Comparison of Sorting Algorithm

Merge Sort Compare with Radix Sort

Muhammad As'ad Muyassir
Departement of Electrical Engineering
University of Indonesia
Jakarta, Indonesia
muhammad.asad@ui.ac.id

***Abstract*—Pada paper ini akan dibahas tentang perbandingan antara merge sort dengan radix sort. Perbandingan tersebut akan dijelaskan dengan menggunakan perhitungan notasi big O serta hasil uji coba yang telah dilakukan. Dari data tersebut akan diolah kembali sedemikian rupa untuk mendapatkan hasil algoritma sorting manakah yang lebih baik serta lebih efisien.**

Keywords—algorithm, sorting, recursive, iterative, merge sort, radix sort, pseudocode, python, running time, efficient, big O

I. PENDAHULUAN

Algoritma sorting adalah sebuah algoritma dasar yang biasa digunakan untuk menyusun sebuah list menjadi berurutan dari urutan terendah ataupun tertinggi. Dalam aplikasinya, algoritma sorting sering digunakan pada berbagai jenis aplikasi pengolahan database, operator logika, filter data, pencarian pola, dll. Dengan pentingnya penggunaan algoritma sorting dalam aplikasi sehari-hari, maka dibutuhkan efisiensi tinggi dalam melakukan sorting supaya tidak menghambat kinerja dari algoritma utama yang ingin dijalankan. Algoritma terbagi menjadi dua metode yaitu rekursif dan iterative.

Pada paper ini akan dibahas tentang pembandingan diantara keduanya menggunakan salah satu contoh algoritma pada masing-masing metode, pada metode rekursif digunakan algoritma merge sort sedangkan pada metode iterative digunakan metode radix sort. Pembandingan tersebut dilakukan untuk mengetahui algoritma mana yang paling efektif dalam menjalankan fungsinya untuk mengurutkan list dengan waktu yang stabil pada saat diberikan input yang sedikit bahkan saat diberikan input dengan jumlah yang sangat banyak. Dari berbagai variasi input juga dapat diketahui keunggulan dan kelemahan pada setiap algoritma.

Pada tiap-tiap bab akan dibahas secara mendetail mengenai pembandingan antara merge sort dengan radix sort. Pada pembahasan di awal paper akan dimulai dengan memberikan pengetahuan kepada pembaca tentang hal-hal yang mendasari percobaan yang dilakukan, bagaimana percobaan itu dapat dilakukan, analisis menggunakan notasi big O. Kemudian akan diberikan data hasil uji coba dengan kondisi ideal saat percobaan dilakukan. Secara terperinci juga akan dijelaskan bagaimana hasil tersebut bisa didapat.

II. DASAR TEORI

A. Merge Sort

Merge sort merupakan salah satu jenis algoritma sorting yang bertipe rekursif. Hal tersebut karena pada merge sort sorting melakukan pemanggilan pada dirinya sendiri secara berulang dengan cara yang sama pada setiap list yang diberikan. List yang dapat diurutkan bisa berupa angka yang berada pada array atau gabungan huruf atau string.

Merge sort adalah salah satu algoritma sorting yang digunakan untuk mengurutkan suatu list data dengan cara membaginya menjadi dua bagian yang kemudian bagian tersebut akan dibagi menjadi dua bagian hingga hanya tersisa satu buah data pada satu bagian. Setelah bagian tersebut dipisah, maka tiap-tiap bagian akan digabungkan kembali dengan bagian lainnya dengan memperhatikan nilai pada bagian mana yang lebih besar untuk kemudian disorting setiap bagiannya hingga kembali menjadi kesatuan yang utuh dan telah tersorting.

Berikut adalah contoh gambaran langkah dengan detail setiap step dari proses merge sort yang dilakukan pada contoh array berupa [38, 27, 43, 3, 9, 82, 10].

