Prosiding
ANNUAL RESEARCH SEMINAR 2016
6 Desember 2016, Vol 2 No. 1

ISBN : 979-587-626-0 | UNSRI http://ars.ilkom.unsri.ac.id

Perbandingan Performa Kombinasi Algoritma Pengurutan Quick-Insertion Sort dan Merge-Insertion Sort

Muhammad Ezar Al Rivan
Teknik Informatika
STMIK GI MDP
Palembang, Indonesia
meedzhar@mdp.ac.id

Abstrak—Pengurutan merupakan bagian penting dalam pengolahan data. Data yang terurut memudahkan dalam pencarian data. Algoritma pengurutan hanya cocok untuk permasalahan dengan karakteristik terntentu. Algoritma pengurutan yang cocok digunakan untuk data dalam skala besar yaitu Quick Sort dan Merge Sort namun untuk data dalam skala kecil algoritma Insertion Sort lebih cocok. Karakteristik Quick Sort dan Merge Sort yang membagi-bagi data ke dalam bagian dan setiap bagian menjadi sub-bagian maka akan didapat data dalam ukuran kecil. Proses pengurutan sub-bagian dapat digantikan dengan menggunakan Insertion Sort. Kombinasi algoritma Quick-Insertion Sort memiliki performa yang lebih baik dibandingkan dengan Quick Sort sendiri dan Merge-Insertion Sort memiliki performa yang lebih baik dibandingkan dengan Merge Sort sendiri. Quick-Insertion Sort 15% lebih cepat dibandingkan dengan Quick Sort dengan batas 16. Merge-Insertion Sort lebih cepat 34,8% lebih cepat dibandingkan dengan Merge Sort dengan batas 16 .

Keywords—Kombinasi; Quick Sort; Merge Sort; Insertion Sort;

I. LATAR BELAKANG

Perkembangan teknologi berperan dalam perkembangan informasi. Untuk menghasilkan informasi yang banyak dan beragam diperlukan pengolahan data yang efektif dan efisien. Salah satu bagian dari pengolahan data tersebut yaitu pengurutan. Algoritma pengurutan memiliki kelebihan dan kelemahan masing-masing sehingga tidak dapat digunakan untuk semua jenis permasalahan. Algoritma-algoritma tertentu hanya cocok untuk data dengan karakteristik dan ukuran tertentu.

Pengurutan merupakan bagian penting dari ilmu komputer. Pengurutan diperlukan dalam banyak aplikasi yang memerlukan data dalam bentuk terurut baik dalam bentuk *ascending* maupun dalam bentuk *descending*. Pengurutan adalah proses penyusunan elemen-elemen yang tersusun secara acak dalam kumpulan tertentu mengikuti susunan yang baru dengan urutan tertentu. Pengurutan diperlukan ketika akan melakukan pencarian. Pencarian akan lebih mudah dilakukan apabila data dalam kondisi terurut. Pencarian menggunakan parameter-parameter tertentu yang sudah diurutkan terlebih dahulu. Sebagai contoh penggunaan dalam *database*. Untuk memudahkan pencarian informasi data disimpan dalam kondisi terurut.

Algoritma pengurutan dapat dinilai berdasarkan kompleksitas yang dimiliki oleh masing-masing algoritma. Kompleksitas dapat dikelompokan menjadi tiga jenis yaitu *Best Case*, *Average Case* dan *Worst Case*. *Best Case* yaitu kondisi dimana algoritma dapat memberikan performa yang terbaik. *Average Case* yaitu kondisi dimana algoritma memberikan performa rata-rata. *Worst Case* yaitu kondisi dimana algoritma memberikan performa terburuk.

II. TINJAUAN PUSTAKA

2.1. *Insertion Sort*

Insertion Sort merupakan algoritma pengurutan yang bekerja dengan mencari posisi yang tepat untuk data dengan membandingkan dengan data yang lainnya. *Insertion Sort* memiliki kompleksitas untuk *Best Case* yaitu *O(n)* [1]. Kompleksitas untuk *Worst Case* yaitu *O(n2)* [1], [2] sedangkan untuk *Average Case* yaitu *O(n2)* [1], [2].

