Tugas Besar Analisis Kompleksitas Algoritma

Studi Kasus Algoritma Insertion-Sort dan Bubble-Sort

IF-44-09

Kelompok 2 :

RYAN OKTAVIANDI SUSILO WIBOWO - **1301204289**

RAIHAN ATSAL HAFIZH - **1301204485**

PROGRAM STUDI S1 INFORMATIKA
FAKULTAS INFORMATIKA

2021

Daftar Isi **2**

BAB I **3**

PENDAHULUAN 3

BAB II **4**

2.1.1 Definisi Algoritma Insertion-Sort 4

2.1.2 Pseudocode Algoritma Insertion-Sort 4

2.1.3 Cara Kerja Algoritma Insertion-Sort 5

2.2.1 Definisi Algoritma Bubble-Sort 5

2.2.2 Pseudocode Algoritma Bubble-Sort 6

2.2.3 Cara Kerja Algoritma Bubble-Sort 6

BAB III **7**

3.1.2 Perhitungan Kompleksitas Waktu dan Kelas Efisiensi dari Insertion-Sort 7

3.1.2 Ilustrasi Insertion-Sort 7

3.1.3 Tabel dan Grafik Perubahan Running Time Insertion-Sort 8

3.2.1 Perhitungan Kompleksitas Waktu dan Kelas Efisiensi dari Bubble-Sort 8

3.2.2 Ilustrasi Bubble-Sort 9

3.2.3 Tabel dan Grafik Perubahan Running Time Bubble-Sort 10

3.3.1 Perbandingan Antara Insertion-sort dan Bubble-Sort 10

BAB IV **11**

4.1 Kesimpulan 11

Referensi **12**

BAB I

PENDAHULUAN

Pengurutan merupakan proses mengatur susunan data dengan syarat tertentu. Pengurutan dapat memberikan dampak efisiensi waktu saat melakukan pengelolaan data. Data yang terurut mempermudah dalam pencarian data. Dalam dunia informatika, pengurutan sangat dibutuhkan saat pengolahan data. Pengurutan data (*sorting*) adalah suatu proses pengurutan data yang tersusun secara acak pada suatu pola tertentu, sehingga tersusun secara teratur menurut aturan tertentu. pengurutan ini dapat dilakukan dengan cara *ascending* dan *descending* serta digunakan juga untuk mengurutkan data yang bertipe angka atau karakter, (Rahayuningsih,2016). Pencarian data lebih mudah dilakukan apabila data sudah dalam kondisi terurut.

Ada berbagai jenis algoritma pengurutan. Algoritma-algoritma pengurutan ini digunakan sesuai dengan permasalahan tertentu. Algoritma pengurutan yang sering digunakan antara lain, *Insertion-Sort*, *Bubble-Sort*, *Selection-Sort*, *Shell-Sort*, *Heap-Sort*, *Merge-Sort*. Algoritma pengurutan ini memiliki fungsi dan ke efesiensinan masing-masing sesuai dengan permasalahannya. Namun pada makalah laporan tugas besar ini hanya akan membahas algoritma pengurutan *Insertion-Sort* dan *Bubble-Sort*.

Untuk dapat mengetahui seberapa efisien suatu algoritma, dipakailah teori kompleksitas waktu algoritma sebagai dasar kajian. Kompleksitas Waktu, T(n), adalah jumlah operasi yang dilakukan untuk melaksanakan algoritma sebagai fungsi dari ukuran masukan n, (Tjaru, 2009). Kompleksitas waktu, mengukur dengan menghitung banyaknya operasi yang dilakukan oleh algoritma. Idealnya, memang harus menghitung semua operasi dalam suatu algoritma. Namun, akan lebih efisien cukup dengan menghitung jumlah operasi abstrak atau disebut dengan operasi dasar yang mendasari suatu algoritma.

