Vol. 4, No. 2, Juni 2020, 178 – 187
E-ISSN: 2548-3587

Algoritma Metode Pengurutan *Bubble Sort* dan *Quick Sort* Dalam Bahasa Pemrograman C++

Rita Wahyuni Arifin [1,*], Didik Setiyadi [2]

[1] Manajemen Informatika;Universitas Bina Insani;Jl. Siliwangi No.6 Rawa Panjang Bekasi Timur 17114 Indonesia; Telp. (021) 82436886 / (021) 82436996;e-mail:ritawahyuni@binainsani.ac.id

[2] Teknik Informatika;Universitas Bina Insani;Jl. Siliwangi No.6 Rawa Panjang Bekasi Timur 17114 Indonesia; Telp. (021) 82436886 / (021) 82436996;e-mail:didiksetiyadi@binainsani.ac.id

* Korespondensi: e-mail: ritawahyuni@binainsani.ac.id

Diterima: 04 Mei 2020; Review: 14 Mei 2020; Disetujui: 20 Mei 2020

Cara sitasi: Arifin RW, Setiyadi D. 2020. Algoritma Metode Pengurutan *Bubble Sort* dan *Quick Sort* Dalam Bahasa Pemrograman C++. Information System for Educators and Professionals. 4 (2): 178 – 187.

Abstrak: Data yang disimpan dalam komputer sangat beragam dan banyak. Agar data tersebut dapat dengan mudah terbaca, saat penyisipan dan dalam pencarian maka data perlu diurutkan. Namun kenyataanya dalam proses pengurutan tidak semudah yang dibayangkan karena dibalik proses pengurutan ada algoritma yang harus dibuat sehingga proses pengurutan data dapat dilakukan dengan secara efektif dan efisien. Permasalahan yang muncul metode algoritma apa yang bisa digunakan agar proses pengurutan *(sorting)* dapat dengan mudah dilakukan. Terdapat beberapa metode pengurutan yaitu metode *insertion, selection, bubble, merge*, dan *Quick Sort*. Pada penelitian ini dilakukan perbandingan dua metode pengurutan yaitu pengurutan *Bubble Sort*, dan *Quick Sort* dalam melakukan proses pengurutan data dengan menyusun dalam urutan algortimanya, dan implementasinya menggunakan bahasa pemrograman C++. Hasil dari penelitian ini adalah analisis proses pengurutan *Bubble Sort* lebih lama dibanding *Quick Sort* karena terdapat 7 putaran dan setiap putaran melakukan 10 kali perbandingan serta terdapat 52 baris sintak programnya sedangkan *Quick Sort* langkah penyelesaiannya menghasilkan 2 partisi dan rekursi 21 kali dengan 42 baris sintak programnya.

Kata kunci: algoritma pengurutan, bahasa pemrograman C++, *bubble sort*, *Quick Sort*.

Abstract: *Data collected in computers is very diverse and numerous. So that the data can be easily read, during insertion and in search the data needs to be sorted. But in fact the ordering process is not as easy as imagined because the sorting process there is an algorithm that must be made so that the data sorting process can be done effectively and efficiently. The problem that arises is what method can be used so that the sorting process can be done easily. There are several sorting methods, namely insertion, selection, bubble, merge, and Quick Sort methods. In this research, two sorting methods are tested, namely sorting Bubble Sort, and Quick Sorting in the process of sorting data by arranging in order of acceptance, and its implementation using C ++ programming language. The results of this research are the Bubble Sorting analysis process takes longer than fast sorting because there are 7 rounds and every round 10 times including and 52 program syntax lines while fast sorting steps successfully produce 2 partitions and recursion 21 times with 42 program syntax lines.*

Keywords: *bubble sort, C ++ programming language, Quick Sort, sorting algorithm.*

1. Pendahuluan

Manusia diciptakan oleh sang pencipta dianugerahi sebuah pemikiran atau biasa disebut dengan logika. Dengan Logika manusia dapat memecahkan suatu persoalan atau permasalahan yang terjadi dalam hidupnya. Pada saat dihadapkan suatu permasalahan

tindakan selanjutnya adalah mencari cara bagaimana menyelesaikannya. Komputer adalah alat yang digunakan untuk dapat memecahkan persoalan, namun komputer harus diberikan perintah pemograman dengan bahasa pemrograman yang dimengerti untuk mengerjakan perintah yang diberikan [1]. Program adalah kumpulan intruksi komputer, sedangkan metode dan tahapan sistematis dalam program adalah algoritma [2]. Perintah kepada komputer harus sistematis, rinci serta memenuhi kaidah logika tertentu [1]. Algoritma sangat diperlukan untuk menyelesaikan berbagai masalah pemrograman pada bidang komputer. Jika algoritma dirancang tidak baik maka proses pemrograman akan menjadi rusak, lambat dan tidak efisien. Algoritma pertama kali dperkenalkan oleh Abu Ja'far Muhammad Ibnu Musa Al-Kwarizmi dalam buku Aljabarnya, jadi pengertian algoritma adalah urutan langkah-langkah penyelesaian suatu masalah secara sistematika dan logis [3].

Pengurutan data atau sorting merupakan salah satu konsep yang sangat penting dalam dunia pemrograman komputer dan merupakan materi operasi yang paling banyak dipelajari dalam dunia komputer [4]. Terdapat banyak bentuk algoritma telah dikembangkan dan ditingkatkan untuk membuat pengurutan lebih cepat baik untuk angka ataupun huruf. Semakin efisien suatu algoritma, maka saat dieksekusi dan dijalankan akan menghabiskan waktu yang lebih cepat dan bisa menerima lebih banyak masukan dari user. Secara umum ada 2 jenis pengurutan (sorting) data yaitu pengurutan naik (*ascending*), yaitu jika data disusun mulai dari nilai yang paling kecil hingga yang paling besar dan pengurutan turun (*descending*), yaitu jika data yang disusun mulai dari nilai yang paling besar hingga paling kecil. Algoritma pemrograman pengurutan sangat banyak dan bervariasi dari yang sederhana hingga yang kompleks. Karena materi pengurutan salah satu materi yang sangat banyak dipelajari sehingga banyak metode pengurutan yang ditemukan atau digunakan untuk melakukan pengurutan data diantaranya metode *Insertion sort*, *Selection sort*, *Bubble Sort*, *Merge Sort*, *Quick Sort*.

