Algoritma *Quick Sort*

Oleh

233406008 YOHANES EKA PUTRA

233401016 JEFFEREY CHEALFIRO I.

Program Studi Teknik Informatika

Fakultas Teknik

Universitas Katolik Darma Cendika

2024

Latar Belakang

Di tahun 2024 teknologi telah berkembang pesat banyak informasi dengan mudah didapatkan. Data adalah bagian kecil sebelum diolah menjadi sebuah informasi. Masalah terjadi ketika data yang disajikan tidak berurutan, agar menjadi data yang tersusun berurutan digunakannya metode pengurutan. Dalam penelitian ini peneliti ingin menganalisis algoritma *quick sort*. Tony Hoare adalah penemu algoritma *quick sort* pada tahun 1959 dan 2 tahun setelah itu Tony Hoare menerbitkannya [1]. Algoritma *quick sort* adalah algoritma yang membandingkan setiap elemennya dengan pivot atau disebut elemen yang dijadikan acuannya dan melakukan penukaran posisi pada elemenya tergantung pada perbandingan setiap elemennya dengan pivot setelah itu sekumpulan elemen tersebut dipecah menjadi bagian yang lebih kecil dan didapatkan elemen yang lebih kecil dari pivot berada di sebelah kiri pivot dan elemen yang lebih besar dari pivot berada di sebelah kanan pivot, hal ini dilakukan terus menerus hingga elemennya tersusun secara berurutan [2]. Algoritma *quick sort* memiliki kompleksitas yang berbeda-beda setiap kasusnya pada kasus rata rata adalah $O(n \log n)$ dan pada kasus terburuknya adalah $O(n^2)$ [3]. Algoritma *quick sort* mampu menyelesaikan masalah pada data yang tidak tersusun secara berurutan.

Landasan Teori

Data

Data adalah sekumpulan fakta yang dapat berupa angka, gambar, tulisan yang dapat diolah menjadi sebuah informasi [4].

Algoritma

Algoritma adalah sebuah instruksi yang logis dan disusun secara terstruktur yang digunakan untuk memecahkan sebuah masalah yang terjadi [5].

Sejarah

Tony Hoare penemu *quick sort* mendapatkan tawaran untuk mengerjakan proyek untuk menerjemahkan bahasa dari Rusia ke Inggris ketika menempuh pendidikan di Moscow State

Univeristy. Tony Hoare memerlukan cara agar dapat mengurutkan kata dalam kalimat ke dalam abjad sebelum di terjamahkan karena kamusnya disimpan di sebuah *magnetic tape*. Singkat cerita Tony Hoare menggunakan metodenya yaitu *quick sort* tetapi tidak berhasil mengimplementasikannya karena keterbatasan pada bahasa yang digunakan. Pada tahun 1961, Tony Hoare mempelajari bahasa ALGOL 60 dan dapat menggunakan cara rekursif yaitu kemampuan sebuah fungsi yang memanggil dirinya sendiri karena hal tersebut Tony Hoare dapat membuat program pengurutan yang dinamakan *quick sort* [6].

Metode *Quick sort*

Algoritma

1. Menentukan pivot. Pivot dapat ditentukan pada bagian awal, tengah, atau akhir tetapi pada contoh ini pivot ditentukan pada bagian akhir array.
2. Inisialisasi variabel pertama sebagai index awal
3. Inisialisasi variabel kedua (index awal - 1)
4. Melakukan perbandingan, jika value variabel pertama lebih besar dari pivot, tambahkan 1 pada index variabel pertama. Jika value variabel pertama lebih kecil dari pivot tambahkan 1 pada index variabel kedua setelah itu melakukan perbandingan, jika value variabel kedua lebih kecil dari value variabel pertama, value variabel pertama dan value variabel kedua saling menukar posisi. Jika value variabel pertama sama dengan value variabel kedua, tambahkan 1 pada index variabel pertama. Selanjutnya jika kondisi value variabel pertama lebih kecil dari pivot tidak terpenuhi hingga variabel pertama menepati index akhir maka index variabel kedua akan ditambahkan 1 dan melakukan pertukaran dengan pivot.
5. Sehingga membentuk 2 partisi atau bagian, elemen yang lebih kecil dari pivot berada pada bagian sebelah kiri pivot dan elemen yang lebih besar dari pivot berda pada bagian sebelah kanan pivot.
6. Mengulangi langkah (1) sampai (6) agar mendapatkan susunan yang berurutan pada setiap elemen di dalam array.

