kasih diucapkan kepada PT Telkom Indonesia, yang juga tempat penulis bekerja untuk mengembangkan layanan internet TV UseeTV.

DAFTAR PUSTAKA

[1] Breitman, Karin, et al. “When TV dies, will it go to the cloud?.” Computer 43.4 (2010): 81-83.

[2] PT Telkom Indonesia, http://www.useetv.com. Diakses tanggal 15 Mei 2015.

[3] Achmad Rouzni Noor, http://inet.detik.com/read/2014/09/07/145134/2683613/328/2/useetv-diharap-bisa-sebesar-netflix. Diakses tanggal 15 Mei 2015.

[4] Cambridge Advanced Learners Dictionary & Thesaurus, http://dictionary.cambridge.org/dictionary/british/*catch-up*-tv. Diakses tanggal 15 Mei 2015.

[5] Pantos, R. (30 September 2011). “HTTP Live Streaming”. Internet Engineering Task Force.

[6] Mantoro, T., M. A. Ayu, and D. Jatikusumo. “Live video streaming for mobile devices: An application on Android platform.” Uncertainty Reasoning and Knowledge Engineering (URKE), 2012 2nd International Conference on. IEEE, 2012.

[7] Schulzrinne, Henning. “Real Time Streaming Protocol (RTSP).” (1998).

[8] Parmar, Ed H., and Ed M. Thornburgh. “Real-Time Messaging Protocol (RTMP) specification.” Adobe specifications, December (2012).

[9] Zambelli, Alex. “IIS Smooth Streaming technical overview.” Microsoft Corporation 3 (2009).

[10] Jordan, Larry (10 June 2013). “The basics of HTTP Live Streaming”. Larry’s Blog. Larry Jordan & Associates.

[11] Andriescu, Emil, Roberto Speicys Cardoso, and Valérie Issarny. “AmbiStream: a middleware for multimedia streaming on heterogeneous mobile devices.” Middleware 2011. Springer Berlin Heidelberg, 2011. 249-268.

[12] Foti, George. “Method and Internet Protocol Television (IPTV) content manager *server* for IPTV servicing.” U.S. Patent No. 7,716,310. 11 May 2010.

[13] Syme, Matthew, and Philip Goldie. Optimizing network performance with content switching: server, firewall, and cache load balancing, Chapter 3. Understanding application layer protocols. Prentice Hall Professional, 2004.

[14] Grois, Dan, et al. “Performance comparison of H. 265/MPEG-HEVC, VP9, and H. 264/MPEG-AVC encoders.” PCS. Vol. 13. 2013.

Komparasi Algoritma Quicksort dan Bucket Sort pada Pengurutan Data Integer

Audy
Program Studi Teknik Informatika, Universitas Multimedia Nusantara, Tangerang, Indonesia
audytanudjaja@gmail.com

Diterima 31 Maret 2015
Disetujui 08 Mei 2015

***Abstract*—Data sorting is a technique that widely used as a part of a bigger process. Therefore, data sorting should not be the problem of program complexity. This paper gives the reader a comparison between two sorting algorithms, which are comparison based and non-comparison based, in time and space performance. The data type that used in this paper is an integer data type. Testing is carried out by using two types of data’s condition, which are the worst-case condition in each algorithm, and two amounts of data, which represent the maximum and minimum amount data.**

***Index Terms*—quicksort, bucket sort, pseudocode, cost, pivot, bucket, rekursif, integer, divide and conquer**

I. PENDAHULUAN

Perkembangan teknologi dunia sangat pesat. Hal ini terbukti dari kecepatan dan kemudahan dalam penerimaan suatu informasi. Informasi terbentuk dari hasil pemrosesan data. Di dalam buku *Reference Model for an Open Archival Information System (OAIS)* [1], data adalah suatu hal yang dapat diterjemahkan dan direpresentasikan ke dalam bentuk formal agar dapat digunakan untuk komunikasi, interpretasi, atau pengolahan informasi. Seiiring berkembangnya teknologi, perkembangan jumlah data yang dapat kita olah semakin besar sehingga dibutuhkan suatu cara untuk dapat mengolah data secara efisien dan efektif.