Dalam penelitian ini penulis melakukan beberapa referensi dari beberapa sumber yang membahas mengenai algoritma metode pengurutan diantaranya adalah Sonita [5] yang melakukan analisa perbandingan algortima pengurutan data, yaitu: *Bubble Sort*, *Merge Sort*, dan *Quick Sort* untuk mendapatkan waktu proses yang baik dalam proses pengurutan kombinasi angka dan huruf dan hasil dari penelitiannya adalah Algoritma *Bubble Sort*, *Quick Sort*, dan *Merge Sort*, dapat disimpulkan bahwa Algoritma *Quick Sort* memiliki waktu yang lebih cepat dan *Bubble Sort* membutuhkan waktu komputasi yang paling lama [6].

Dari penulis lain juga melakukan penelitian mengenai pengurutan data dengan melakukan percobaan untuk menyortir menggunakan tipe data integer. Efektivitas suatu algoritma dapat diukur dari waktu dan ruang memory yang diperlukan untuk menjalankan algoritma. Metode pengurutan dengan *Selection sort*, *Insertion sort*, shell sort dan *Bubble Sort*. Algoritma diimplementasikan dengan menggunakan bahasa pemrograman Visual 6.0 dan hasil dari penelitiannya adalah Algoritma *Quick Sort* lebih cepat dalam melakukan pengurutan data jika dibandingkan dengan *Shell Sort, Insertion sort, Selection sort dan Bubble Sort* [7].

Pujiatiningsih melakukan penelitian algoritma dengan metode pengurutan *Quick Sort*, *Selection sort* dan metode pengurutan *Heapsort*, dari algoritma tersebut akan dibandingkan dari segi waktu dan memori yang dipakai. Hasil penelitiannya adalah dari sisi waktu selectionsort lebih unggul dibanding algoritma Ouicksort dan Heapsort sedangkan dari sisi memori yang lebih unggul adalah adalah Heapsort [8]

Penulis Rivan melakukan perbandingan performa kombinasi antara algoritma pengurutan *quick-Insertion sort* dan *Merge-Insertion sort* dan hasilnya kombinasi algoritma *Quick Insertion sort* 15% lebih cepat dibanding dengan *Quick Sort*, sedangkan untuk Merge-*Insertion sort* lebih cepat 34,8% dibanding dengan *Merge Sort* dalam batas 16 [9].

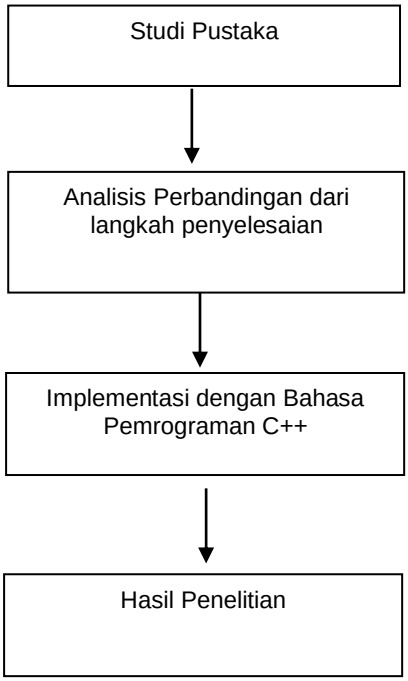
Penelitian lannya yang dilakukan oleh Muhammad Saiful mengenai seberapa cepat dan efisien *Quick Sort* bekerja, dan menunjukkan isu-isu penting yang muncul dalam proses implementasi *Quick Sort*. Hasil penelitiannya menyatakan *Quick Sort* sebagai pilihan implementasi dari berbagai system sort yang ditemukan dalam berbagai bahasa pemrograman serta digunakan untuk menyelesaikan berbagai persoalan pengurutan data [10].

Penulis lain Retnoningsih juga melakukan penelitian dengan melakukan perbandingan 2 algoritma metode pengurutan yaitu *Selection sort* menggunakan prinsip pertukaran elemen dalam proses sorting, serta metode insertionsort dengan prinsip geser dan sisip elemen dalam proses pengurutan. Hasil dari penelitiannya adalah metode insertion lebih baik dengan jumlah data yang sedikit, sedangkan metode selectionsort lebih baik untuk data yang berjumlah lebih banyak [11].

Dalam penelitian ini penulis hanya mengambil 2 metode pengurutan yaitu bubble dan quick alasannya karena memiliki pola yang berbeda dalam proses pengurutan datanya. Dalam penelitian ini penulis melakukan analisa perbandingan dari kedua algoritma dalam proses pengurutan dari sisi banyaknya langkah penyelesaian dan penyelesaian dengan bahasa pemrograman C++. Seorang ahli bernama Bjarne Stroustrup pada tahun 1980 mengembangkan bahasa C yang dinamakan "*C with Classes*" dan pada tahun 1983 menjadi C++. Perbedaannya adalahh menambahkan *Object Oriented Programming* (OOP), dengan tujuan utamanya membantu dan mengelola program besar dan kompleks [12].

