Jurnal Algoritme, Vol.4, No.1, Oktober 2023, pp. 13~20
E-ISSN: 2775-8796
DOI: 10.35957/algoritme.xxxx

Perbandingan Penempatan *Pivot* Pada *Quick Sort* Berdasarkan Ukuran Pemusatan Data

Rheza Rijaya*[1], Muhammad Ezar Al Rivan[2]
[1,2]Universitas Multi Data Palembang; Jl. Rajawali No.14 Palembang, 0711-376400
Program Studi Informatika, Universitas Multi Data Palembang, Palembang
e-mail: *[1]rhezarijaya@mhs.mdp.ac.id, [2]meedzhar@mdp.ac.id

Abstrak

Sorting atau pengurutan adalah salah satu algoritma dasar yang sangat sering dieksekusi dalam suatu program. Algoritma sorting yang paling umum digunakan dalam suatu program adalah algoritma Quick Sort karena lebih cepat dalam sebagian besar skenario daripada algoritma lainnya. Namun pemilihan pivot pada Quick Sort sangat penting untuk menghindari worst case. Dalam penelitian ini akan diuji penempatan pivot yang umum digunakan (awal, tengah, akhir) dan pivot berdasarkan ukuran pemusatan data (mean, median, modus). Data yang diuji adalah data acak (berulang), acak (permutasi), terurut, terurut terbalik, dan hampir terurut. Jumlah data yang diuji adalah 1.000, 10.000, 100.000, dan 1.000.000. Hasil terbaik didapatkan oleh pivot pada tengah array berdasarkan waktu eksekusi pada setiap skenario.

Kata kunci— *Pivot, Quick Sort*, Ukuran Pemusatan

Abstract

Sorting is one of the basic algorithm that executed frequently in a program. The most popular sorting algorithm is Quick Sort because it is faster in most scenarios than other algorithms. However, pivot selection on Quick Sort algorithm is very important to avoid the worst case scenario. This study aims to test commonly used pivot selection methods (first, middle, last) and pivot selection based on central tendency of data (mean, median, mode). The data that is tested are random data (repeated), random data (permutation), sorted, reverse-sorted, and almost sorted. The size of data that is tested are 1.000, 10.000, 100.000, dan 1.000.000. The best result is achieved by selecting middle element as pivot based on the execution time of each scenario.

Keywords— *Central Tendency, Pivot, Quick Sort*

1. PENDAHULUAN

Sorting atau pengurutan adalah salah satu algoritma dasar yang sangat sering dieksekusi dalam suatu program, karena dengan melakukan *sorting* pada kumpulan data dapat mempermudah pemrosesan data tersebut. Contoh pada kehidupan nyata adalah rak buku. Jika rak buku tersebut berantakan dan buku disusun tidak berdasarkan abjad atau semacamnya, maka mencari buku tertentu akan lebih sulit. Sehingga proses *sorting* sangat berguna dalam mengurangi kompleksitas masalah serta meningkatkan efisiensi pekerjaan.

Received Sept11th,2023; Revised Sept23th, 2023; Accepted Ocotber 6th, 2023

Untuk melakukan *sorting*, terdapat banyak jenis algoritma yang dapat digunakan, dengan berbagai macam karakteristik dan strateginya. Namun algoritma sorting yang paling umum digunakan dalam suatu program adalah algoritma *Quick Sort*. Sesuai dengan namanya yang berarti cepat, algoritma ini lebih cepat dalam sebagian besar skenario daripada algoritma lainnya seperti *mergesort* atau *heapsort*, walaupun memiliki *worst case complexity* yang lebih buruk daripada algoritma *mergesort* atau *heapsort*. Berdasarkan hasil pengujian pada bahasa pemrograman *Python*, algoritma *Quick Sort* menjadi algoritma tercepat jika diuji dengan algoritma *Bubble Sort, Insertion Sort, Merge Sort,* dan *Selection Sort*, walaupun menggunakan memori yang sedikit lebih tinggi [1].