Pengolahan data erat kaitannya dengan pencarian data, dimana dalam pencarian tersebut terdapat proses memilah-milah data sesuai kebutuhan. Pencarian data yang efektif dan efisien tidak dapat dilepaskan dari faktor keterurutan data. Data yang sudah terurut akan mempermudah dan mempercepat pencarian data. Oleh karena itu, dibutuhkan suatu algoritma yang dapat mengurutkan data secara benar, efektif, dan efisien.

Menurut Astrachan (2003) [2], algoritma pengurutan, atau yang biasa dikenal sebagai *Sorting Algorithm*, telah muncul sejak tahun 1956. Algoritma tersebut dikenal dengan nama *Sorting by Exchange*. Seiiring berjalannya waktu, berbagai macam metode dalam algoritma pengurutan terus ditemukan sampai saat ini. Canaan dkk. (2011) [3] memberikan beberapa contoh dari algoritma pengurutan yang populer, yakni Bubble Sort, Insertion Sort, Selection Sort, Shell Sort, Merge Sort, Heapsort, Quicksort, dan Bucket Sort.

Setiap algoritma pengurutan memiliki pendekatan dan metode yang berbeda-beda dalam menjalankan fungsinya. Secara garis besar, algoritma pengurutan dapat dikelompokkan menjadi dua kategori, yaitu algoritma pengurutan berbasis perbandingan (*comparison based*) dan tidak berbasis perbandingan (*non-comparison based*). Joshi R., Panwar G.R., dan Pathak P. (2013) [4] menyatakan bahwa pada umumnya, algoritma pengurutan yang tidak berbasiskan perbandingan lebih cepat dibandingkan algoritma pengurutan yang berbasiskan perbandingan. Namun di sisi lain, dalam buku *Data Structures & Algorithms in Java* yang dikarang oleh Lafore (2003) [5] menyatakan bahwa algoritma Quicksort merupakan algoritma pengurutan tercepat secara praktis, dimana algoritma Quicksort merupakan

salah satu contoh dari algoritma yang berbasiskan perbandingan.

Penelitian sebelumnya yang dilakukan oleh Horsmalahti (2012) membahas mengenai perbandingan algoritma Bucket Sort dan Radix Sort menyatakan bahwa algoritma Bucket Sort lebih cepat dibandingkan Radix Sort pada setiap kondisi [6]. Namun, dalam hal penggunaan memori, Radix Sort membutuhkan kapasitas memori lebih kecil dibanding Bucket Sort, terutama pada pengurutan data yang berjumlah besar. P. K., Archana, dan Kumar (2014) juga melakukan penelitian mengenai analisis performa dari algoritma pengurutan, yang diantaranya adalah Bubble Sort, Merge Sort, Radix Sort, dan Quicksort. Hasil penelitian tersebut menunjukan bahwa dari keempat algoritma pengurutan tersebut, algoritma Quicksort adalah algoritma yang paling baik untuk digunakan [7].

Oleh karena pemaparan diatas, penulis ingin melakukan komparasi antara dua algoritma pengurutan, yaitu Quicksort dan Bucket sort, dalam mengurutkan data bertipe *integer* dan membandingkan performa yang dihasilkan oleh kedua algoritma tersebut, baik dari sisi waktu maupun penggunaan memori.

II. ALGORITMA PENGURUTAN

A. Definisi

Rosen (202) [8] menyatakan bahwa algoritma adalah sebuah urutan instruksi yang terbatas untuk melakukan komputasi atau untuk menyelesaikan suatu masalah. Kalimat "urutan instruksi yang terbatas" menggambarkan bahwa ada suatu nilai akhir atau *output* yang dihasilkan oleh algoritma. Dalam mencapai nilai akhir tersebut, terdapat berbagai faktor yang menjadi penentu tingkat kecepatan dan ketepatan suatu hasil akhir.

Menurut Bell dan Aspvall (2011) [9], algoritma pengurutan adalah salah satu pokok bahasan pada pembelajaran algoritma yang menyediakan kesempatan baik untuk menggambarkan analisis, kondisi terburuk, batas kompleksitas, dan teknik desain yang digunakan. Algoritma pengurutan telah banyak digunakan dalam aplikasi di dunia nyata, seperti mengurutkan data NIM mahasiswa dan mengurutkan tanggal laporan transaksi.