2. Metode Penelitian

Dalam penyusunan penelitian ini metode penelitian yang digunakan adalah metode pengumpulan data dengan melakukan studi pustaka dari beberapa jurnal dan buku yang membahas mengenai algoritma metode pengurutan data. Kemudian melakukan analisis dari penyelesaian Tujuan dari metode ini adalah untuk menganalisis perbandingan mengurutan dengan metode merge, quick, dan heap sort dari banyak langkah penyelesaian. Dengan teknik pengumpulan data yang baik akan menghasilkan perbandingan dari apa yang dilakukan sebelumnya. Adapun kerangka pemikiran penelitian dalam gambar 1.



Sumber: Hasil Penelitian (2020)

Gambar 1. Kerangka Pemikiran

Berdasarkan Gambar 2 kerangka penelitian yang dirancang mulai dari studi pustaka yaitu penulis membaca beberapa referensi dari buku dan jurnal yang membahas mengenai metode algoritma pengurutan bubble dan *Quick Sort*, kemudian melakukan studi kasus dengan menghitung banyaknya langkah penyelesaian, kemudian diimplementasikan dalam bahasa pemrograman c++.

Algoritma pengurutan bubblesort membandingkan elemen-elemen larik dari kanan ke kiri secara garis besar adalah sebagai berikut [13]:

Untuk setiap *pass i*=1, 2, ..., n-1, lakukan:.....(1)
Untuk setiap *k*=n, n-1, ..., *i*+1, lakukan:.....(2)
Bandingkan *L[k]* dengan *L[k-1]*.
Jika *L[k]* < *L[k-1]*, maka pertukarkan *L[k] dengan L[k-1]* .....(3)

3. Hasil dan Pembahasan

Konsep Algoritma *Bubble Sort*

Algoritma *Bubble Sort* adalah algoritma yang terinspirasi dari gelembung sabun yang berada pada permukaan air. Karena berat jenis gelembung sabun lebih ringan daripada berat jenis air, maka gelembung sabun akan terapung ke atas permukaan. Apabila kita menginginkan larik terurut naik, maka elemen larik yang paling kecil diapungkan ke atas melalui proses pertukaran atau bisa disebutkan bahwa arah perbandingan dilakukan dari belakang.

Studi kasus diberikan urutan bilangan adalah seperti berikut:

59 20 56 90 3 40 14 78 3 -1

Jika data tersebut akan diurutkan dengan urutan menaik (*ascending*) menggunakan metode *Bubble Sort* maka proses dan langkah penyelesaiannya pada tabel 1.

Tabel 1. Proses penyelesaian pengurutan data secara *ascending*

k	Elemen yang dibandingkan	Pertukarkan	Hasil Sementara pengurutan
***pass* 1:**			
k=10	L[10]<L[9]? (3<-1)	Tidak	59, 20, 56, 90, 3, 40, 14, 78, **-1, 3**
k=9	L[9]<L[8]? (-1<78)	Ya	59, 20, 56, 90, 3, 40, 14, **-1, 78**, 3
k=8	L[8]<L[7]? (-1<14)	Ya	59, 20, 56, 90, 3, 40, **-1, 14**, 78, 3
k=7	L[7]<L[6]? (-1<40)	Ya	59, 20, 56, 90, 3, **-1, 40,** 14, 78, 3
k=6	L[6]<L[5]? (-1<3)	Ya	59, 20, 56, 90, **-1, 3,** 40**,** 14, 78, 3
k=5	L[5]<L[4]? (-1<90)	Ya	59, 20, 56, **-1, 90**, 3, 40**,** 14, 78, 3
k=4	L[4]<L[3]? (-1<56)	Ya	59, 20, **-1, 56**, 90, 3, 40, 14, 78, 3
k=3	L[3]<L[2]? (-1<20)	Ya	59, **-1, 20**, 56, 90, 3, 40, 14, 78, 3
k=2	L[2]<L[1]? (-1<59)	Ya	**-1, 59**, 20, 56, 90, 3, 40, 14, 78, 3
***pass* 2:**			
k=10	L[10]<L[9]? (3<78)	Ya	-1, 59, 20, 56, 90, 3, 40, 14, **3, 78**
k=9	L[9]<L[8]? (3<14)	Ya	-1, 59, 20, 56, 90, 3, 40, **3, 14**, 78
k=8	L[8]<L[7]? (3<40)	Ya	-1, 59, 20, 56, 90, 3, **3, 40**, 14, 78
k=7	L[7]<L[6]? (3<3)	Tidak	-1, 59, 20, 56, 90, **3, 3**, 40, 14, 78
k=6	L[6]<L[5]? (3<90)	Ya	-1, 59, 20, 56, **3, 90**, 3, 40, 14, 78
k=5	L[5]<L[4]? (3<56)	Ya	-1, 59, 20, **3, 56**, 90, 3, 40, 14, 78
k=4	L[4]<L[3]? (3<20)	Ya	-1, 59, **3, 20**, 56, 90, 3, 40, 14, 78
k=3	L[3]<L[2]? (-3<59)	Ya	-1, **3, 59**, 20, 56, 90, 3, 40, 14, 78
k=2	L[2]<L[1]? (3<-1)	Tidak	**-1, 3**, 59, 20, 56, 90, 3, 40, 14, 78
***pass* 3:**			
k=10	L[10]<L[9]? (78<14)	Tidak	-1, 3, 59, 20, 56, 90, 3, 40, **14, 78**
k=9	L[9]<L[8]? (14<40)	Ya	-1, 3, 59, 20, 56, 90, 3, **14, 40**, 78
k=8	L[8]<L[7]? (14<3)	Tidak	-1, 3, 59, 20, 56, 90, **3, 14**, 40, 78
k=7	L[7]<L[6]? (3<90)	Ya	-1, 3, 59, 20, 56, **3, 90**, 14, 40, 78
k=6	L[6]<L[5]? (3<56)	Ya	-1, 3, 59, 20, **3, 56**, 90, 14, 40, 78
k=5	L[5]<L[4]? (3<20)	Ya	-1, 3, 59, **3, 20**, 56, 90, 14, 40, 78
k=4	L[4]<L[3]? (3<59)	Ya	-1, 3, **3, 59**, 20, 56, 90, 14, 40, 78
k=3	L[3]<L[2]? (-3<3)	Tidak	-1, **3, 3**, 59, 20, 56, 90, 14, 40, 78
***pass* 4:**			
k=10	L[10]<L[9]? (78<40)	Tidak	-1, **3, 3**, 59, 20, 56, 90, 14, **40, 78**
k=9	L[9]<L[8]? (40<14)	Tidak	-1, **3, 3**, 59, 20, 56, 90, **14, 40**, 78
k=8	L[8]<L[7]? (14<90)	Ya	-1, **3, 3**, 59, 20, 56, **14, 90**, 40, 78
k=7	L[7]<L[6]? (14<56)	Ya	-1, 3, 3, 59, 20, **14, 56**, 90, 40, 78
k=6	L[6]<L[5]? (14<20)	Ya	-1, 3, 3, 59, **14, 20**, 56, 90, 40, 78
k=5	L[5]<L[4]? (14<59)	Ya	-1, 3, 3, **14, 59,** 20, 56, 90, 40, 78
k=4	L[4]<L[3]? (14<3)	Tidak	-1, 3, **3, 14,** 59, 20, 56, 90, 40, 78
***pass* 5:**			
k=10	L[10]<L[9]? (78<40)	Tidak	-1, 3, 3, 14, 59, 20, 56, 90, **40, 78**
k=9	L[9]<L[8]? (40<90)	Ya	-1, 3, 3, 14, 59, 20, 56, **40, 90**, 78
k=8	L[8]<L[7]? (40<56)	Ya	-1, 3, 3, 14, 59, 20, **40, 56**, 90, 78
k=7	L[7]<L[6]? (40<20)	Tidak	-1, 3, 3, 14, 59, **20, 40**, 56, 90, 78
k=6	L[6]<L[5]? (20<59)	Ya	-1, 3, 3, 14, **20, 59**, 40, 56, 90, 78
k=5	L[5]<L[4]? (20<14)	Tidak	-1, 3, 3, **14, 20,** 59, 40, 56, 90, 78
***pass* 6:**			
k=10	L[10]<L[9]? (78<90)	Ya	-1, 3, 3, 14, 20, 59, 40, 56, **78, 90**
k=9	L[9]<L[8]? (78<56)	Tidak	-1, 3, 3, 14, 20, 59, 40, **56, 78**, 90
k=8	L[8]<L[7]? (56<40)	Tidak	-1, 3, 3, 14, 20, 59, **40, 56**, 78, 90
k=7	L[7]<L[6]? (40<59)	Ya	-1, 3, 3, 14, 20, **40, 59**, 56, 78, 90
k=6	L[6]<L[5]? (40<20)	Tidak	-1, 3, 3, 14, **20, 40**, 59, 56, 78, 90
***pass* 7:**			
k=10	L[10]<L[9]? (90<78)	Tidak	-1, 3, 3, 14, 20, 40, 59, 56, **78, 90**
k=9	L[9]<L[8]? (78<56)	Tidak	-1, 3, 3, 14, 20, 40, 59, **56, 78**, 90