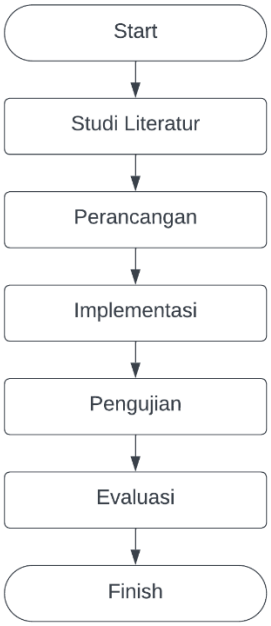
Untuk mendapatkan performa yang maksimal pada algoritma *Quick Sort*, dapat dilakukan dengan beberapa cara seperti menggabungkan *Quick Sort* dengan algoritma lain, atau cara yang paling mudah adalah memilih *pivot* yang tepat. Berdasarkan penelitian yang dilakukan dengan menggabungkan *Quick Sort* dengan *Insertion Sort* didapatkan peningkatan performa rata-rata hingga 15% [2]. Penelitian lain yang dilakukan dengan menggabungkan *Quick Sort* dan *Insertion Sort* juga menghasilkan performa *sorting* data acak dalam jumlah besar (>100000) yang lebih baik daripada *Merge Sort* yang digabung dengan algoritma lain, walaupun *Merge Sort* yang digabung lebih baik pada kasus data yang berjumlah sedikit, kondisi data terurut terbalik, dan kondisi data hampir terurut [3].

Kemudian, pemilihan pivot sangat penting untuk menghindari *worst case*. Contohnya jika *pivot* yang dipilih adalah elemen terkecil atau elemen terbesar dalam *array*, maka perbandingan akan dilakukan terhadap setiap elemen, dan kasus terburuknya jika pemilihan *pivot* tersebut berulang sampai seluruh elemen menjadi *pivot*. Namun *best case* akan terjadi jika *pivot* dapat membagi dua *array* tersebut, sehingga perbandingan akan dibagi dua terus pada setiap rekursi nya [4]. Terdapat beberapa cara umum untuk memilih *pivot*, yaitu pada elemen pertama, elemen di tengah, dan elemen terakhir. Namun pada beberapa kasus, pemilihan *pivot* tersebut masih dapat mengakibatkan *worst case*. Sehingga metode lain seperti mengambil nilai pemusatan data (*median of three, median of five,* dsb*)* dapat dilakukan untuk memastikan algoritma ini lebih kecil kemungkinannya untuk mendapatkan *worst case* [5].

Berdasarkan penjelasan di atas, penelitian ini bertujuan untuk menguji penempatan *pivot* pada umumnya (elemen pertama, tengah, dan akhir) dengan *pivot* berdasarkan nilai pemusatan data (rata-rata, median, dan modus). Sehingga diharapkan dengan pemilihan *pivot* berdasarkan ukuran pemusatan dapat meningkatkan performa *quick sort*.

2. METODE PENELITIAN

Penelitian ini terdiri dari beberapa tahapan, yang dimulai dari studi literatur, perancangan algoritma, implementasi, pengujian, dan evaluasi hasil. Tahapan tersebut dapat dilihat pada Gambar 1.



Gambar 1. Tahapan Penelitian

2.1. Studi Literatur

Pada tahapan studi literatur akan dipelajari buku serta jurnal terkait mengenai algoritma *Quick Sort* dan pemilihan *pivot* berdasarkan ukuran pemusatan data. Diharapkan dengan tahapan studi literatur dapat memperkuat landasan penelitian ini.

2.1.1. Algoritma Sorting

Algoritma *sorting* adalah algoritma yang dapat menyimpan suatu *array* dalam urutan tertentu misalnya secara menaik atau menurun. Peningkatan efisiensi dalam algoritma *sorting* dapat meningkatkan kecepatan pemrosesan data. Algoritma *sorting* diukur efisiensinya menggunakan *Big-O Notation*, yang menyatakan kerumitan algoritma tersebut terhadap ukuran inputnya [6].

2.1.2. Big-O Notation

Big-O Notation adalah cara untuk menyatakan laju pertumbuhan suatu fungsi berdasarkan ukuran inputnya. Notasi ini dapat didefinisikan secara sederhana sebagai berikut:

Jika n adalah ukuran masukan dari $f(n)$, dan $g(n)$ adalah fungsi positif dari n, maka $f(n)$ adalah $O(g(n))$ jika dan hanya jika terdapat konstanta positif c dan bilangan bulat positif n_0 sedemikian rupa sehingga $f(n) \leq c \times g(n)$ untuk semua $n \geq n_0$. Dalam kata lain untuk laju pertumbuhan waktu eksekusi algoritma, untuk ukuran input n yang lebih besar daripada n_0 dan konstanta c, maka waktu eksekusinya tidak akan melebihi $c \times f(n)$. Contohnya, algoritma *insertion sort* memiliki kompleksitas waktu $O(n^2)$. Perlu diketahui juga bahwa dalam *big O-Notation* yang diambil hanya suku dengan orde tertinggi pada fungsi pertumbuhannya [7].