Pseudo code

```
Pseudocode algoritma quick sort

deklarasi :
    pivot = arr[high]
    j = 0
    i = -1

implementasi :

    main :
        array = [data yang belum terurut]
        quicksort(array, 0, array.length - 1)


    deklarasi fungsi quicksort(arr, low, high):
        if low < high:
            n = partisi(arr, low, high)

            quicksort(arr, low, n - 1)
            quicksort(arr, n + 1, high)

    deklarasi fungsi partisi(arr, low, high):
        for j range(low, high)
            if arr[j] <= pivot
              i = i + 1

              (arr[i], arr[j]) = (arr[j], arr[i])
            end loop
        (arr[i + 1], arr[high]) = (arr[high], arr[i + 1])
        return i + 1
```

Gambar 1 - Pseudocode algoritma quick sort

Cara kerja algoritma *quick sort*

2	**4**	**1**	**5**	**3**

Gambar 2 - Input yang belum berurutan

Sebagai contoh diberikan input pada sebuah array seperti gambar 2. Pertama yang dilakukan adalah menentukan pivot, nilai pivot berdasarkan data yang diberikan adalah 3. Selanjutnya memberikan index dan inisialisasi variabel, variabel I = index awal – 1 variabel J = index awal.

i 0	i 1	i 2	i 3	i 4
2	**4**	**1**	**5**	**3**
J = i 0 I = i 0 – 1				PIVOT

Gambar 3 - Hasil setelah diberikan index dan inisiliasisasi pivot serta variabel

Setelah mendapatkan letak pivot beserta index dan variabelnya, melakukan perbandingan antara data J dan PIVOT seperti yang tertera pada langkah(4), jika data J lebih kecil atau sama dengan(≤) PIVOT, index variabel I ditambahkan 1 diperoleh I menepati posisi i 0 karena sebelumnya i 0 dikurang dengan 1.

i 0	i 1	i 2	i 3	i 4
2	**4**	**1**	**5**	**3**
J = I 0 I = i 0				PIVOT

Gambar 4 - Hasil setalah index variabel I ditambahkan 1

Setelah I menepati posisi i 0, data J dan I melakukan pertukaran posisi karena J dan I berada posisi yang sama, data pada i 0 berada pada posisi tetap. Selanjutnya index J akan bertambah 1 menepati posisi i 1 dan melakukan perbandingan, jika data J lebih besar (>) dari PIVOT, index variabel J akan ditambah 1 hingga kondisi data variabel J lebih kecil atau sama dengan (≤) PIVOT.

i 0	i 1	i 2	i 3	i 4
2	**4**	**1**	**5**	**3**
I = i 0		J = I 2		PIVOT

Gambar 5 - Hasil pada kondisi data variabel J kurang dari atau sama dengan PIVOT

Pada gambar 5 terjadi kondisi data J lebih kecil atau sama dengan (≤) PIVOT. Selanjutnya index variabel I akan bertambah 1, data pada variabel I dan J melakukan pertukaran posisi diperoleh i 1 = 1 dan i 2 = 4.

i 0	i 1	i 2	i 3	i 4
2	**1**	**4**	**5**	**3**
	I = i 1	J = i 2		PIVOT

Gambar 6 - Hasil setelah melakukan pertukaran I dan J pada index i 1 dan i 2

Setelah data pada i 1 dan i 2 melakukan pertukaran posisi, index variabel J akan bertambah 1 hingga mendapatkan kondisi data variabel J lebih kecil atau sama dengan (≤) PIVOT. Kondisi tersebut tidak terpenuhi hingga J menepati i 4 bersamaan dengan PIVOT karena kondisi data J lebih kecil atau sama dengan (≤) PIVOT tidak terpenuhi, PIVOT akan melakukan petukaran posisi dengan data variabel I yang indexnya ditambahkan dengan 1, i 2 = 3 dan i 4 = 4.

i 0	i 1	i 2	i 3	i 4
2	**1**	**3**	**5**	**4**
		PIVOT I = i 2		J = i 4

Gambar 7 - Hasil setelah J menepati index akhir dan PIVOT melakukan pertukaran dengan data variabel I