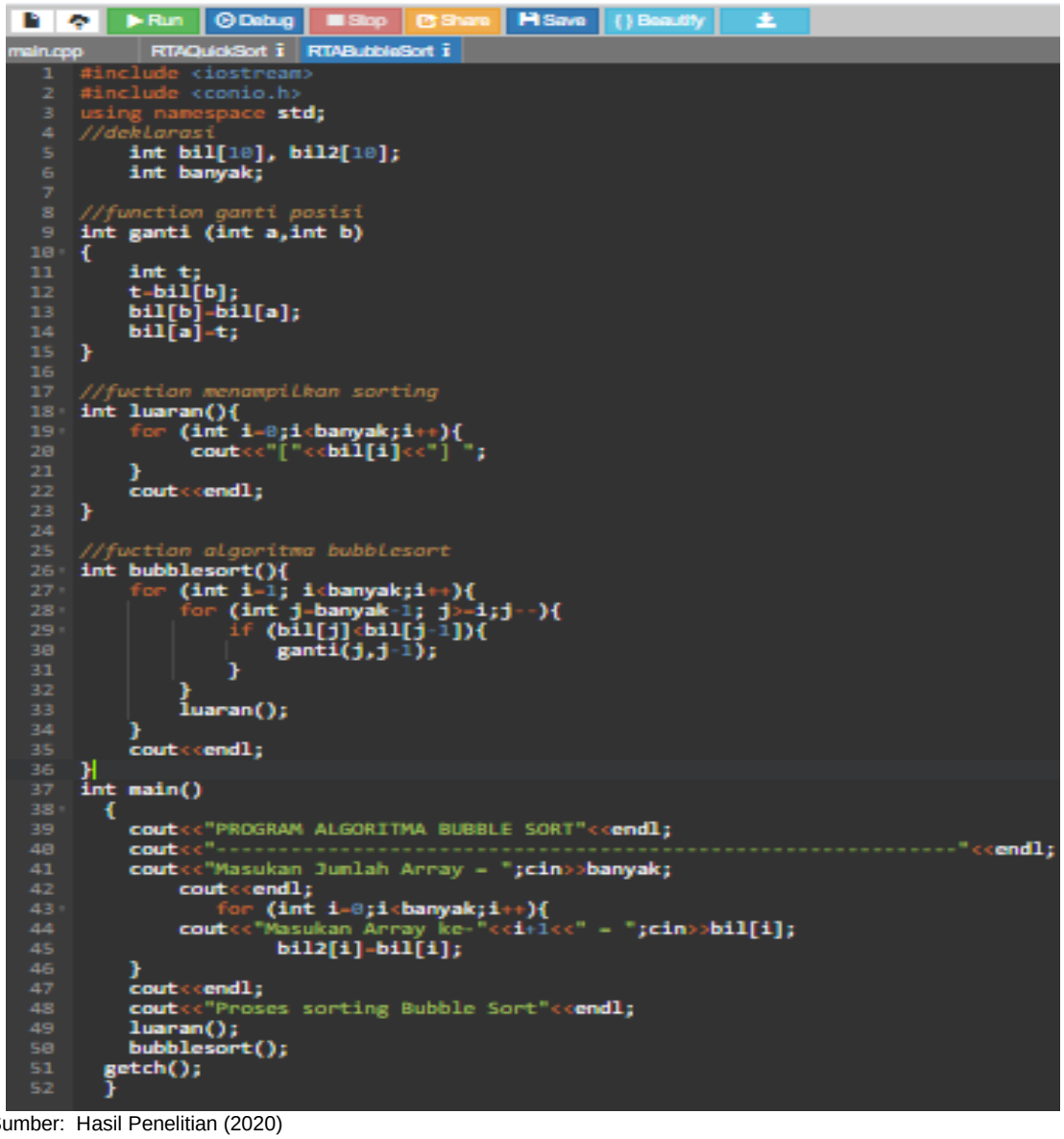
k	Elemen yang dibandingkan	Pertukarkan	Hasil Sementara pengurutan
k=8	L[8]<L[7]? (56<59)	Ya	-1, 3, 3, 14, 20, 40, **56, 59**, 78, 90
k=7	L[7]<L[6]? (56<40)	Tidak	-1, 3, 3, 14, 20, **40, 56**, 59, 78, 90