2.2. *Merge Sort*

Merge Sort merupakan algoritma pengurutan yang bekerja dengan cara membagi data ke dalam bagian-bagian yang kemudian setiap bagian ini akan dibagi lagi menjadi sub-bagian. Data pada setiap sub-bagian ini kemudian diurutkan lalu digabungkan dengan sub-bagian yang lain kemudian diurutakan lagi. Istilah dari proses ini kemudian dikenal dengan istilah *divide and conquer*. Kompleksitas yang dimiliki oleh *Merge Sort* untuk *Best* dan *Average Case* yaitu *O(n log n)* sedangkan *Worst Case* yaitu *O(n2)* [1]

ISBN : 979-587-626-0 | UNSRI http://ars.ilkom.unsri.ac.id

2.3. *Quick Sort*

Quick Sort merupakan algoritma yang bekerja secara *divide and conquer* sama seperti *Merge Sort*. Algoritma ini bekerja dengan mencari posisi yang tepat untuk data dengan membandingkan data dengan *pivot* sebagai data pembanding. Kompleksitas yang dimiliki oleh algoritma *Quick Sort* sama seperti *Merge Sort* karena cara kerja yang sama *divide and conquer*. Kompleksitas yang dimiliki oleh *Merge Sort* untuk *Best* dan *Average Case* yaitu *O(n log n)* sedangkan *Worst Case* yaitu *O(n2)* [1]

III. PENELITIAN TERKAIT

Penelitian mengenai perbandingan performa algoritma-algoritma pengurutan telah banyak dilakukan diantaranya dilakukan oleh [1]–[6]. Pada penelitian yang dilakukan oleh [1] algoritma-algoritma yang dibandingkan ada lima algoritma yaitu *Quick Sort, Merge Sort, Selection Sort* dan *Bubble Sort*. Berdasarkan penelitian tersebut algoritma *Quick Sort* merupakan algoritma yang memiliki performa yang terbaik untuk mengurutkan 1000 data namun jika data berjumlah 100 data algoritma yang memiliki performa terbaik yaitu algoritma *Insertion Sort.*

Pada penelitian lain, penelitian yang dilakukan oleh [4] algoritma yang dibandingkan ada enam yaitu, *Selection Sort, Insertion Sort, Merge Sort, Quick Sort, Bubble Sort* dan *Comparison Sort.* Hasil dari penelitian tersebut *Quick Sort* merupakan algoritma yang memiliki performa terbaik. Penelitian yang dilakukan oleh [2] juga membandingkan performa dari lima algoritma yaitu *Bubble Sort, Selection Sort, Insertion Sort, Merge Sort* dan *Quick Sort.* Hasil dari penelitian tersebut yaitu *Quick Sort* merupakan algoritma dengan performa terbaik dibandingkan dengan algoritma lain. Algoritma pengurutan dapat digunakan dengan menyesuaikan kondisi dari data yang akan diurutkan. Jika data dalam skala kecil algoritma yang dapat digunakan yaitu *Insertion Sort* dan *Selection Sort*. Jika data memiliki pola dan aturan tertentu dapat menggunakan *Bubble Sort* dan *Insertion Sort*. Jika data dalam skala besar maka *Quick Sort* dan *Merge Sort* dapat digunakan [2].

Penelitian yang dilakukan oleh [6] yaitu dengan membandingkan algoritma *Insertion Sort* dan *Merge Sort* secara *head to head*. Data yang digunakan dalam skala kecil. Dari hasil penelitian untuk data dibawah 100, *Insertion Sort* memiliki performa yang lebih baik dibandingkan dengan *Merge Sort* namun ketika data diatas 100 maka *Merge Sort* menjadi algoritma yang memiliki performa yang mengungguli *Insertion Sort.*