Kompleksitas waktu untuk algoritma-algoritma yang dibahas akan dinyatakan dengan notasi O besar (*Big-O notation*). Definisi dari notasi O besar adalah, jika sebuah algoritma mempunyai waktu asimptotik O(f(n)), maka jika n dibuat semakin besar, waktu yang dibutuhkan tidak akan pernah melebihi suatu konstanta C dikali dengan f(n). Jadi f(n) adalah adalah batas atas (*upper bound*) dari T(n) untuk n yang besar. O(n) dihitung berdasarkan banyaknya jumlah operasi perbandingan yang dilakukan dalam algoritma tersebut.

BAB II

2.1.1 Definisi Algoritma Insertion-sort

Metode pengurutan *insertion-sort* merupakan pengurutan data dengan membandingkan dua elemen data pertama, kemudian membandingkan elemen-elemen data yang sudah diurutkan, perbandingan tersebut akan terus diulang hingga tidak ada elemen data yang tersisa.

Insertion-sort merupakan sebuah teknik pengurutan dengan cara membandingkan dan mengurutkan dua data pertama pada array, kemudian membandingkan data para array berikutnya apakah sudah berada di tempat semestinya. Algoritma insertion sort dapat mengurutkan data dari besar ke kecil (*Ascending*) dan kecil ke besar (*Descending*). Algoritma *insertion-sor*t ini sederhana dalam penerapannya, proses pengurutan menggunakan metode *insertion* dapat dibilang proses pengurutan yang paling cepat dan efisien.

2.1.2 Pseudocode Algoritma Insertion-sort

```
Procedure InsertionSortAscending (input/output Arr : Array, input
nMax : integer)
{I.S Terdefinisi array integer yang mungkin belum terurut}
{F.S Array akan terurut dari kecil menuju besar atau ascending}
Kamus
      i : integer
      j : integer
      temp : integer
Algoritma
      for i ← 0 to nMax do
            for j ← i to j ≥ 0 do
                  if (arr[j - 1] > arr[j] then
                        temp ← arr[j - 1]
                        arr[j - 1] ← arr[j]
                        arr[j] ← temp
                  {endif}
            j ← j - 1
            {endfor}
      i ← i + 1
      {endfor}
{endprocedure}
```

2.1.3 Cara Kerja Algoritma Insertion-sort

Cara kerja *insertion-sort*, jika suatu array memiliki panjang 10 maka kita bisa mendefinisikan elemen *array* pertama adalah 0 sampai ke n - 1 atau bisa juga dari 1 hingga ke n, dalam metode *sorting* kami akan mendefinisikan elemen *array* pertama sebagai i, dan j sebagai elemen *array* yang akan dibandingkan dengan i.

Dalam metode ini i akan memulai dari elemen pertama + 1 dan j akan dimulai dari i dan akan mundur hingga $j \geq 0$ dan akan mengecek apabila array[j - 1] < array[j] atau array[j-1] tidak lebih kecil dari array[j], apabila array[j-1] lebih kecil maka akan dilakukan pertukaran nilai antara nilai dari array[j - 1] dan array[j] dengan bantuan variabel temp (sebagai tempat penyimpanan sementara suatu nilai), setelah ditukar maka j akan berkurang 1 terus menerus hingga $j \geq 0$ dan akan melakukan pertukaran nilai lagi.

Apabila kondisi tersebut *true*, maka untuk i akan terus menerus bertambah maju hingga n - 1, untuk *array* pertama dimulai dari 0 dan akan terus bertambah maju hingga n apabila *array* pertama dimulai dari 1.

2.2.1 Definisi Algoritma Bubble-sort

Diberi Nama "*Bubble*" karena proses pengurutan algoritma ini terjadi dengan cara berangsur-angsur bergerak atau berpindah ke posisi yang tepat, seperti gelembung yang keluar dari sebuah gelas bersoda.

Bubble-sort adalah metode pengurutan algoritma dengan cara melakukan penukaran data secara terus menerus sampai bisa dipastikan dalam suatu iterasi tertentu tidak ada lagi perubahan/penukaran. Algoritma ini menggunakan perbandingan dalam operasi antar elemennya.