Sumber: Hasil Penelitian (2020)

Berdasarkan data diatas maka banyaknya langkah penyelesaian dari algoritma *Bubble Sort* adalah 7 kali pass dengan 49 kali perbandingan.

Implementasi *Bubble Sort* dalam Program C++

Pengurutan dengan menggunakan sintak pemrograman C++ dengan urutan sintak dimulai dari melakukan deklarasi variabel array yaitu data [10] dan data2 [10]. Untuk menuliskan sintak program menggunakan aplikasi Borland 5.02 atau bisa juga menggunakan teks editor online yaitu https://www.onlinegdb.com/ sintak lebih lanjut dapat dilihat pada gambar 2.

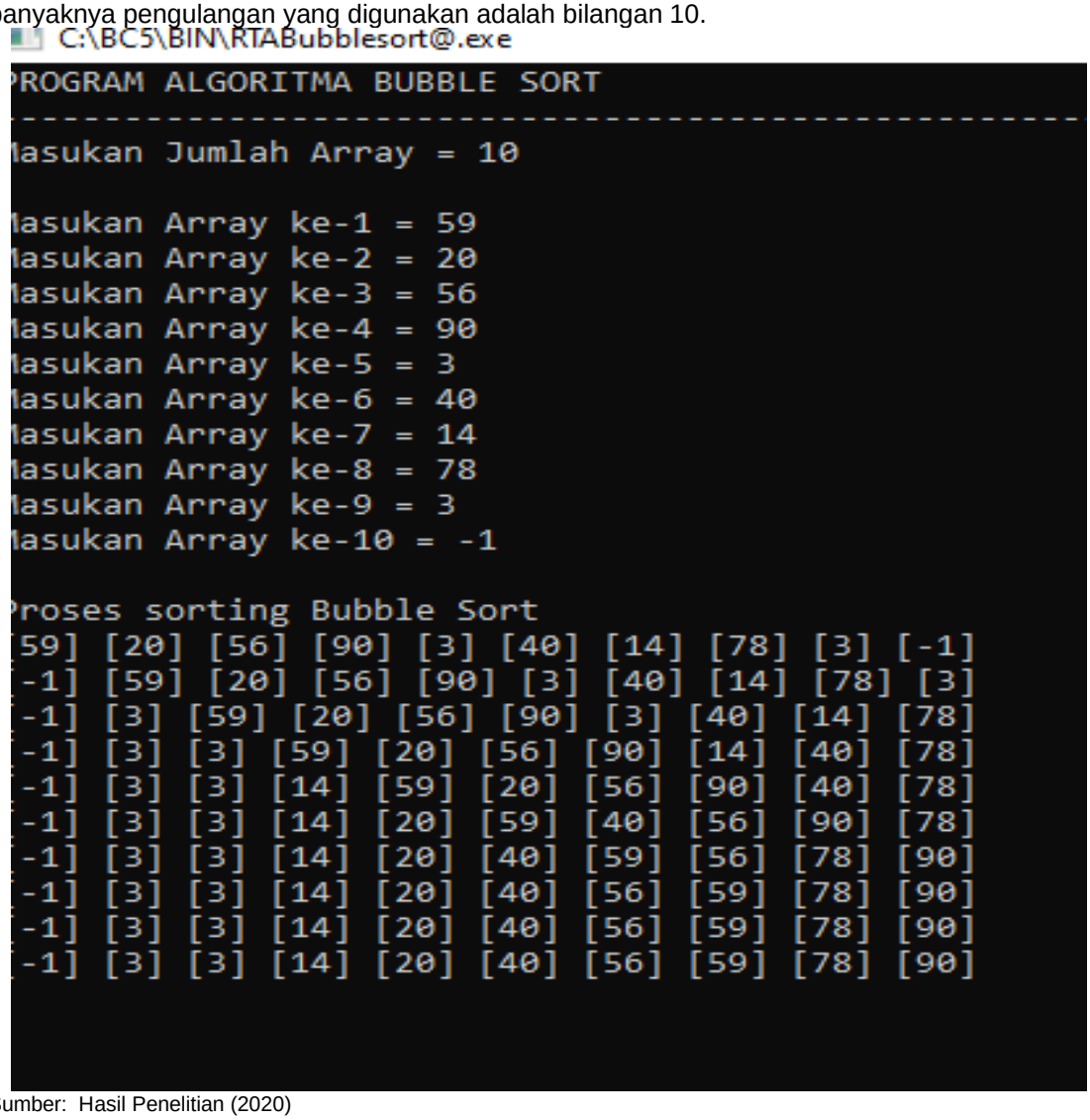


Sumber: Hasil Penelitian (2020)