Penelitian lain yang membandingkan *Insertion Sort* dan *Merge Sort* dilakukan oleh [5]. Penelitian dilakukan dengan mengukur performa berdasarkan jumlah data yang diurutkan. Perbedaan dengan penelitian yang dilakukan oleh [6] yaitu pada jumlah data. Data yang diurutkan dalam skala besar. Hasil dari penelitian ini *Merge Sort* memiliki performa yang lebih baik. Berdasarkan kesimpulan dari [2] algoritma yang tepat untuk mengurutkan data dengan skala besar dapat menggunakan algoritma *Quick Sort* dan *Merge Sort* sesuai dengan penelitian dilakukan oleh [5]

Penelitian yang dilakukan oleh [3] yaitu dengan membandingkan algoritma *Insertion Sort* dan *Quick Sort* secara *head to head*. Data yang digunakan dalam skala kecil. Dari hasil penelitian, ketika data berada di antara 10 dan 100, *Insertion Sort* dan *Quick Sort* memiliki performa yang relatif sama namun ketika data berada diatas 100, *Quick Sort* menjadi algoritma yang terbaik. Berdasarkan kesimpulan dari [2] bahwa algoritma yang tepat digunakan untuk mengurutkan data dengan skala kecil yaitu *Insertion Sort* dan *Selection Sort* sesuai dengan penelitian yang dilakukan oleh [3] dan [6].

Berbagai cara dilakukan untuk meningkatkan performa algoritma pengurutan. Untuk meningkatkan performa dapat dilakukan dengan melakukan mengoptimasi algoritma yang sudah ada seperti pada penelitian yang dilakukan oleh [7]. Optimasi dilakukan terhadap algoritma *Bubble Sort* dan *Selection Sort.* Optimasi dilakukan dengan cara mengurangi jumlah *swap*. Penelitian yang dilakukan oleh [8] yaitu melakukan optimasi terhadap algoritma *Selection Sort* dan *Bubble Sort.* Optimasi dilakukan dengan cara menghindari operasi perbandingan data dengan data yang lain. Penelitian yang dilakukan oleh [9] dengan cara mengoptimasi algoritma *Insertion Sort.* Optimasi dilakukan dengan mengurangi pencarian posisi yang tepat untuk data dan mengurangi perpindahan data ke posisi yang tepat.

IV. METODE YANG DIUSULKAN

Pada penelitian ini, kombinasi algoritma pengurutan digunakan untuk mengurutkan data. Algoritma yang dikombinasikan yaitu antara *Merge Sort* dan *Insertion Sort* lalu *Quick Sort* dan *Insertion Sort*. Kombinasi algoritma dilakukan untuk meningkatkan performa algoritma pengurutan. Kombinasi dilakukan dengan mengambil kelebihan masing-masing algoritma. Pada kombinasi pertama yaitu *Merge Sort* dan *Insertion Sort*. Pada tahap awal algoritma yang digunakan yaitu *Merge Sort.* Pada algoritma *Merge Sort* data akan dibagi menjadi beberapa bagian kemudian setiap bagian dibagi lagi menjadi sub-bagian. Jika kondisi banyak data pada sub-bagian kurang dari atau sama dengan batas maka *Insertion Sort* akan digunakan untuk melakukan pengurutan. Adapun *pseudocode* dari *Merge-Insertion Sort* sebagai berikut :

ISBN : 979-587-626-0 | UNSRI http://ars.ilkom.unsri.ac.id

```
procedure mergeSort(input A[] :
array of integer,input p,r :
integer)
q : integer
if p < r then
      if(r - p + 1) > batas then
            q <- (p+r)/2
            mergeSort(A, p, q)
            mergeSort(A, q+1, r)
            merge(A, p, q, r)
      else
            insertionSort(A,p,r)
      endif
endif
```