Algoritma pengurutan juga diimplementasikan dalam penggunaan algoritma pencarian yang efektif. Salah satu algoritma pencarian yang mengharuskan untuk menggunakan algoritma pengurutan dalam implementasinya adalah Binary Search. Tipe data yang diurutkan beragam, mulai dari bilangan bulat, bilangan bertipe *floating point*, karakter, ataupun string. Namun, tidak semua algoritma pengurutan dapat mengurutkan seluruh tipe data. Beberapa algoritma pengurutan hanya dapat digunakan dengan tipe data tertentu. Heineman, Selkow, dan Pollice (2008) [10] menyatakan salah satu contoh algoritma *non-comparison based* yang hanya cocok digunakan untuk mengurutkan data bilangan saja yaitu algoritma Bucket sort.

B. Klasifikasi Algoritma Pengurutan

Cormen, dkk. (2001) [11] menyatakan bahwa algoritma pengurutan dapat diklasifikasikan menjadi 2 kelompok, yaitu pengurutan berbasis perbandingan (*comparison based*) dan tidak berbasiskan perbandingan (*non-comparison based*). Pada umumnya, algoritma yang berbasiskan perbandingan dapat digunakan untuk mengurutkan berbagai tipe data, ataupun variasi data dengan yang bertipe sama. Contoh dari algoritma pengurutan yang berbasiskan perbandingan adalah Bubble Sort, Insertion Sort, dan Selection Sort.

Algoritma pengurutan yang tidak berbasiskan perbandingan mengharuskan terdapat suatu properti-properti tertentu pada data yang ingin diurutkan, contoh properti adalah tipe data harus suatu bilangan bulat. Hal ini menyebabkan algoritma yang tidak berbasis perbandingan menjadi lebih kaku (tidak fleksibel) dibandingkan algoritma berbasis perbandingan. Namun, keuntungan yang didapat dari penggunaan algoritma pengurutan yang tidak berbasis perbandingan adalah waktu pemrosesan data yang lebih cepat dibandingkan algoritma pengurutan yang berbasis perbandingan. Salah satu contoh algoritma pengurutan yang tidak berbasis perbandingan adalah Counting Sort.

III. ALGORITMA QUICKSORT

A. Definisi

Quicksort adalah algoritma pengurutan yang dikembangkan oleh Tony Hoare [12]. Algoritma Quicksort melakukan perbandingan sebayak $n \log n$ untuk mengurutkan data sebanyak n. kondisi paling buruk dari jumlah perbandingan adalah n^2. Secara praktis, Quicksort beroperasi lebih cepat dibanding algoritma dengan kompleksitas $n \log n$ lainnya [5].

Sareen (2013) [13] menyatakan bahwa keuntungan dari penggunaan algoritma Quicksort adalah cepat dan efisien, sedangkan kerugian yang dapat dialami adalah menghasilkan performa yang buruk apabila data sudah terurut.

B. Metode Rekursif dalam Algoritma Quicksort

Quicksort dengan pendekatan rekursif tidak membutuhkan struktur data khusus, seperti *stack*, karena setiap kelompok akan berjalan secara rekursif. Sedgewick R. dan Wayne K. (2011) [14] menyatakan bahwa terdapat dua bagian pada pendekatan rekursif, yaitu *sort* dan partisi. Partisi merupakan bagian yang melakukan tugas untuk mengelompokkan data, sedangkan *sort* adalah bagian yang melakukan proses rekursif. Semakin besar jumlah data, maka kompleksitas ruang suatu algoritma rekursif akan semakin besar.

Sedgewick R. dan Wayne K. (2011) [14] juga menyatakan bahwa terdapat tiga kasus kompleksitas waktu pada penggunaan algoritma Quicksort. Pertama adalah kasus terbaik, yaitu ketika proporsi data yang ingin diurutkan seimbang. Seimbang adalah keadaan dimana kelompok kiri dan kelompok kanan memiliki jumlah anggota kelompok yang relatif sama. Biasanya kasus ini terjadi jika memakai pivot berupa nilai median. Kasus terbaik algoritma Quicksort memiliki kompleksitas $n \log n$.