2.1.3. Quick Sort

Algoritma *quick sort* adalah algoritma sorting yang dikembangkan oleh Tony Hoare pada tahun 1959 dan dipublikasikan pada tahun 1961. Algoritma ini menggunakan strategi *divide and conquer*, dimana menggunakan "*pivot*" untuk membagi dua *array* tersebut dengan memindahkan elemen yang lebih kecil dari *pivot* ke sebelah kiri *pivot* dan elemen yang lebih besar dari *pivot* ke sebelah kanan *pivot*, dan proses ini dilakukan secara rekursif pada *array* yang telah dibagi tersebut. Algoritma ini memiliki kompleksitas *worst case* $O(n^2)$ dan kompleksitas *average case* $O(n \log n)$ [8].

Kelebihan dari *quick sort* adalah cepat dan efektif, memiliki kompleksitas waktu yang lebih baik daripada algoritma lain walau secara matematis yaitu $O(n^2)$, dan memiliki kompleksitas ruang $O(\log n)$. Namun kekurangannya antara lain *unstable* (urutan dua elemen yang sama akan berbeda setelah sorting) dan ketika *pivot* yang dipilih adalah elemen terkecil atau terbesar akan menimbulkan *worst case scenario* [9].

2.1.4. Ukuran Pemusatan

Ukuran pemusatan atau ukuran tendensi pusat adalah suatu nilai yang dapat mewakili sekumpulan data. Nilai ini merupakan cerminan atau gambaran secara umum dari sekumpulan data. Terdapat beberapa bentuk ukuran pemusatan, antara lain [10]:

1. Rata-rata hitung (*mean*)

Rata-rata hitung adalah jumlah dari seluruh nilai data dibagi banyak data tersebut. Untuk data yang belum dikelompokkan, persamaan dari rata-rata sampel dan rata-rata populasi dapat dilihat pada persamaan (1) dan (2).

a. Rata-rata sampel

$$\bar{x} = \frac{\sum x_i}{n}, i = 1, 2, 3, .., n \ (1)$$

b. Rata-rata populasi

$$\mu = \frac{\sum x_i}{N}, i = 1, 2, 3, .., N \ (2)$$

2. Median

Median dari suatu data adalah nilai yang letaknya tepat di tengah-tengah data jika jumlah data adalah ganjil atau rata-rata dua nilai di tengah jika jumlah data genap. Perhitungan median mengharuskan untuk mengurutkan data terlebih dahulu. Dalam kata lain, median membagi sekelompok data menjadi dua bagian yang sama, sehingga 50% data berada di bawah nilai median dan 50% data terletak di atas nilai median. Untuk data yang memiliki nilai data yang sangat besar atau kecil, median lebih mewakili kumpulan data tersebut dibandingkan ukuran pemusatan lainnya.

Cara menghitung median untuk data yang belum dikelompokkan adalah sebagai berikut:

1. Urutkan data tersebut baik secara menaik atau menurun
2. Tentukan letak median (L_{Md})
   - Bila jumlah data ganjil (n ganjil), maka L_{Md} terletak di data $\frac{n+1}{2}$
   - Bila jumlah data genap (n genap), maka L_{Md} diantara data ke $\frac{n}{2}$ dan $\frac{n+2}{2}$
3. Hitung nilai median
   - Bila jumlah data ganjil (n ganjil), maka gunakan persamaan (3)

$$Md = X_{\frac{n+1}{2}} \ (3)$$

   - Bila jumlah data genap (n genap), maka gunakan persamaan (4)

$$Md = \frac{\left(X_{\frac{n}{2}} + X_{\frac{n+2}{2}}\right)}{2} \ (4)$$

3. Modus

Modus adalah nilai yang paling banyak muncul pada sekumpulan data. Pada data kuantitatif, modus menunjukkan nilai yang paling banyak muncul. Pada data kualitatif, modus menunjukkan sifat atau keadaan yang paling banyak terjadi. Sehingga dalam sekumpulan data, mungkin saja tidak mempunyai modus, memiliki satu modus, atau memiliki dua modus atau lebih.