Secara sederhana, bisa didefinisikan bahwa algoritma *bubble-sort* adalah pengurutan dengan cara pertukaran data dengan data di sebelahnya secara terus menerus sampai pada satu iterasi tertentu dimana tidak ada lagi perubahan yang signifikan. Algoritma *bubble-sort* ini memiliki metode algoritma yang mudah dipahami dan simpel. Metode ini merupakan metode yang paling sederhana dalam mengurutkan data.

2.2.2 Pseudocode Algoritma Bubble-sort

```
Procedure BubbleSortAscending (input/output Arr : Array, input
nMax : integer)
{I.S Terdefinisi array integer yang mungkin belum terurut}
{F.S Array akan terurut dari kecil menuju besar atau ascending}
Kamus
      i : integer
      j : integer
      temp : integer
Algoritma
      for i ← 0 to i < (nMax - 1) do
            for j ← 0 to j < nMax - (i - 1) do
                  if (arr[j] > arr[j + 1]) then
                        temp ← arr[j]
                        arr[j] ← arr[j + 1]
                        arr[j] ← temp
                  {endif}
            j ← j + 1
            {endfor}
      i ← i + 1
      {endfor}
{endprocedure}
```

2.2.3 Cara Kerja Algoritma Bubble-sort

Cara kerja dari *bubble-sort*, *bubble-sor*t akan melakukan pengurutan data dengan cara membandingkan elemen yang sekarang dengan elemen berikutnya. Saat elemen yang sekarang lebih besar atau lebih kecil dari elemen berikutnya, maka kedua elemen tersebut akan ditukar sesuai dengan pengurutannya *ascending* atau *descending*.

Algoritma ini seolah menggeser satu per satu elemen sesuai dengan jenis pengurutannya. Di saat proses penukaran selesai, maka *bubble-sort* akan terus mengulangi prosesnya. *Bubble-sort* akan berhenti ketika seluruh array telah diperiksa dan tidak dapat melakukan pertukaran lagi atau bisa dikatakan saat tercapai pengurutan yang diinginkan.

BAB III

3.1.2 Perhitungan Kompleksitas Waktu dan Kelas Efisiensi dari Insertion-sort

$$T(n) = \sum_{i=0}^{nMax} \sum_{j=i}^{0} 1 = \sum_{i=0}^{nMax} [u - l + 1]$$

$$= \sum_{i=0}^{nMax} [0 - i + 1]$$

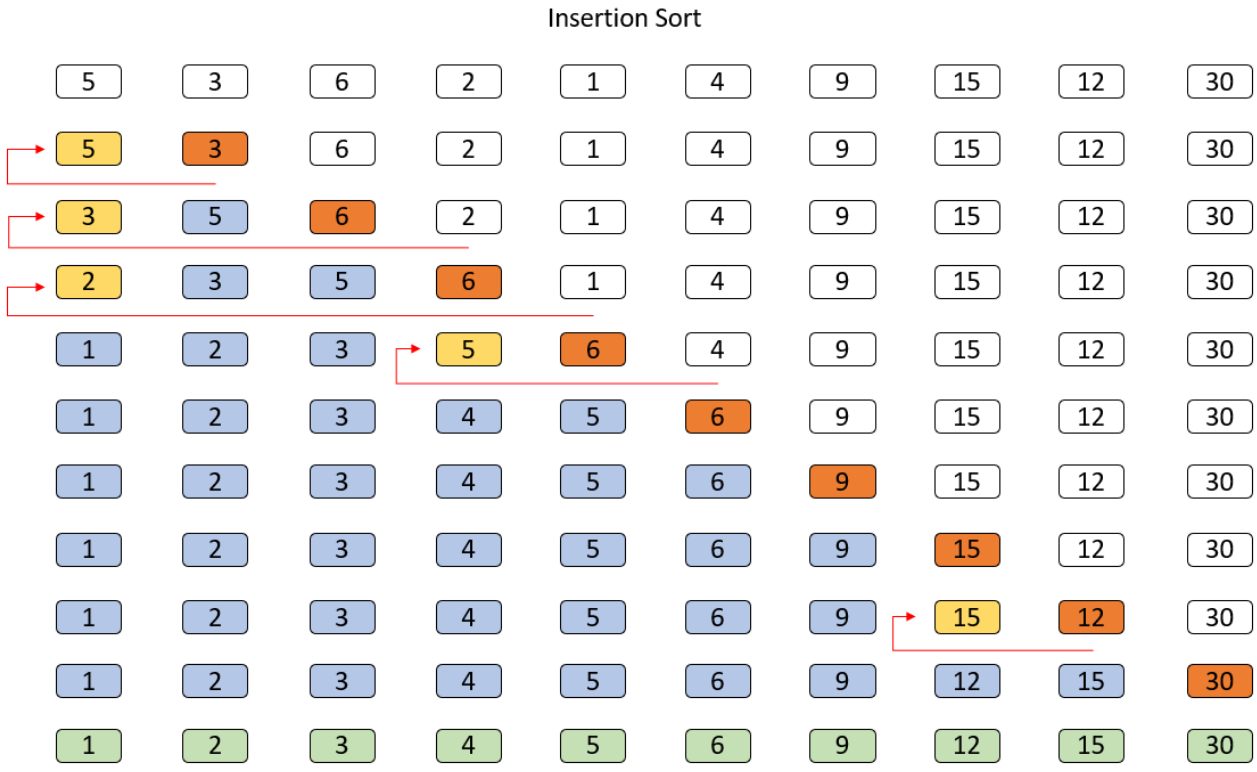
$$= \sum_{i=0}^{nMax} 1 - \sum_{i=0}^{nMax} i$$