Kombinasi kedua yaitu *Quick Sort* dan *Insertion Sort*. Sama seperti kombinasi pertama, pada tahap awal algoritma yang digunakan yaitu *Quick Sort*. Pada *Quick Sort* data akan dibagi-bagi menjadi beberapa bagian kemudian setiap bagian akan dibagi-bagi lagi menjadi beberapa sub-bagian. Jika kondisi banyak data kurang dari atau sama dengan batas maka *Insertion Sort* akan digunakan untuk melakukan pengurutan. Adapun *pseudocode* dari *Quick-Insertion Sort* sebagai berikut :

```
procedure quickSort(input A[] :
array of integer, input p,r :
integer)
q : integer
if p < r then
      if(r - p + 1) > batas then
       q<-partition(A,p,r)
            quickSort(A,p, q-1)
            quickSort(A,q+1, r)
     else
            insertionSort(A,p,r)
      endif
endif
```

Data yang digunakan akan di-*generate* secara acak. Untuk menjaga data agar dalam kondisi yang sama untuk digunakan dalam eksperimen maka data akan disimpan ke dalam *file*. Setiap eksperimen akan menggunakan data yang sama. Data akan diacak sebanyak yang diinginkan. Di dalam penelitian ini banyak data yaitu 100, 1000, 10000, 100000 dan 1000000 data.

Data yang diacak dipastikan tidak ada pengulangan data. Data yang digunakan merupakan data angka bulat. Selain banyak data parameter lain yang digunakan yaitu nilai batas. Nilai batas yang digunakan yaitu 8, 16, 32 dan 64.

Eksperimen dilakukan dengan menggunakan komputer yang memiliki spesifikasi Sistem Operasi Windows 8.1 Professional 64 bit, prosessor Intel Core i5-4210U dan memori 4 GB. IDE yang digunakan yaitu Netbeans 8.0.2 dengan JDK 8. *Running time* untuk setiap eksperimen akan dihitung dan dicatat. Algoritma yang akan digunakan dalam eksperimen ini selain *Merge-Insertion Sort* (MIS) dan *Quick Insertion Sort* (QIS) terdapat algoritma lain yang digunakan sebagai pembanding yaitu *Quick Sort* (QS) dan *Merge Sort* (MS).

V. HASIL DAN PEMBAHASAN

Hasil dari eksperimen dapat dilihat pada tabel berikut.

TABEL I. HASIL EKSPERIMEN DENGAN NILAI BATAS 8 (DALAM NANOSEKON)

Batas = 8				
	MS	MIS	QS	QIS
100	312188	200570	415681	376764
1K	4792725	2755379	4749104	3587594
10K	14826783	14376035	15322434	14719869
100K	110222825	124812687	72292006	72406617
1M	933679335	566501564	496145977	483277425

ISBN : 979-587-626-0 | UNSRI http://ars.ilkom.unsri.ac.id

TABEL II. HASIL EKSPERIMEN DENGAN NILAI BATAS 16 (DALAM NANOSEKON)

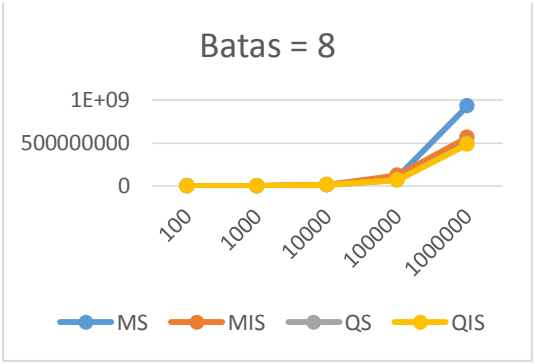
Batas = 16				
	MS	MIS	QS	QIS
100	536706	229223	472986	366073
1K	6206123	3428507	4392440	3101778
10K	16613096	13110177	17920436	18329701
100K	43509145	33038023	42476787	35234885
1M	306399080	224754697	173813771	159685348

TABEL III. HASIL EKSPERIMEN DENGAN NILAI BATAS 32 (DALAM NANOSEKON)