Untuk menentukan modus dari data yang belum dikelompokkan adalah sebagai berikut:

1. Hitung frekuensi masing-masing data
2. Data yang memiliki frekuensi terbesar merupakan modus dari kumpulan data tersebut

2.2. Perancangan

Pada penelitian ini, bahasa pemrograman yang digunakan adalah *Java*. Versi JDK yang digunakan adalah JDK 19 dengan *target compile Java* versi 8. Terdapat 3 macam data yang memerlukan bilangan acak, yaitu data angka hampir terurut, data angka acak non permutasi, dan data acak permutasi. Sehingga untuk memastikan bahwa data selalu sama, perlu dibuat *file* untuk menyimpan data tersebut. Data tersebut dibuat menggunakan program dengan bahasa pemrograman *Python* dan disimpan dalam bentuk JSON *array*. Untuk data hampir terurut, jumlah pertukaran secara acak adalah sebanyak 10 persen dari jumlah data.

2.3. Implementasi

Pada tahap implementasi, rancangan program yang telah dibuat akan dijadikan kode program yang digunakan untuk pengujian pada tahap selanjutnya. Implementasi algoritma *Quick Sort* dilakukan dengan konsep OOP untuk mengurangi redundansi kode yang berulang pada setiap jenis pivot yang diuji. Untuk penempatan *pivot* dengan ukuran pemusatan, akan dilakukan dengan kondisi di bawah ambang batas / *threshold* panjang *array* pada partisi tersebut. Jika panjang partisi masih lebih besar dari ambang batas maka akan digunakan *pivot* pada tengah *array*. Hal ini dilakukan karena perhitungan ukuran pemusatan pada *array* yang besar memiliki dampak yang cukup signifikan pada waktu eksekusi.

2.4. Pengujian

Pada tahap pengujian, waktu eksekusi pada setiap skenario jenis *pivot* dan jumlah data akan diuji. Karena jumlah data pada beberapa skenario cukup banyak, maka akan diberikan argumen Java pada saat memulai program untuk meningkatkan *stack size* dan menghindari *StackOverflowError*. *Stack size* akan diatur menjadi 96MB dengan *flag* -Xss96M, memori awal diatur menjadi 128 MB dengan *flag* -Xms128M, dan memori maksimal diatur menjadi 1024MB dengan *flag* -Xmx1024M. Pengujian dilakukan dengan perangkat yang berspesifikasi prosesor Intel Core i5 8300H, RAM 16GB, dan sistem operasi Windows 11. Hasil pengujian ini adalah waktu eksekusi untuk setiap skenario dalam satuan nanodetik. Untuk memastikan pengujian yang konsisten, program yang berjalan di latar belakang akan ditutup untuk mengurangi penggunaan sumber daya oleh program lain.

2.5. Evaluasi

Pada tahap evaluasi, akan dianalisis hasil waktu eksekusi pada setiap skenario untuk menentukan performa setiap jenis *pivot* pada berbagai skenario jenis data dan jumlah data

3. HASIL DAN PEMBAHASAN

Berikut adalah hasil waktu eksekusi program yang telah dirancang pada setiap skenario penempatan *pivot*, jenis data, dan jumlah data dalam satuan nanosekon.

Tabel 1. Hasil Pengujian Untuk Data Acak Non Permutasi

Penempatan Pivot / Jumlah Data	**1000**	**10000**	**100000**	**1000000**
Awal	92,500	674,100	8,109,700	93,272,200
Tengah	96,100	683,600	8,077,800	93,529,400
Akhir	85,200	597,900	6,938,001	79,753,900
Mean (batas 1000)	206,101	1,235,300	13,492,700	145,779,300
Mean (batas 10000000)	205,500	1,562,000	19,475,400	235,327,701
Median (batas 1000)	391,200	2,332,700	23,560,400	286,406,200
Median (batas 10000000)	375,300	2,804,799	38,515,101	491,260,501

Modus (batas 1000)	1,918,099	9,716,299	67,975,801	741,628,099
Modus (batas 10000000)	1,773,500	10,445,200	147,051,500	3,292,336,999

Pada Tabel 1, didapatkan hasil pengujian untuk data acak non permutasi, hasil terbaik didapat dengan menggunakan *pivot* akhir, dan pada ukuran pemusatan yang terbaik adalah *pivot mean* dengan batas 1000.