Gambar 2. Tampilan sintak program C++

Berdasarkan gambar 2 maka penjelasan source algoritma *Bubble Sort* dibahasa C++ adalah dalam program dibuatkan 3 *function()* yaitu *function* ganti(), *function* luaran(), *function Bubble Sort*(). Pada *function* ganti di baris 9-15 dengan memberikan variabel t tipe data integer, untuk function luaran() adalah perintah menampilkan hasil sorting dengan memberikan perintah looping sebanyak bil yang diinput, sedangkan untuk function *Bubble Sort* melakukan

perulangan sebanyak data yang diinput dan melakukan pemeriksaan kondisi jika bilangan indek ke j lebih besar dari variabel j-1 maka memanggil function ganti(). Jumlah baris pada sintak diatas adalah sebanyak 52 baris. Sintak program diatas Setelah dirun maka tampilan output dapat dilihat pada gambar 5, banyaknya jumlah iterasi sesuai dengan banyaknya jumlah data yang diinputkan. Berdasarkan hasil keluaran tampil 9 kali hasil pengurutan karena dalam sintak banyaknya pengulangan yang digunakan adalah bilangan 10.



Sumber: Hasil Penelitian (2020)

Gambar 3. Tampilan luaran pogram *Bubble Sort*

Konsep Algoritma *Quick Sort*

Metode pengurutan *Quick Sort* adalah metode pengurutan yang menjadikan sebuah tabel data yang akan diurutkan menjadi dua buah sub bagian yang ditelursuri dari kiri dan kanan. *Quick Sort* mengurutkan menggunakan strategi Divide and Conquer untuk membagi array menjadi dua sub-array. Disebut metode *Quick Sort*, karena algoritma *Quick Sort* mengurutkan dengan sangat cepat.

Pada algoritma *Quick Sort*, memilih pivot adalah menentukan quickshort memberikan performa baik atau buruk. Salah satu cara menemukan lokasi pivot adalah memilih nilai yang paling kiri dalam suatu urutan yang tidak terurut, kemudian memindahkan nilai elemen yang lebih kecil dari pivot ke sebelah kiri dan nilai yang lebih besar dari nilai pivot pindah ke sebelah kanan.

Beberapa cara dalam menentukan pivot: Pivot adalah elemen pertama, tengah atau elemen tengah, pivot dipilih secara acak dan bagus untuk data yang belum terurut, dan pivot sebagai elemen median tabel dengan cara membagi dua bagian tabel dengan rumus = n/2.

Algoritma *Quick Sort* terdiri dari dua prosedur yaitu partisi dan prosedur *Quick Sort*. Studi kasus diberikan urutan bilangan adalah seperti berikut:

59 20 56 90 3 40 14 78 3 -1

Jika data tersebut akan diurutkan dengan urutan menaik (*ascending*) menggunakan metode *Quick Sort* maka proses dan langkah penyelesaiannya dalam bahasa manusia dan bahasa algoritmik pada tabel 2.

Tabel 2. Proses penyelesaian pengurutan data secara *ascending*

No	Keterangan	Hasil Sementara pengurutan
1	Deklarasikan nilai arraynya Data[a]={ 59 20 56 90 3 40 14 78 3 -1}	59 20 56 90 3 40 14 78 3 -1
2	Tentukan nilai pivotnya dan nilai dipilih adalah yang ditengah yaitu 3.	3
3	Bandingkan dengan angka 59 Karena angka 59 lebih besar maka letakkan angka 59 setelah pivot	3 59
4	Lanjut ke angka 20 lalu bandingkan dengan pivot karena angka 20 lebih besar maka letakkan disebelah kanan setelah pivot	3 59 20
5	Lanjut ke angka 56, cek apakah angka 56 lebih besar atau lebih kecil dari pivot, karena angka 56 lebih besar maka letakkan disebelah kanan setelah pivot	3 59 20 56
6	Lanjut ke angka 90, cek apakah angka 90 lebih besar atau lebih kecil dari pivot, karena angka 90 lebih besar maka letakkan disebelah kanan setelah pivot.	3 59 20 56 90
7	Lanjut ke angka 40, cek apakah angka 40 lebih besar atau lebih kecil dari pivot, karena angka 40 lebih besar dari pivot dan letaknya disebelah kanan maka letaknya tetap.	3 59 20 56 90 40
8	Lanjut ke angka 14, cek apakah angka 14 lebih besar atau lebih kecil dari pivot, karena angka 14 lebih besar dari pivot dan letaknya disebelah kanan pivot maka letaknya tetap.	3 59 20 56 90 40 14
9	Lanjut ke angka 78, cek apakah angka 78 lebih besar atau lebih kecil dari pivot, karena angka 78 lebih besar dari pivot dan letaknya disebelah kanan pivot maka letaknya tetap.	3 59 20 56 90 40 14 78
10	Lanjut ke angka 3, cek apakah angka 3 lebih besar atau lebih kecil dari pivot, karena angka 3 sama dengan pivot, letaknya disebelah kanan pivot maka letaknya pindah ke kiri.	3 ; 3 59 20 56 90 40 14 78
11	Lanjut ke angka -1, cek apakah angka 3 lebih besar atau lebih kecil dari pivot, karena angka -1 lebih kecil dari pivot, letaknya disebelah kanan pivot maka letaknya pindah ke kiri.	-1 3 ; 3 59 20 56 90 40 14 78 (Partisi 1 ; Partisi 2)
12	Untuk partisi 1 sudah terurut maka perlu lakukan langkah yang sama untuk mengurutkan array pada partisi 2 dengan menentukan kembali pivotnya, yaitu 56.	-1 3 ; 3 59 20 56 90 40 14 78
13	Bandingkan dengan angka 3 Karena angka 3 sama dengan pivot maka tidak ada perpindahan dan letak tetap.	-1 3 ; 3 59 20 56 90 40 14 78
14	Bandingkan dengan angka 59 Karena angka 59 lebih besar maka letakkan angka 59 setelah pivot	-1 3 3 ; 56 59
15	Lanjut ke angka 20 lalu bandingkan dengan pivot karena angka 20 lebih kecil maka letakkan disebelah kiri pivot	-1 3 3 20 ; 56 59
16	Lanjut ke angka 90, cek apakah angka 90 lebih besar atau lebih kecil dari pivot, karena angka 90 lebih besar maka letakkan disebelah kanan setelah pivot.	-1 3 3 20 ; 56 59 90
17	Lanjut ke angka 40, cek apakah angka 40 lebih besar atau lebih kecil dari pivot, karena angka 40 lebih kecil dari pivot dan letaknya disebelah kanan maka letaknya pindah ke kiri pivot.	-1 3 3 20 40 ; 56 59 90
18	Lanjut ke angka 14, cek apakah angka 14 lebih besar atau lebih kecil dari pivot, karena angka	