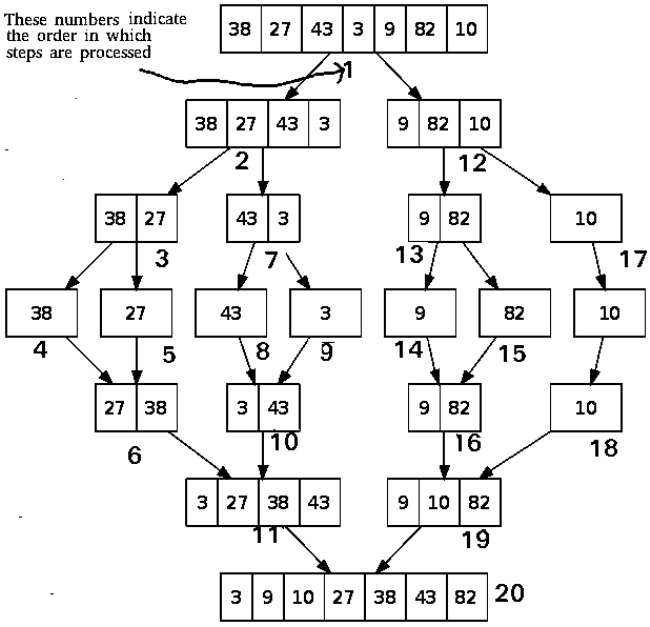

Fig. 1. Step of Merge Sort Process

Dapat dilihat dari gambar bahwa data yang awalnya terdiri dari 7 data dibagi menjadi 2 bagian terlebih dahulu dengan jumlah 4 data dan 3 data, kemudian data tersebut dibagi lagi hingga menyisakan hanya 1 buah data saya pada tiap bagiannya. Setelah itu maka data akan digabungkan dengan memperhatikan setiap nilai pada bagian lainnya supaya hasil penggabungan yang dilakukan akan menghasilkan list data baru dengan nilai yang sudah tersusun.

B. Radix Sort

Radix sort merupakan salah satu jenis algoritma sorting yang bertipe iteratif. Hal tersebut karena pada merge sort sorting dilakukan secara berulang hingga list tersebut terurut sesuai dengan banyaknya list yang diberikan. List yang dapat diurutkan biasanya berupa integer atau angka yang berada pada list array dengan jumlah yang banyak dan besar.

Radix sort adalah salah satu algoritma sorting yang digunakan untuk mengurutkan suatu list data dengan cara

membaginya menjadi 1 digit angka paling akhir yang kemudian akan diurutkan berdasarkan 1 digit angka tersebut, seletah 1 digit angka telah berurutan maka digit selanjutnya juga akan menjadi patokan dan angka akan diurutkan lagi dari awal, begitu seterusnya sampai semua digit angka telah berurutan dan angka telah selesai disorting.

Berikut adalah contoh gambaran langkah dengan detail setiap step dari proses radix sort yang akan mengurutkan 3 digit angka yang diberikan dengan jumlah yang tidak diperdulikan.

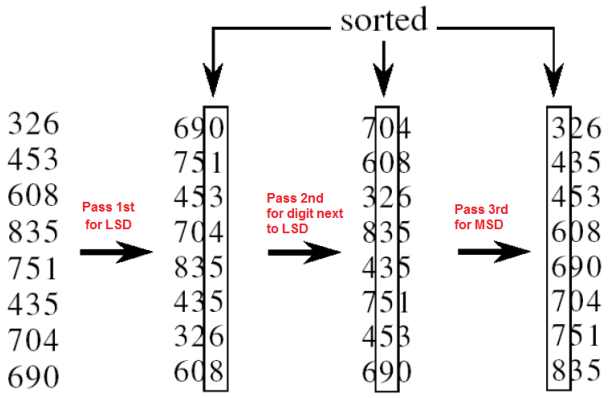

Fig. 2. Step of Radix Sort Process

Dapat dilihat dari gambar bahwa data akan diurutkan berdasarkan digit paling kanannya terlebih dahulu, setelah digit paling kanan telah tersorting maka digit selanjutnya akan disorting dan begitu seterusnya. Pada proses sorting di setiap digit angka radix sort membutuhkan bantuan dari algoritma sorting lainnya untuk mengurutkannya, karena radix hanya bertugas untuk memisahkan setiap digit angka untuk kemudian diurutkan dengan algoritma lainnya. Jadi time complexity pada algoritma radix sort juga terpengaruh oleh algoritma yang membantunya, karena n input yang ada pada algoritma radix akan menjadi n input pada algoritma yang membantunya dan membuat algoritma radix menjadi sedikit lebih kompleks dalam pengerjaannya.