Tabel 2. Hasil Pengujian Untuk Data Acak Permutasi

Penempatan Pivot / Jumlah Data	**1000**	**10000**	**100000**	**1000000**
Awal	73,000	680,600	7,934,000	94,618,000
Tengah	85,800	589,100	6,888,800	81,442,400
Akhir	86,800	600,900	7,014,600	81,262,500
Mean (batas 1000)	205,500	1,256,000	13,565,100	146,405,100
Mean (batas 10000000)	205,300	1,577,300	19,512,700	232,421,300
Median (batas 1000)	281,200	2,364,300	23,525,800	269,991,700
Median (batas 10000000)	377,000	2,898,100	38,716,500	479,298,500
Modus (batas 1000)	1,305,900	6,132,900	63,811,500	746,046,100
Modus (batas 10000000)	1,075,000	8,352,600	137,846,200	2,695,117,300

Pada Tabel 2, didapatkan hasil pengujian untuk data acak permutasi, hasil terbaik didapat dengan menggunakan *pivot* awal, dan pada ukuran pemusatan yang terbaik adalah *pivot mean* dengan batas 1000.

Tabel 3. Hasil Pengujian Untuk Data Terurut

Penempatan Pivot / Jumlah Data	**1000**	**10000**	**100000**	**1000000**
Awal	466,700	23,231,000	2,164,721,300	216,408,690,200
Tengah	18,900	154,700	2,171,700	19,844,700
Akhir	516,000	27,524,800	2,676,299,700	280,755,903,600
Mean (batas 1000)	125,400	1,123,500	8,278,300	87,157,300
Median (batas 1000)	1,081,300	7,515,000	72,396,800	759,330,800
Modus (batas 1000)	17,265,600	109,151,600	1,444,280,600	15,285,410,000

Pada Tabel 3, didapatkan hasil pengujian untuk data terurut, hasil terbaik didapat dengan menggunakan *pivot* tengah, dan pada ukuran pemusatan yang terbaik adalah *pivot mean* dengan batas 1000. Pada pengujian dengan data terurut, batas untuk ukuran pemusatan hanya dipakai 1000 karena untuk jumlah 10.000.000 memerlukan waktu sangat lama pada data yang banyak.

Tabel 4. Hasil Pengujian Untuk Data Terurut Terbalik

Penempatan Pivot / Jumlah Data	**1000**	**10000**	**100000**	**1000000**
Awal	555,200	28,155,300	2,609,073,700	259,765,927,000
Tengah	18,000	260,400	2,778,300	19,859,100
Akhir	418,300	20,788,700	2,021,659,900	213,056,483,100
Mean (batas 1000)	136,200	1,323,900	8,919,800	87,123,800
Median (batas 1000)	1,186,500	6,282,500	72,192,100	754,014,500
Modus (batas 1000)	15,315,200	95,607,100	1,173,718,700	15,004,122,600

Pada Tabel 4, didapatkan hasil pengujian untuk data terurut terbalik, hasil terbaik didapat dengan menggunakan *pivot* tengah, dan pada ukuran pemusatan yang terbaik adalah *pivot mean* dengan batas 1000. Pada pengujian dengan data terurut terbalik, batas untuk ukuran pemusatan hanya dipakai 1.000 karena untuk jumlah 10.000.000 memerlukan waktu sangat lama pada data yang banyak.

Tabel 5. Hasil Pengujian Untuk Data Hampir Terurut

Penempatan Pivot / Jumlah Data	**1000**	**10000**	**100000**	**1000000**
Awal	34,700	339,500	3,771,600	45,390,100
Tengah	43,400	267,100	3,122,200	35,359,500
Akhir	82,300	440,500	5,206,700	70,649,900
Mean (batas 1000)	101,800	896,400	9,659,000	101,772,600
Mean (batas 10000000)	159,600	1,273,900	15,254,000	184,092,100
Median (batas 1000)	311,600	1,874,800	18,895,400	206,341,800
Median (batas 10000000)	335,600	2,618,400	40,663,800	485,193,600
Modus (batas 1000)	1,074,100	13,958,000	115,321,700	1,084,393,400
Modus (batas 10000000)	1,232,800	10,338,200	177,361,900	2,896,082,200

Pada Tabel 5, didapatkan hasil pengujian untuk data hampir terurut, hasil terbaik didapat dengan menggunakan *pivot* awal, dan pada ukuran pemusatan yang terbaik adalah *pivot mean* dengan batas 1000.