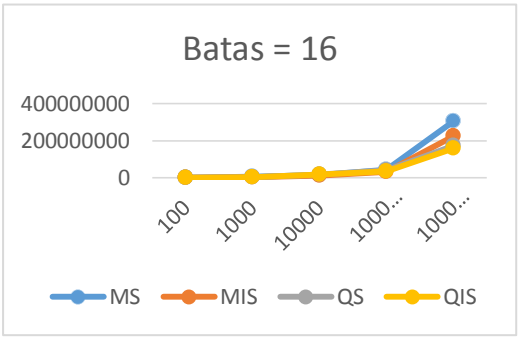
Batas = 32				
	MS	MIS	QS	QIS
100	481112	237776	534568	481112
1K	5399566	3004700	3434066	5399566
10K	35622767	22279518	22299191	35622767
100K	115834508	112167370	110707358	115834508
1M	870615258	558624595	505207550	870615258

TABEL IV. HASIL EKSPERIMEN DENGAN NILAI BATAS 64 (DALAM NANOSEKON)

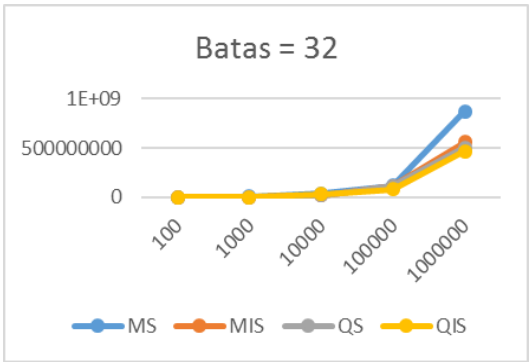
Batas = 64				
	MS	MIS	QS	QIS
100	472131	249322	456307	336564
1K	5559509	3817672	4817956	4171770
10K	19187149	24682937	17010387	21828771
100K	35728826	34395827	21487502	21872392
1M	880068989	532812650	508546676	447630708



Gambar 1. Perbandingan *running time* algoritma dengan berbagai ukuran data dengan Nilai Batas 8 (Dalam nanosekon)

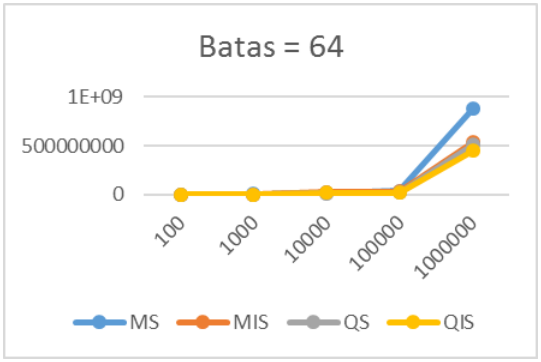


Gambar 2. Perbandingan *running time* algoritma dengan berbagai ukuran data dengan Nilai Batas 16 (Dalam nanosekon)

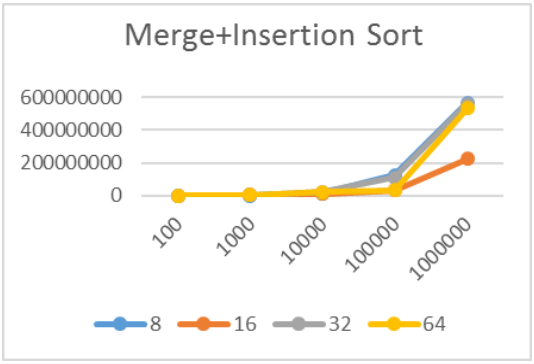


Gambar 3. Perbandingan *running time* algoritma dengan berbagai ukuran data dengan Nilai Batas 32 (Dalam nanosekon)

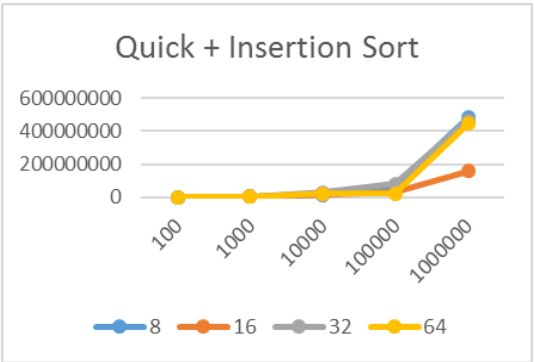
ISBN : 979-587-626-0 | UNSRI http://ars.ilkom.unsri.ac.id