C. *Pseudocode*

Pseudocode dari algoritma merge sort

```
Require : array x
1.  n ← Panjang x
2.  function MergeSort (x, p, n)
3.      if p < n do
4.          m ← (p+n)/2
5.          MergeSort (x, p, m)
6.          MergeSort (x, m+1, r)
7.          Merge (x, p, m, n)
8.      end if
9.  end function
```

Pseudocode dari algoritma radix sort

Require : array x

```
1.  function RadixSort(x)
2.      max ← nilai maksimum pada array x
3.      index ← 1
4.      while max/index > 0:
5.          countingSort(arr, index)
6.          index *= 10
7.      end while
8.  end function
```

D. *Analisis Worst Case*

Untuk pseudocode dari algoritma merge sort, dapat didefinisikan fungsi merge dengan f(n) = 2*f(n/2) + n -1 dimana fungsi tersebut dapat disederhanakan menjadi O(nlogn) untuk mendapatkan nilai dari running time pada algoritma merge sort.

Untuk pseudocode dari algoritma radix sort membutuhkan waktu dengan time complexity O(nk), namun radix sort tidak bisa berdiri sendiri karena harus mengurutkan setiap angka pada tiap digitnya, pada kasus ini digunakan counting sort untuk mengurutkannya dengan time complexity O(n + k).

III. Hasil Percobaan dan Analisis

A. *Spesifikasi Komputer yang Digunakan*

Pada paper ini, semua percobaan dilakukan pada computer dengan spesifikasi processor Intel Core I7-7700HQ dengan graphic card NVIDIA geforce GTX1050 2GB serta memory berukuran 8GB DDR4-2400MHz lalu untuk penyimpanan menggunakan harrdisk berukuran 1TB 7200RPM dengan system operasi windows 10. Uji coba dijalankan menggunakan python notebook dengan tools untuk menjalankannya menggunakan jupyter notebook yang terdapat pada anaconda3.

B. *Hasil Uji Coba*

TABLE I. TABLE OF ACTUAL RUNNING TIME

Banyak data	Actual Running Time (Microsesonds)	
	Merge Sort	Radix Sort
100	0.0	31253.34
1.000	15614.03	437427.28
10.000	46748.64	4041156.05
100.000	593496.80	42751903.30
1.000.000	7411774.87	439991469.62