Dari seluruh pengujian yang telah dilakukan, hasil terbaik berdasarkan waktu eksekusi didapatkan oleh *pivot* pada tengah *partition*. Untuk *pivot* berdasarkan ukuran pemusatan, yang terbaik adalah *pivot mean*, walau memerlukan waktu yang sedikit lebih lama karena perlu perhitungan untuk *mean*-nya. Penambahan batas jumlah untuk menggunakan ukuran pemusatan akan cenderung meningkatkan waktu eksekusi. Sehingga pemilihan batas harus lebih dioptimalkan agar performanya bisa konsisten.

4. KESIMPULAN

Berdasarkan hasil pengujian pada Tabel 1, 2, 3, 4, dan 5, penggunaan *pivot* dengan ukuran pemusatan masih perlu dilakukan optimisasi agar waktu untuk menghitung ukuran pemusatan lebih sedikit. Namun untuk performa yang paling mendekati *pivot* awal, tengah, atau akhir adalah *pivot* mean. Batas jumlah untuk perhitungan ukuran pemusatan perlu dijaga sekecil mungkin untuk mengurangi waktu perhitungan. Dengan menggunakan *pivot* ukuran pemusatan, terutama *pivot* mean dapat menghindari dari *worst case* pada pengujian ini walau memerlukan waktu yang lebih lama daripada pemilihan *pivot* secara umumnya.

5. SARAN

Penelitian lebih lanjut dapat dilakukan agar algoritma untuk perhitungan *pivot* berdasarkan ukuran pemusatan dapat lebih efisien dan dapat mendekati waktu eksekusi pemilihan pivot secara statis (awal, tengah, akhir).

DAFTAR PUSTAKA

[1] Y. Heryanto and T. Wira Harjanti, "Analisis Perbandingan Ruang dan Waktu pada Algoritma Sorting Menggunakan Bahasa Pemrograman Python," *Jurnal Penerapan Sistem Informasi (Komputer & Manajemen)*, vol. 4, no. 2, pp. 342–347, Apr. 2023.

[2] M. E. Al Rivan, "Perbandingan Performa Kombinasi Algoritma Pengurutan Quick-Insertion Sort dan Merge-Insertion Sort," *Annual Research Seminar (ARS)*, vol. 2, no. 1, pp. 6–10, Dec. 2016, [Online]. Available: http://ars.ilkom.unsri.ac.id6

[3] M. E. Al Rivan, "Perbandingan Kecepatan Gabungan Algoritma Quick Sort Dan Merge Sort Dengan Insertion Sort, Bubble Sort Dan Selection Sort," *Jurnal Teknik Informatika*

dan Sistem Informasi, vol. 3, no. 2, pp. 319–331, Aug. 2017, doi: https://doi.org/10.28932/jutisi.v3i2.675.

[4] V. Iliopoulos, "The Quicksort algorithm and related topics," 2015. [Online]. Available: http://arxiv.org/abs/1503.02504

[5] A. Mohammed and M. Othman, "Comparative Analysis of Some Pivot Selection Schemes for Quicksort Algorithm," *Information Technology Journal*, vol. 6, no. 3, pp. 424–427, 2007.

[6] R. Maulana, "Analisa Perbandingan Kompleksitas Algoritma Selectionsort Dan Insertionsort," *INFORMATIKA*, vol. 3, pp. 208–218, Sep. 2016.

[7] Subandijo, "Efisiensi Algoritma Dan Notasi O-Besar," *ComTech*, vol. 2, no. 2, pp. 849–858, 2011.

[8] Y. Yao, *A Detailed Analysis of Quicksort Algorithms with Experimental Mathematics*. 2019. [Online]. Available: http://sites.math.rutgers.edu/~yao/QuickSort.html

[9] H. Patel, "An Overview of QuickSort Algorithm," 2022. https://towardsdatascience.com/an-overview-of-quicksort-algorithm-b9144e314a72 (accessed Jun. 06, 2023).

[10] N. Wirawan, *Cara Mudah Memahami Statistika Ekonomi dan Bisnis (Statistika Deskriptif)*, vol. 1. Denpasar: Keraras Emas, 2016.