Gambar 4. Perbandingan *running time* algoritma dengan berbagai ukuran data dengan Nilai Batas 64 (Dalam nanosekon)



Gambar 5. Perbandingan *Merge-Insertion Sort* dengan berbagai batas (Dalam nanosekon)



Gambar 6. Perbandingan *Quick-Insertion Sort* dengan berbagai batas (Dalam nanosekon)

Dari eksperimen yang dilakukan dapat disimpulkan, penggabungan *Quick Sort* dan *Insertion Sort* jika dibandingkan dengan *Quick Sort*, *Quick-Insertion Sort* memiliki *running time* yang lebih cepat. Secara rata-rata, untuk nilai batas = 8, *Quick-Insertion* Sort 8% lebih cepat, nilai batas = 16, 15% lebih cepat, nilai batas = 32, 5,35% lebih cepat dan nilai batas = 64, 4,3% lebih cepat.

Penggabungan *Merge Sort* dan *Insertion Sort* jika dibandingkan dengan *Merge Sort*, *Merge Insertion Sort* memiliki *running time* yang lebih cepat. Secara rata-rata, untuk nilai batas = 8, *Merge-Insertion Sort* 21,5% lebih cepat, nilai batas = 16, 34,8% lebih cepat, nilai batas = 32, 34,28% lebih cepat, dan nilai batas = 64, 18,6% lebih cepat.

Secara keseluruhan, *Quick-Insertion Sort* memiliki *running time* yang paling cepat diantara empat algoritma pengurutan yang digunakan dalam eksperimen ini. Penggunaan *Insertion Sort* di dalam *Quick Sort* memberikan dampak *running time* yang lebih cepat paling tinggi 15% untuk nilai batas = 16. Penggunaan *Merge-Insertion Sort* memberikan dampak *running time* yang lebih cepat paling tinggi 34,8% untuk nilai batas = 16. Jika dilihat dari nilai batas yang digunakan. Nilai batas = 16 memberikan dampak *running time* paling cepat dibandingkan nilai batas yang lain.

REFERENSI

[1] P. Sareen, "International Journal of Advanced Research in Computer Science and Software Engineering Comparison of Sorting Algorithms (On the Basis of Average Case)," *Int. J. Adv. Res. Comput. Sci. Softw. Eng.*, vol. 3, no. 3, pp. 522–532, 2013.

[2] Y. Yang, P. Yu, and Y. Gan, "Experimental study on the five sort algorithms," *2011 2nd Int. Conf. Mech. Autom. Control Eng. MACE 2011 - Proc.*, pp. 1314–1317, 2011.

[3] R. Hibbler, "Quick sort," *Dept. Comput. Sci. Florida Inst. Technol. Florida, USA.*, 2008.

[4] I. M. Alturani, A. Mahmoud, and I. Alturani, "Review on Sorting Algorithms A Comparative Study," *Int. J. Comput. Sci. Secur.*, vol. 7, no. 3, pp. 120–126, 2013.

[5] Arief Hendra Saptadi and D. W. Sari, "Analisis algoritma insertion sort, merge sort dan implementasinya dalam bahasa pemrograman c++," vol. 8, pp. 1–8, 2012.

[6] R. Hibbler, "Merge sort," *Dept. Comput. Sci. Florida Inst. Technol. Florida, USA.*, 2008.

[7] J. Alnihoud and R. Mansi, "An enhancement of major sorting algorithms," *Int. Arab J. Inf. Technol.*, vol. 7, no. 1, pp. 55–62, 2010.

[8] M. Khairullah, "Enhancing Worst Sorting Algorithms," *Int. J. Adv. Sci. Technol.*, vol. 56, pp. 13–26, 2013.

[9] P. Y. Padhye, "Circular Sort : A New Improved Version of Insertion Sort Algorithm," *Sch. Comput. Math. Deakin Univ. Geelong, Victoria 3217*, pp. 1–10.