Fig. 3. Table actual running time from experiment

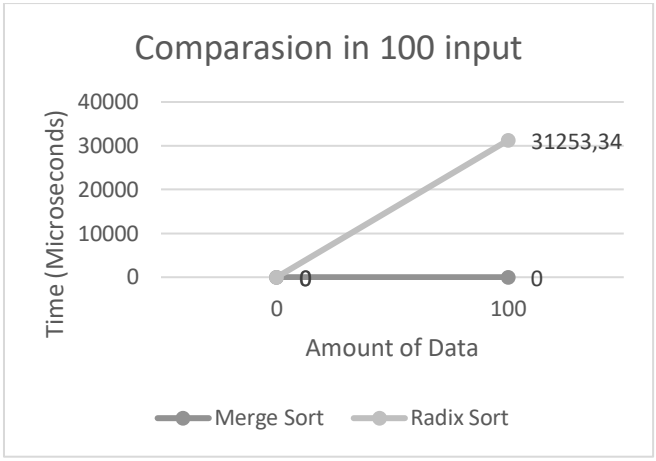

Fig. 4. Graphic comparison in 100 random input

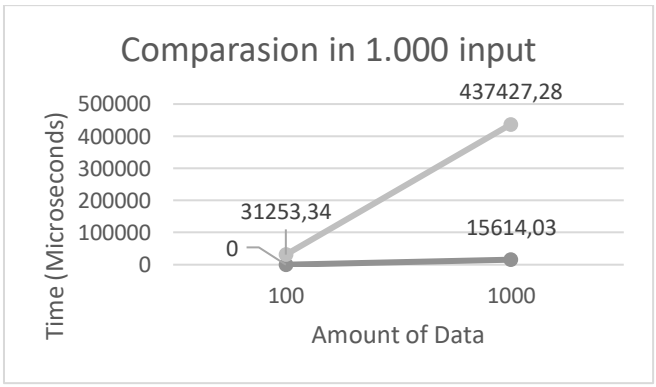

Fig. 5. Graphic comparison in 1.000 random input

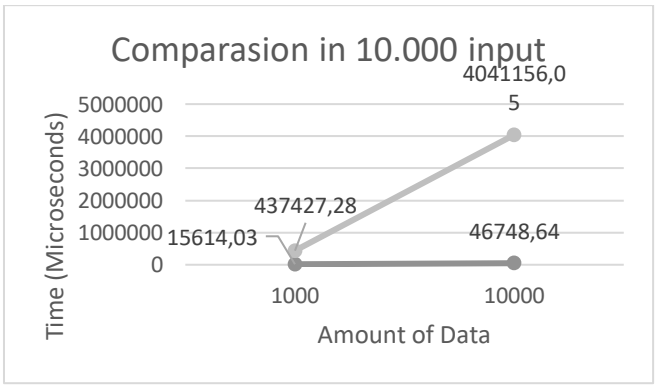

Fig. 6. Graphic comparison in 10.000 random input

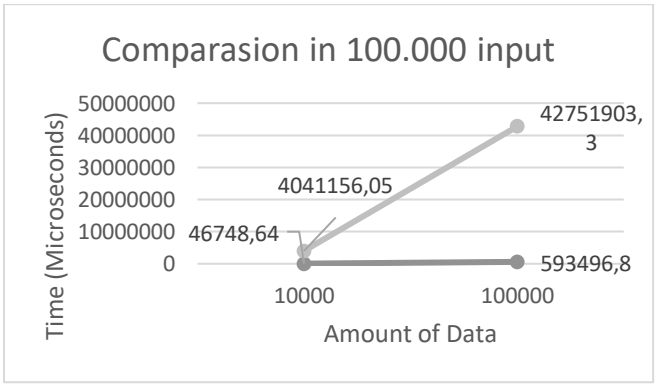

Fig. 7. Graphic comparison in 100.000 random input

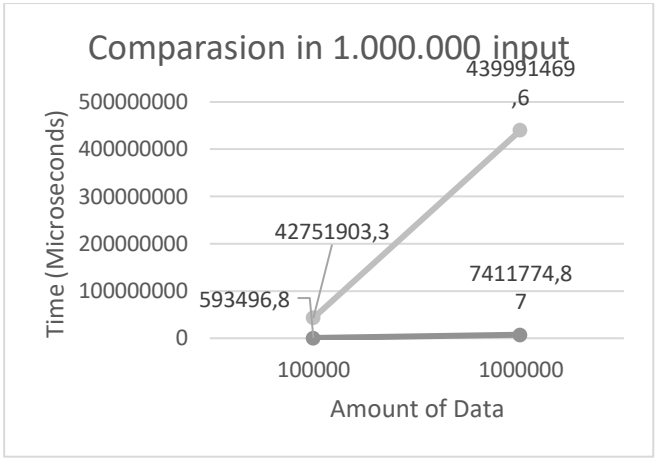

Fig. 8. Graphic comparison in 1.000.000 random input

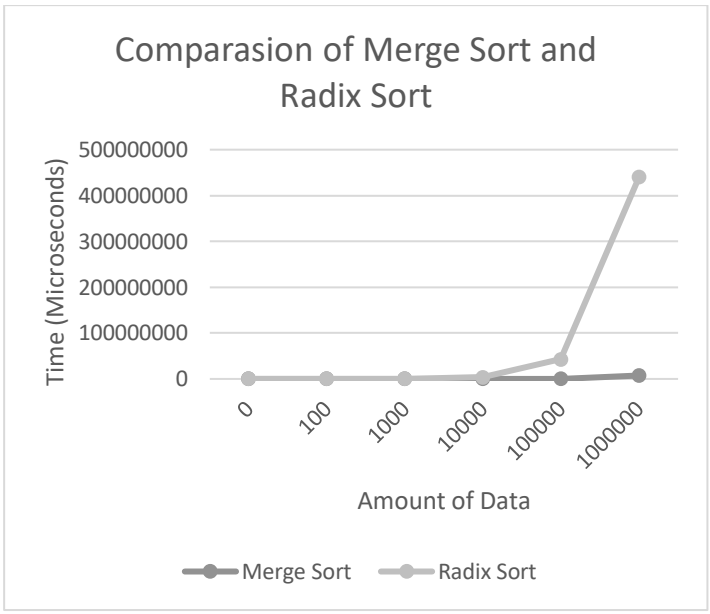

Fig. 9. Graphic actual running time from experiment

C. Analisis Hasil Eksperimen

Dari experiment yang dilakukan didapat hasil bahwa performa yang ditunjukkan oleh radix sort sangat lambat, terutama jika faktor konstanta k pada big O notation dari radix sort semakin besar. Dapat dilihat dari hasil percobaan bahwa grafik yang terjadi menunjukkan bahwa radix sangat tidak cocok jika digunakan untuk mensorting data dengan jumlah yang sangat banyak, dikarenakan juga pada radix sort membutuhkan tipe sorting lain untuk mengurutkan angka pada tiap digit bilangan, Walau seharusnya kecepatan radix sort akan konstan karena data yang diujikan berupa angka dengan range 1 – 100 saja dengan berarti pada time complexity O(nk) nilai k maksimum hanya bernilai 3, namun karena penggunaan counting sort untuk mengurutkan data pada tiap-tiap digit, maka algoritma radix sort akan menurun performanya beriringan dengan banyaknya data yang diolah. Pada radix sort juga sorting keseluruhan data dilakukan setiap digit angka, maka jika terdapat 3 digit angka maka seluruh data akan diurutkan 3 kali, maka dari itu time complexity yang didapat O(nk) karena akan melakukan perulangan sorting setiap digit angka.

Berbeda dengan algoritma merge sort yang memang memiliki time complexity tersendiri dengan O(nlogn) tanpa memerlukan bantuan sorting lainnya, dimana grafik yang akan terbentuk dari time complexity tersebut cenderung linear dengan kenaikan yang stabil, dapat dilihat juga dari hasil percobaan yang menunjukkan kenaikan waktu running time pada saat program dijalankan juga stabil tidak menunjukkan lonjakan serta memiliki running time yang rendah dibandingkan dengan radix sort. Pada merge sort waktu yang dibutuhkan hanya untuk membagi-bagi array kedalam bagian yang lebih kecil, lalu array tersebut akan disusun kembali dengan urutan yang sudah tersorting dan hanya membutuhkan waktu 2 kali pemecahan saja untuk menyelesaikan proses sortingnya sehingga running time akan konstan peningkatannya sesuai dengan banyaknya data yang diinputkan atau disorting.