$$= \sum_{i=0}^{nMax} 1 - \frac{(nMax)(nMax - 1)}{2}$$

$$= \frac{nMax(nMax - 1)}{2}$$

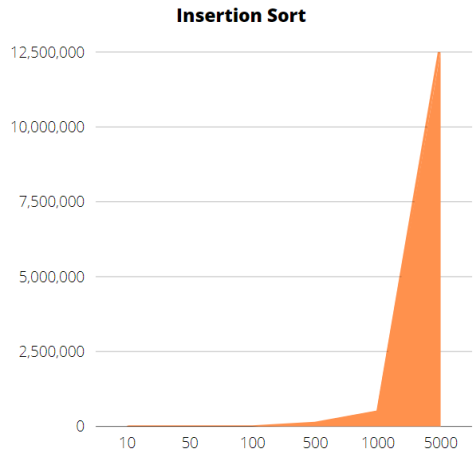
$$\approx \frac{1}{2} nMax^2 \in O(nMax^2)$$

3.1.2 Ilustrasi Insertion Sort



3.1.3 Tabel dan Grafik Perubahan Running Time Insertion Sort

Jumlah Data	T(n)
10	50
50	1250
100	5000
500	125000
1000	500000
5000	12500000



3.2.1 Perhitungan Kompleksitas Waktu dan Kelas Efisiensi dari Bubble-sort

$$T(n) = \sum_{i=0}^{nMax-1} \sum_{j=0}^{nMax-(i-1)} 1 = \sum_{i=0}^{nMax-1} [u - l + 1]$$