Proses ini akan berhenti karena variabel J telah menepati i 4 atau index akhir dan proses ini akan mengembalikan sebuah nilai yang digunakan untuk membuat sebuah partisi. Nilai yang dikembalikan adalah index pada variabel I, diperoleh index variabel I adalah 2 maka nilai yang

dikembalikan adalah 2. Selanjutnya data data tersebut akan dibuat menjadi 2 bagian/partisi karena akan dilakukan perbandingan dan pertukaran. Partisi pertama dimulai dari i 0 hingga i 1 atau nilai yang dikembalikan dikurang dengan 1. Partisi kedua dimulai dari i 3 atau nilai yang dikembalikan ditambah dengan 1.

i 0	i 1
2	**1**

Gambar 8 - Partisi 1

i 3	i 4
5	**4**

Gambar 9 - Partisi 2

Pada gambar 8 akan diinisialisasi variabel lagi yaitu variabel I dan J, J = index awal yaitu i 0, I = index awal - 1. Selanjutnya pada gambar 9 akan dinisialisasi variabel yaitu yaitu variabel I dan J, J = index awal yaitu i 3, I = index awal - 1. Pada gambar 8 dan 9 posisi PIVOT berada pada index akhir.

i 0	i 1
2	**1**
J = i 0 I = i 0 - 1	PIVOT

Gambar 10 - Hasil yang telah dinisialisasi variabel dan PIVOT pada partisi 1

i 3	i 4
5	**4**
J = i 3 I = i 3 - 1	PIVOT

Gambar 11 - Hasil yang telah diinisialiasi variabel dan PIVOT pada partisi 2

Selanjutnya pada gambar 10 index variabel J akan bertambah hingga mendapatkan data kondisi J lebih kecil atau sama dengan (≤) PIVOT karena kondisi data J lebih kecil atau sama dengan (≤) PIVOT tidak terpenuhi hingga J menepati index paling akhir maka PIVOT akan melakukan pertukaran posisi dengan data variabel I setelah index variabel I ditambahkan dengan 1.

i 0	i 1
1	**2**
PIVOT I = i 0	J = i 1

Gambar 12 - Hasil setelah index variabel I ditambah 1 dan PIVOT melakukan pertukaran posisi dengan data variabel I pada partisi 1

Pada gambar 12, proses perbandingan akan berhenti karena J telah berada pada index akhir dan akan mengembalikan nilai index I yaitu 0. Selanjutnya proses tersebut akan berhenti karena data pada partisi pertama telah terurut. Partisi kedua akan melakukan perbandingan, pada gambar 11 index variabel J akan bertambah hingga mendapatkan kondisi data J lebih kecil atau sama dengan (≤) PIVOT. Setelah index J melakukan pertambahan setiap kondisi tidak terpenuhi, kondisi tersebut tidak terpenuhi hingga J berada pada index akhir. Selanjutnya PIVOT akan melakukan pertukaran dengan data variabel I setelah index variabel I yang ditambah dengan 1.

i 3	i 4
4	**5**
PIVOT I = i 3	J = i 4

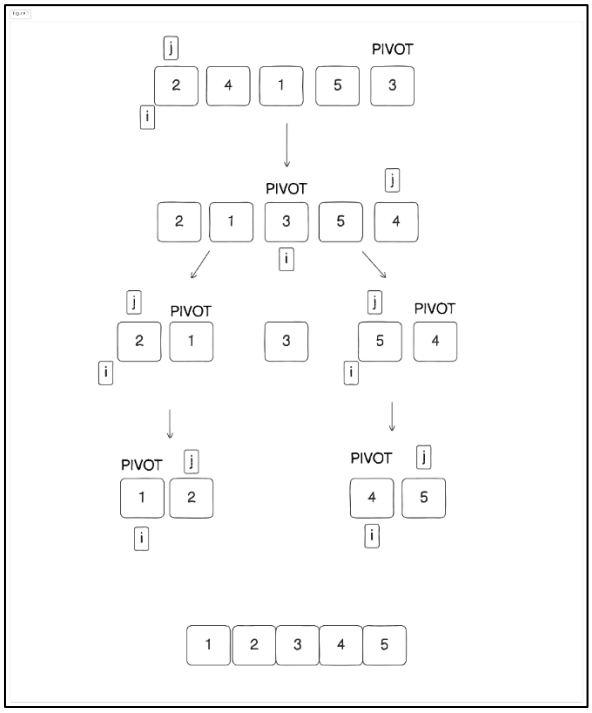
Gambar 13 - Hasil setelah index variabel 1 ditambah 1 dan PIVOT melakukan pertukaran dengan data variabel I pada partisi 2