IV. KESIMPULAN

Kesimpulan yang dapat diambil adalah penggunaan kedua sorting dapat digunakan untuk data dengan jumlah yang tidak begitu banyak, walau pada case ini merge sort lebih unggul dibandingkan dengan radix sort. Namun pada case data dengan jumlah lebih dari 100.000 penggunaan radix sort sangat tidak disarankan karena pada radix sort waktu yang dibutuhkan sangat lama karena sangat tergantung pada algoritma sorting lainnya, sedangkan pada merge sort memiliki running time yang rendah dan dengan kenaikan yang konstan pada jumlah data yang bervariasi. Mungkin pada algoritma radix sort bisa ditingkatkan performanya dengan mengganti algoritma pembantunya dengan algoritma yang memiliki performa lebih cepat dibandingkan dengan counting sort, namun tetap dengan performa yang belum bisa mengalahkan merge sort karena jika nilai konstanta k pada time complexity atau jumlah digit angka lebih dari satu maka performa radix sort akan menurun cukup signifikan.

REFERENCES

[1] Anonim. Sorting Algorithms. https://www.geeksforgeeks.org/sorting-algorithms/. diakses pada 22 Mei 2020.

[2] Anonim. Merge Sort. https://www.geeksforgeeks.org/merge-sort/. diakses pada 22 Mei 2020.

[3] Anonim. Radix Sort. https://www.geeksforgeeks.org/radix-sort/. diakses pada 22 Mei 2020.

[4] Anonim. Counting Sort. https://www.geeksforgeeks.org/counting-sort/. diakses pada 22 Mei 2020.

[5] Victor S.Adamchik. 2009. “Sorting” pittsburgh: Carniege Mellon University.

[6] Anonim. Time Complexities.
https://www.hackerearth.com/practice/notes/sorting-and-searching-algorithms-time-complexities-cheat-sheet/. diakses pada 22 Mei 2020

[7] Anonim. Radix Sort Pseudocode.
https://www.codingeek.com/algorithms/radix-sort-explanation-pseudocode-and-implementation/. diakses pada 23 Mei 2020

[8] A. Andersson, and S. Nillson. “A new efficient radix sort”, Lund. International Journal of Computer Science and Information Security (IJCSIS), December 2016.

[9] N. Akhter, M. Idrees, and F. Rehman, “Sorting Algorithms – A Comparative Study”, Lund. Sweden, December 1994.

[10] Th. H. Cormen, Ch. E. Leiserson, and R. L. Rivest. “Introduction to algorithms”, McGraw-Hill, 1990.