Kedua adalah kasus terburuk, yaitu ketika semua elemen, kecuali pivot, hanya berada di salah satu kelompok sehingga kelompok lainnya menjadi kosong. Hal ini akan menyebabkan terjadinya banyak pengulangan pada saat pengurutan data. Nilai pivot pada kasus ini biasanya adalah nilai maksimum atau minimum dari kelompok data yang ingin diurutkan. Kompleksitas dari kasus terburuk ini adalah $\frac{1}{2} n^2$.

Kondisi yang ketiga adalah kasus rata-rata, yaitu ketika pivot tidak terpilih secara acak dari data. Kompleksitas dari kasus rata-rata sama dengan kasus terbaikadalah $1.39\ n \log n$.

C. Pseudocode

Quicksort menggunakan metode *divide and conquer* untuk membagi suatu *list* menjadi dua *sub-list* [12]. Langkah-langkahnya adalah sebagai berikut.

1. Pilih salah satu elemen, bernama *pivot*, dari *list*.
2. Melakukan pengurutan pada *list* sehingga semua elemen yang memiliki nilai lebih kecil dari *pivot* diletakkan sebelum *pivot*, sedangkan semua elemen yang memiliki nilai lebih besar dari *pivot* diletakkan setelah *pivot*. Elemen yang memiliki kesamaan nilai dengan *pivot* dapat diletakkan sebelum ataupun sesudah *pivot*. Operasi ini dinamakan operasi partisi.
3. Melakukan pengurutan pada *sub-list* yang lebih kecil dan besar sampai *sub-list* berukuran 1. Hal ini dapat dilakukan secara rekursif maupun non-rekursif.
4. Menggabungkan *list*.

Berdasarkan hasil pemaparan langkah-langkah diatas, maka *pseudocode* dari algoritma Quicksort dengan pendekatan rekursif adalah sebagai berikut [3].

```
Function quicksort(array)
    var list less, greater

    if length(array) ≤ 1
            return array // an array of zero or
            one elements is already sorted

    select and remove a pivot value pivot
    from array

    for each x in array
            if x ≤ pivot then append x to
            less else append x to greater

    return concatenate(quicksort(less),
    pivot, quicksort(greater))
end function
```

IV. ALGORITMA BUCKET SORT

A. Definisi

Menurut Joshi, Panwar, dan Pathak (2013) [4], Bucket Sort adalah algoritma pengurutan yang membagi *n* data ke dalam berbagai ember, atau *bucket*, berdasarkan karakteristik tertentu, lalu melakukan pengurutan pada setiap ember secara parsial. Cormen, dkk. (2001) [11] menyatakan bahwa algoritma Bucket Sort mengasumsikan bahwa masukan dibuat secara acak dan didistribusikan secara seragam dengan kondisi tidak berketergantungan satu sama lain. Setiap ember dapat diurutkan dengan algoritma yang berbeda, contohnya adalah menggunakan Insertion Sort pada pengurutan tiap ember, atau dapat melakukan pengurutan dengan memanggil fungsi Bucket Sort secara rekursif.

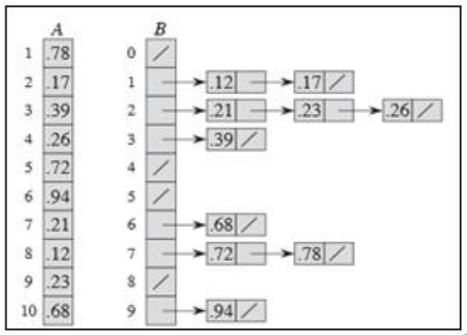
Jumlah dari ember yang dibuat sama dengan jumlah data yang diberikan sehingga Bucket Sort membutuhkan tempat atau memori yang lebih besar sebagai ganti dari pengurangan *cost* pemrosesan data. Algoritma Bucket Sort memiliki kompleksitas waktu rata-rata sebesar $O(n) + n.O(2-1/n) = O(n)$. Walaupun masukan tidak terdistribusi secara seragam, Bucket Sort akan tetap berjalan secara linear terhadap waktu selama jumlah ukuran ember sama dengan jumlah masukan.

B. Pseudocode

Langkah-langkah pada pemrosesan data dengan algoritma Bucket sort adalah sebagai berikut.