No	Keterangan	Hasil Sementara pengurutan
	14 kecil dari pivot dan letaknya disebelah kanan pivot maka pindah ke kiri pivot.	-1 3 3 20 40 14 56 59 90
19	Lanjut ke angka 78, cek apakah angka 78 lebih besar atau lebih kecil dari pivot, karena angka 78 lebih besar dari pivot dan letaknya disebelah kanan pivot maka letaknya tetap.	-1 3 3 20 40 14 56 59 90 78
20	Hasil pengurutannya masih belum terurut dengan maksimal maka perlu dilakukan pengurutan.	-1 3 3 20 40 14 56 59 90 78
21	Selanjutnya lakukan langkah-langkah diatas dengan menentukan pivot dan pertukarannya	-1 3 3 20 14 40 56 59 78 90 78

Sumber: Hasil Penelitian(2020)

Berdasarkan data tabel 2 maka banyaknya langkah penyelesaian dari algoritma *Quick Sort* adalah 21 kali perbandingan.

Implementasi *Quick Sort* dalam Program C++

Pengurutan dengan menggunakan sintak pemrograman C++ dengan urutan sintak dimulai dari melakukan deklarasi variabel array untuk banyaknya iterasi tergantung dari jumlah data yang diinput dan sintak lebih lanjut dapat dilihat pada gambar 4.

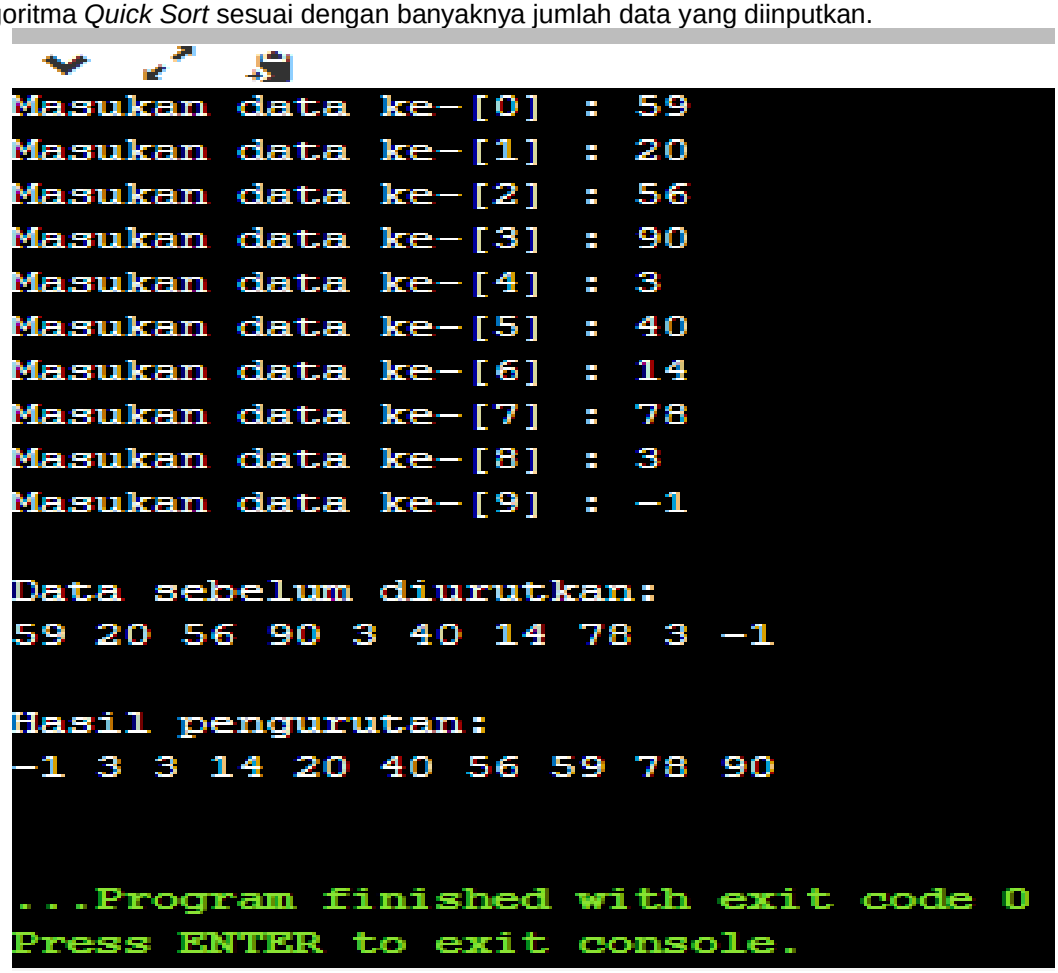
main.cpp | RTAQuickSort | RTABubbleSort

```
#include <conio.h>
using namespace std;
int i,n,data[50];
//fuction quicksort
void quicksort(int data[], int l, int r)
{
      int i = l, j = r;
      int perm;
      int pivot = data[(l+r)/2];
/* peritah partisi */
  while (i<j){
    while (data[i] < pivot)     i++;
    while (data[j] > pivot)     j--;
    if (i<=j){
     perm = data[i];
     data[i] = data[j];
     data[j] = perm;
     i++;j--; };
         };
 /* rekursif */
      if (l < j)
      quicksort(data, l, j);
      if (i < r)
      quicksort(data, i, r);
}
int main(){
cout<<"Masukan banyak data= ";cin>>n;
for(i=0;i<n;i++)
{cout<<"Masukan data ke-["<<i<<"] : ";cin>>data[i];}
cout<<"\nData sebelum diurutkan: "<<endl;
for(i=0;i<n;i++)   {
cout<<data[i]<<" ";
}cout<<"\n";
//hasil pengurutan
quicksort(data,0,n-1);
//hasil pengurutan
cout<<"\nHasil pengurutan:\n";
{
int i;
for (i=0;i<n;i++)
cout<<data[i]<<" ";    cout<<"\n";
}getch();
}
```