$$= \sum_{i=0}^{nMax-1} [(nMax - (i-1)) - 0 + 1]$$

$$= \sum_{i=0}^{nMax-1} [nMax - i + 2]$$

$$= \sum_{i=0}^{nMax-1} (nMax + 2) - \sum_{i=0}^{nMax-1} i$$

$$= (nMax + 2) \sum_{i=0}^{nMax-1} 1 - \frac{nMax - 1((nMax - 1) + 1)}{2}$$

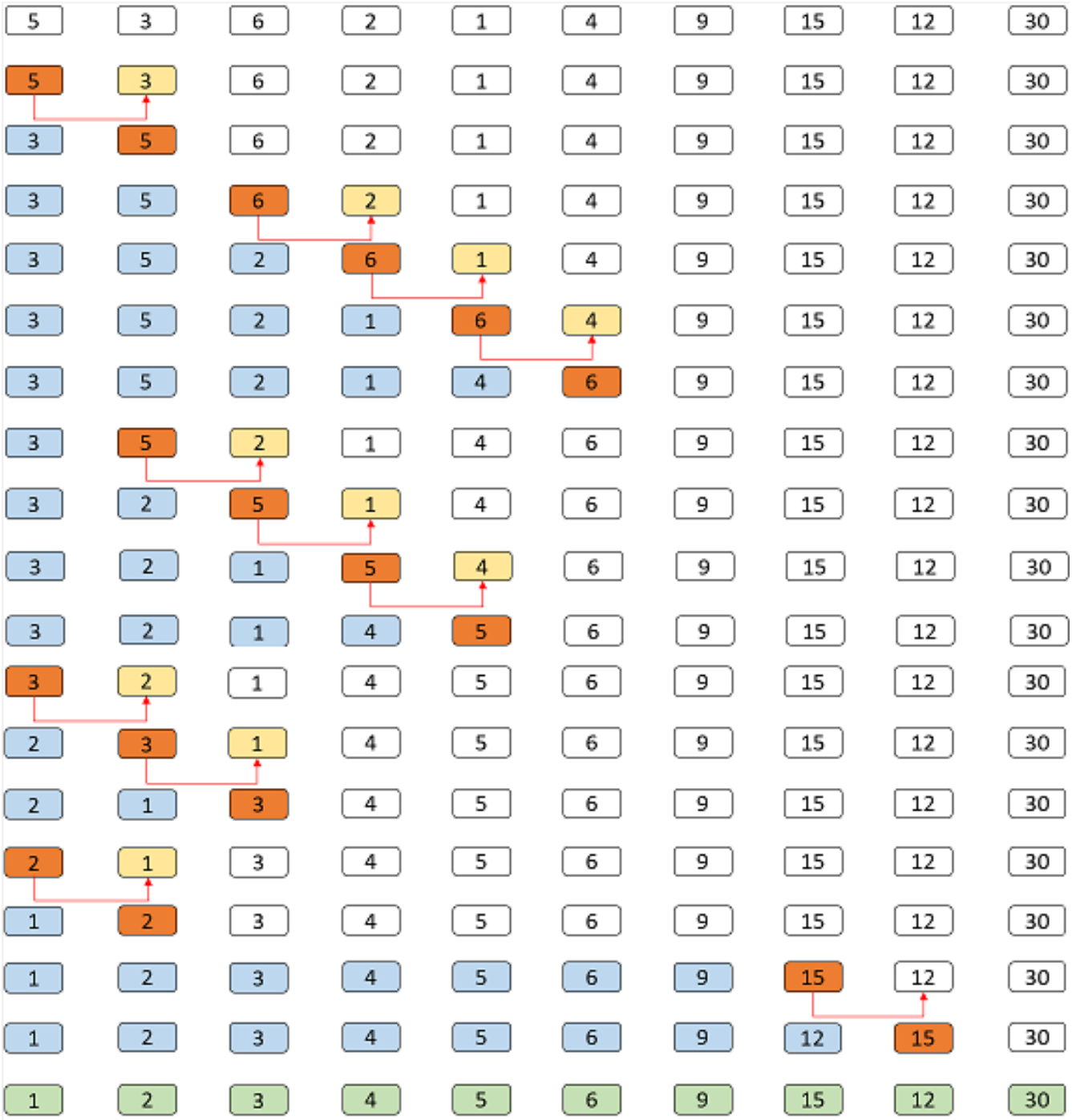
$$= (nMax + 2)^2 - \frac{nMax - 1((nMax - 1) + 1)}{2}$$