1. Membuat suatu inisial *array* (ember) kosong sebanyak jumlah *array* yang diberikan.
2. Memasukan setiap objek pada *array* yang ingin diurutkan ke dalam ember yang sesuai.
3. Melakukan pengurutan pada setiap objek yang ada di tiap ember, dapat dilakukan dengan algoritma lain, seperti Insertion Sort.

Menggabungkan setiap ember secara terurut.



Gambar 1: contoh operasi pengurutan dengan Bucket Sort dengan n = 10

Berdasarkan hasil pemaparan langkah-langkah diatas, maka *pseudocode* dari algoritma Bucket sort adalah sebagai berikut [6].

```
Function bucket-sort(array A,
    n) is n <- length[A]
        buckets ← new array of n empty lists

    for i = 1 to n do
        insert array[i] into
        buckets[n*A[i] / max_value)]
    end for

    for i = 0 to n - 1 do
        sort list buckets[i] with insertion
        sort, as example
    end for

    concatenate list of buckets[0],..,
    buckets[n-1] together
end function
```


V. PENGUJIAN

Dalam melakukan komparasi antara algoritma Quicksort dengan Bucket sort, penulis menggunakan dua jenis data. Jenis data yang pertama adalah data terurut. P. K., Archana, dan Kumar (2014) menjelaskan bahwa kondisi terburuk (*worst-case*) dari algoritma Quicksort adalah ketika *pivot* selalu membagi suatu *array* ke dalam dua bagian, dimana salah satu bagian berukuran besar (terdapat banyak data) dan bagian lain kosong, sehingga kompleksitas yang dibutuhkan menjadi $O(n^2)$ [7]. Oleh karena itu, untuk dapat menghasilkan kondisi ini, maka penulis menggunakan data yang telah terurut sehingga ketika melakukan pemilihan *pivot* dan partisi, sebelah kiri dari *pivot* selalu kosong dan sebelah kanan dari *pivot* akan selalu terisi penuh oleh data-data yang akan diurutkan sehingga jenis data ini dapat digunakan untuk melakukan analisis terhadap kondisi *worst-case* dari Quicksort.

Jenis data yang kedua adalah data yang bernilai antara 0 sampai 99, dimana ketika dilakukan pengurutan dengan algoritma Bucket sort, seluruh data akan berada di dalam ember yang sama. Cormen dkk. menjelaskan bahwa penggunaan algoritma Bucket Sort membagi interval masukan menjadi *n* sub interval yang sama dan mendistribusikan *n* masukan ke dalam ember-ember [11]. Oleh karena itu, besar dari *n* angka masukan harus terdistribusi secara seragam sehingga tidak banyak angka yang masuk pada suatu ember sehingga dapat diperoleh kondisi rata-rata (*average case*). Oleh karena itu, pada penelitian ini penulis menggunakan interval sebesar 100 sehingga jenis data kedua, yaitu data yang berada antara 0 sampai 99, akan berada pada satu ember yang sama. Hal ini akan membuat algoritma Bucket Sort berada pada kondisi terburuk (*worst-case*). Angka interval yang digunakan bukanlah suatu angka mutlak, dapat berubah sesuai kebutuhan. Namun, untuk menghasilkan kondisi terburuk, seluruh data masukan harus berada diantara angka interval yang didefinisikan. Penggunaan kedua jenis data ini bertujuan untuk melihat performa dari kondisi terburuk yang mungkin dialami algoritma Bucket Sort dan Quicksort.

Setiap jenis data terdiri dari dua jumlah data yang berbeda, yaitu berjumlah 100 data dan 3900 data. Hal ini dilakukan untuk melihat performa waktu dan penggunaan memori pada kondisi terburuk dari tiap algoritma. Penulis memilih 3900 sebagai jumlah data maksimal karena angka tersebut merupakan batas maksimum dari jumlah kedalaman suatu fungsi rekursif yang dapat ditangani oleh mesin yang penulis gunakan saat melakukan pengurutan dengan algoritma Quicksort, sedangkan angka 100 dipilih sebagai batas minimal agar dapat melihat perbedaan waktu yang dibutuhkan antar algoritma pengurutan dalam mengurutkan data.