Setelah proses pengurutan menggunakan algoritma *quick sort* terjadi dikembalikan *output* seperti gambar dibawah

1	2	3	4	5

Gambar 14 - Output yang dikembalikan setelah proses pengurutan

Visualisasi algoritma *quick sort*



Gambar 15 -Visualisasi algoritma quicksort

Algoritma *quick sort*

```
1. if __name__ == '__main__':
2.     my_arr = [2, 4, 1, 5, 3]
3.
4.     panjang = len(my_arr)
5.     quicksort(my_arr, 0, panjang - 1)
6.
7.     for k in my_arr:
8.         print(k, end=" ")
```

Gambar 16 - Program utama untuk memanggil fungsi quick sort

```
1. def bantuan(arr, low, high):
2.
3.     pivot = arr[high]
4.
5.     i = low - 1
6.
7.     for j in range(low, high):
8.         if arr[j] <= pivot:
9.             i = i + 1
10.                (arr[i], arr[j]) = (arr[j], arr[i])
11.
12.     (arr[i + 1], arr[high]) = (arr[high], arr[i + 1])
13.
14.     return i + 1
```

Gambar 17 - Fungsi bantuan

```
1. def quicksort(arr: list, low: int, high: int):
2.     if low < high:
3.         n = bantuan(arr, low, high)
4.
5.         quicksort(arr, low, n - 1)
6.         quicksort(arr, n + 1, high)
```

Gambar 18 - Fungsi quick sort

Kesimpulan

Algoritma pengurutan memiliki berbagai macam variasi dan cara. Algoritma *quick sort* menjadi salah satu algoritma yang efisien karena memiliki kompleksitas rata rata $O(n \log n)$. Algoritma ini juga menggunakan metode rekursif untuk membagi array menjadi array yang lebih kecil. Terlepas dari keunggulan *quick sort*, algoritma *quick sort* memiliki kelemahan ketika data yang diberikan telah berurutan membuat kompleksitas pada *quick sort* menjadi tidak optimal tetapi ketika data yang diberikan tidak berurutan, *quick sort* akan berjalan secara optimal. Oleh karena itu dengan memakai algoritma *quick sort*, algoritma ini dapat membantu dalam mengurutkan sebuah data.

Daftar Pustaka

[1] “Quicksort: the history and implementations,” DEV Community. Accessed: Jun. 08, 2024. [Online]. Available: https://dev.to/snj/quicksort-the-history-and-implementations-28o2

[2] “Quick Sort Algorithm: A Complete Guide | Simplilearn,” Simplilearn.com. Accessed: Jun. 08, 2024. [Online]. Available: https://www.simplilearn.com/tutorials/data-structure-tutorial/quick-sort-algorithm

[3] M. E. A. Rivan, “Perbandingan Kecepatan Gabungan Algoritma Quick Sort dan Merge Sort dengan Insertion Sort, Bubble Sort dan Selection Sort,” *J. Tek. Inform. Dan Sist. Inf.*, vol. 3, no. 2, Art. no. 2, Aug. 2017, doi: 10.28932/jutisi.v3i2.675.

[4] P. Relations, “Pengertian Data, Fungsi, Jenis-jenis, Manfaat dan Contohnya,” Telkom University. Accessed: Jun. 08, 2024. [Online]. Available: https://telkomuniversity.ac.id/pengertian-data-fungsi-jenis-jenis-manfaat-dan-contohnya/

[5] G. G. Maulana, “PEMBELAJARAN DASAR ALGORITMA DAN PEMROGRAMAN MENGGUNAKAN EL-GORITMA BERBASIS WEB,” *J. Tek. Mesin*, vol. 6, no. 2, p. 8, Mar. 2017, doi: 10.22441/jtm.v6i2.1183.

[6] “Tony Hoare >> Contributions >> Quicksort.” Accessed: Jun. 08, 2024. [Online]. Available: https://cs.stanford.edu/people/eroberts/courses/soco/projects/2008-09/tony-hoare/quicksort.html#top