$$= (nMax + 2)^2 - \frac{nMax - 1((nMax - 1) + 1)}{2}$$

$$\approx nMax^2 + 4nMax + 4 \in O(nMax^2)$$

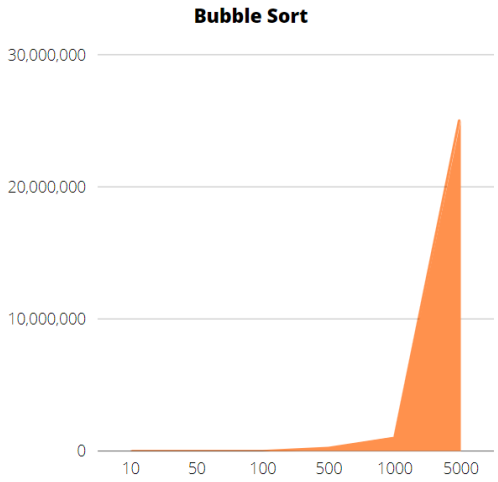
3.2.2 Ilustrasi Bubble Sort

Bubble Sort

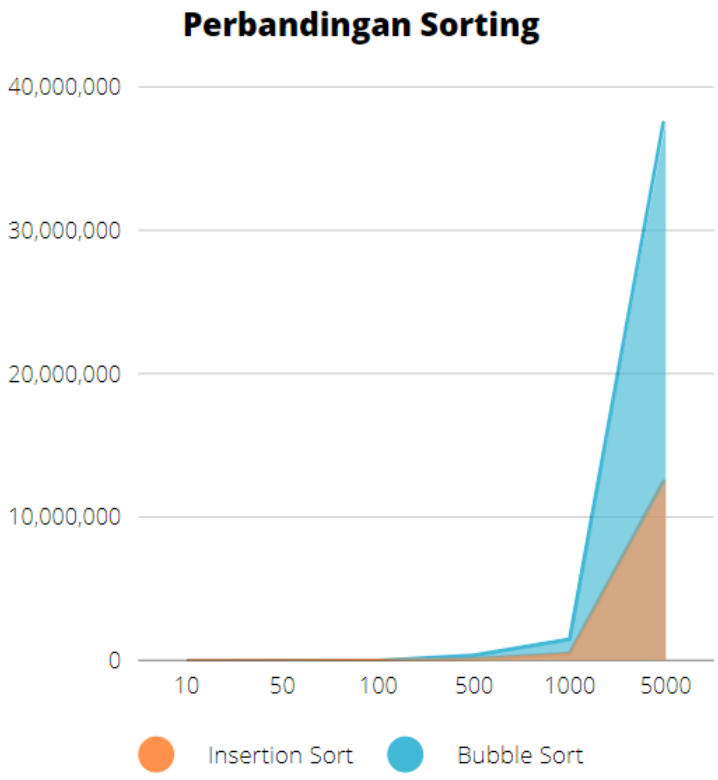


3.2.3 Tabel dan Grafik Perubahan Running Time Bubble Sort

Jumlah Data	T(n)
10	144
50	2704
100	10404
500	252004
1000	1004004
5000	25020004



3.3.1 Perbandingan Antara Insertion sort dan Bubble sort



BAB IV

4.1 Kesimpulan

Dari yang T(n) dan kelas efisiensi yang didapatkan terhadap *insertion-sort* dan *bubble-sort*, dan dari grafik yang telah dibuat, dapat disimpulkan bahwa waktu yang dibutuhkan untuk mengurutkan suatu *array* dengan 10 elemen acak, algoritma pengurutan *insertion-sort* lebih cepat dibandingkan *bubble-sort*, dikarenakan *insertion-sort* membandingkan dua data, antara data yang sedang dieksekusi dengan data sebelumnya dilihat apakah data sebelumnya lebih besar atau tidak, jika data tersebut lebih besar maka data akan langsung ditukar sampai elemen pertama. Sehingga proses ini akan lebih cepat dalam mengurutkan suatu array data acak.

Referensi

(13508054), S. N. (2009). Kompleksitas Algoritma Pengurutan . *MAKALAH IF2091 STRATEGI ALGORITMIK TAHUN 2009*.

Rahayuningsih, P. A. (2016). Analisis Perbandingan Kompleksitas Algoritma Pengurutan Nilai (Sorting). *Jurnal Evolusi Volume 4 Nomor 2 - 2016*.

Rivan, M. E. (2016). Perbandingan Performa Kombinasi Algoritma . *ANNUAL RESEARCH SEMINAR 2016*.