Dalam melakukan penghitungan waktu dan besar penggunaan memori yang dibutuhkan, penulis menggunakan bahasa pemrograman C# dengan *framework* DotNet. Penghitungan waktu dilakukan dengan menggunakan *class* Stopwatch yang disisipkan sebelum dan sesudah pemanggilan fungsi pengurutan, lalu dicetak total waktu yang digunakan dengan fungsi Elapsed. Penghitungan besar memori yang digunakan dilakukan dengan *class* GarbageCollector (GC) yang disisipkan sebelum dan sesudah pemanggilan fungsi pengurutan, lalu menghitung nilai perbedaan antara besar memori sesudah dilakukan pengurutan dengan besar memori sebelum dilakukan pengurutan. Hal ini dilakukan dengan fungsi GetTotalMemory.

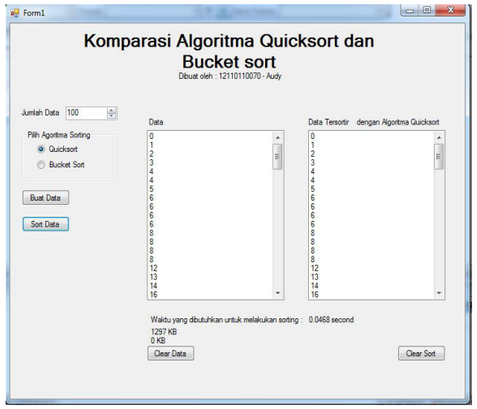
Pada data berjumlah 3900 dengan kondisi sudah terurut, Quicksort memerlukan waktu pemrosesan data sebesar 75.2176 milisecond dengan penggunaan memori sebesar 8 KB, sedangkan Bucket Sort memerlukan waktu pemrosesan data sebesar 2.2923 milisecond dengan penggunaan memori sebesar 48 KB.

Pada data berjumlah 3900 yang bernilai antara 0 sampai 99, Quicksort memerlukan waktu pemrosesan data sebesar 2.9938 milisecond dengan penggunaan memori sebesar 8 KB, sedangkan Bucket Sort memerlukan waktu pemrosesan data sebesar 4.4289 milisecond dengan penggunaan memori sebesar 32 KB.

Pada data berjumlah 100 dengan kondisi sudah terurut, Bucket Sort memerlukan waktu pemrosesan data sebesar 0.0771 milisecond

dengan penggunaan memori 0 KB, sedangkan Quicksort memerlukan waktu sebesar 0.0977 milisecond dengan penggunaan memori sebesar 0 KB.

Pada data berjumlah 100 dengan nilai antara 0 sampai 99, Bucket Sort memerlukan waktu pemrosesan data sebesar 0.0276 milisecond dengan penggunaan memori sebesar 0 KB, sedangkan Quicksort memerlukan waktu pemrosesan data sebesar 0.0459 milisecond dengan penggunaan memori sebesar 0 KB.



Gambar 2 : Contoh tampilan program pengurutan menggunakan algoritma Quicksort dengan n = 100

VI. SIMPULAN

Pada penelitian ini telah dibahas komparasi dari implementasi dari algoritma pengurutan berbasis perbandingan, yaitu Quicksort, dengan algoritma pengurutan yang tidak berbasiskan perbandingan, yaitu Bucket Sort, terhadap data bertipe bilangan atau *integer*. Analisa penggunaan waktu dan memori telah dilakukan untuk menentukan algoritma yang paling tepat digunakan dalam pengurutan data bertipe *integer*.

Kedua algoritma pengurutan, baik Quicksort dan Bucket Sort, telah dibandingkan berdasarkan kondisi terburuk dari tiap algoritma. Berdasarkan hasil pengujian, algoritma Bucket Sort lebih sesuai untuk digunakan ketika data yang ingin diurutkan berjumlah sedikit karena algoritma Bucket Sort lebih cepat dibandingkan Quicksort dan perbedaan penggunaan memori antara kedua algoritma tidak signifikan. Jika data yang ingin diurutkan berjumlah banyak, maka diperlukan pertimbangan lain dalam pemilihan algoritma pengurutan yang ingin digunakan.