Sumber: Hasil Penelitian 2020

Gambar 4. Sintak program *Quick Sort*

Setelah dirun maka tampilan output dapat dilihat pada gambar 5, dengan tampilan masukan banyaknya data adalah variabel n dan jumlah bilangan yang diinput untuk diurutkan dengan algoritma *Quick Sort* sesuai dengan banyaknya jumlah data yang diinputkan.



Sumber: Hasil Penelitian (2020)

Gambar 5. Tampilan luaran pogram *Quick Sort*

4. Kesimpulan

Berdasarkan hasil dan pembahasan maka dapat disimpulkan bahwa: a). Algoritma metode pengurutan *Bubble Sort* dari sisi langkah penyelesaian melakukan 7 putaran dan setiap putaran melakukan 10 kali perbandingan antara n dengan n-1 sampai urutan data terurut secara maksimal. Sedangkan pengurutan *Quick Sort* langkah penyelesaiannya menghasilkan 2 partisi dan rekursi 21 kali . b). Jika dilihat dari penggunaan memori maka pengurutan dengan *Bubble Sort* lebih besar dibanding dengan *Quick Sort* hal ini dapat dilihat dari banyaknya jumlah baris sintak program, untuk *Bubble Sort* berjumlah 52 baris sintak, sedangkan untuk *Quick Sort* 44 baris hal ini mempengaruhi kecepatan komputer dalam membaca perintah program.

Referensi

[1] A. Nugroho, *Algoritma Struktur Data Dengan C#*. Yogyakarta: Cv. Andi Offset, 2009.
[2] E. Utami And Sukrisno, *10 Langkah Belajar Logika Dan Algoritma Menggunakan Bahasa C Dan C++ Di Gnu/Linux*. Yogyakarta: Cv Andi Offset, 2005.
[3] L. Sitorus, *Algoritma Dan Pemrograman*. Yogyakarta: Cv. Andi Offset, 2015.
[4] A. . Rosa And M. Shalahuddin, *Modul Pembelajaran Algoritma Dan Pemrograman*. Bandung: Penerbit Modula, 2010.
[5] A. Sonita And F. Nurtaneo, “Bubble Sort , *Merge Sort* , Dan *Quick Sort* Dalam Proses Pengurutan,” *J. Pseudocode*, Vol. Ii, No. September, Pp. 75–80, 2015.
[6] R. Munir, *Algoritma Dan Pemrograman Dalam Bahasa Pasal Dan Bahasa C*, Revisi.

Bandung: Informatika, 2011.
[7] P. A. Rahayuningsih, “Analisis Perbandingan Kompleksitas Algoritma Pengurutan Nilai (Sorting) Panny,” *Rev. Bras. Ergon.*, Vol. 9, No. 2, P. 10, 2016.
[8] E. Pujiatiningsih, B. Siswoyo, And R. Haviani, “Analisis Perbandingan Algoritma Metode Pengurutan Quicksort, Metode Pengurutan Selectionsort Dan Metode Pengurutan Heapsort,” Vol. 3, No. 2, Pp. 54–67, 2015.
[9] M. E. Al Rivan, “Perbandingan Performa Kombinasi Algoritma Pengurutan Quick-*Insertion sort* Dan Merge-*Insertion sort*,” *Annu. Res. Semin. 2016*, Vol. 2, No. 1, Pp. 6–10, 2016.
[10] M. S. Islam, P. Studi, D. T. Informatika, J. G. Hilir, And D. Ciwaruga, “Quicksort : Metode Pengurutan Array Satu Dimensi Yang Cepat Dan Efisien (Quicksort : Quick And Efficient One-Dimension Array Sorting Method),” Pp. 1–7.
[11] E. Retnoningsih, “Algoritma Pengurutan Data (Sorting) Dengan Metode *Insertion sort* Dan *Selection sort*,” *Inf. Manag. Educ. Prof.*, Vol. 3, No. 1, Pp. 95–106, 2018.
[12] Frieyadie, *Panduan Pemrograman C++*. Yogyakarta: Andi Offset, 2009.
[13] R. Munir And L. Lidya, *Algoritma Dan Pemrograman Dalam Bahasa Pascal, C, Dan C++*. Bandung, 2016.