Algoritma Quicksort lebih sesuai untuk digunakan pada data yang berjumlah banyak dan memiliki nilai yang terdistribusi secara acak karena proses pengurutan data lebih cepat dan penggunaan memori empat kali lebih sedikit dibandingkan dengan penggunaan algoritma Bucket Sort, sedangkan algoritma Bucket Sort akan lebih sesuai digunakan pada data yang berjumlah banyak dan tidak terurut karena proses pengurutan dapat dilakukan dengan 37 kali lebih cepat dibandingkan algoritma Quicksort. Namun, algoritma Bucket Sort membutuhkan tempat atau memori lebih besar dibandingkan algoritma Quicksort yang hanya melakukan perbandingan secara rekursif.

VII. SARAN

Penulis juga menyarankan peneliti selanjutnya untuk melakukan komparasi waktu pemrosesan dan pemakaian memori terhadap algoritma pengurutan lainnya yang dapat mengurutkan data dengan berbagai tipe, seperti *integer* dan *string*, sehingga dapat diketahui algoritma pengurutan yang paling optimal untuk digunakan secara praktis.

UCAPAN TERIMA KASIH

Penulis mengucapkan terima kasih kepada Bapak Seng Hansun, selaku dosen mata kuliah Riset Teknologi Informasi UMN yang telah membimbing penulis dalam menyusun paper ini. Penulis juga mengucapkan terima kasih kepada keluarga dan rekan-rekan mahasiswa Teknik Informatika UMN yang telah memberi dukungan dan semangat selama penelitian dan penyusunan paper ini dilakukan.

DAFTAR PUSTAKA

[1] CCDS. *Reference Model for an Open Archival Information System (OAIS)*. Washington, DC: Magenta Book Issue 2; 2012.

[2] Astrachan Owen. Bubble Sort: An Archaeological Algorithmic Analysis. *SI2GCSE '03 Proceedings of the 34th SIGCSE technical symposium on Computer science education*, 2003; Hal. 1-5.

[3] Canaan C, Garai M.S, Daya M. Popular Sorting Algorithms. *World Applied Programming Journal*, 2011; 1(1): 62-71.

[4] Joshi Rohit, Panwar Govind Singh, Pathak Preeti. Analysis of Non-Comparison Based Sorting Algorithms: A Review. *International Journal of Emerging Research in Management & Technology*, 2013; 2(12): Hal. 61-65.

[5] Robert Lafore. *Data Structures & Algorithms in Java.* USA: Sams Publishing; 2003.

[6] Horsmalahti Panu. Comparison of Bucket Sort and RADIX Sort. *arXiv, Cornell University Library*, 2012.

[7] P.K. Dixit, Archana Sahoo, Kumar S. B. Performance Analysis of Sorting Algorithm. *European Journal of Academic Essays*, 2014; Special Issue (1): Hal. 36-43.

[8] Rosen, K. H. *Discrete Mathematics and Its Applications* (7th ed.). New York: McGraw-Hill; 2012.

[9] Bell Tim, Aspvall Bengt. Sorting Algorithms as Special Cases of a Priority Queue Sort. *SIGCSE '11 Proceedings of the 42nd ACM technical symposium on Computer science education*, 2011; Hal. 123-128.

[10] Heineman George T., Selkow Stanley, Pollice Gary. *Algorithms in a Nutshell*. O'Reilly Media; 2008.

[11] Cormen Thomas H., Leiserson Charles E., Rivest Ronald L., Stein Clifford. *Introduction to Algorithms* (third edition). USA: The MIT Press; 2009.

[12] Signh Ramander, Kumar Vinod, Shrivastava A.K., Kumar Shailendra, Tiwari Amod. RVA Sorting Based On Bubble & Quick Sort Technique. *ICTCS '14 Proceedings of the 2014 International Conference on Information and Communication Technology for Competitive Strategies*, 2014; No. 87.

[13] Sareen Pankaj. Comparison of Sorting Algorithms (On the Basis of Average Case). *International Jounral of Advanced Research in Computer Science and Software Engineering*, 2013; 3 (3): Hal. 522-532.

[14] Sedgewick R., Wayne K. *Algorithms* (Fourth Edition). Addison-Wesley Professional; 2011.