“Implementasi dan Analisis Algoritma Quicksort”

Perancangan dan Analisis Algoritma

Dosen pengampu : Randi Proska Sandra, S.Pd, M.Sc

Kode Kelas : 202423430076

Disusun Oleh :

Ketherin Fathur Rahma

23343072

PROGRAM STUDI INFORMATIKA (NK)
FAKULTAS TEKNIK
UNIVERSITAS NEGERI PADANG
2025

A. Penjelasan Program/Algoritma QuickSort

Quicksort adalah salah satu algoritma pengurutan yang menerapkan konsep Divide and Conquer, di mana sebuah daftar dibagi menjadi dua bagian berdasarkan elemen pivot. Elemen yang lebih kecil dari pivot dipindahkan ke bagian kiri, sedangkan elemen yang lebih besar ditempatkan di bagian kanan. Proses ini berlangsung secara rekursif hingga seluruh elemen tersusun secara terurut.

Keunggulan utama Quicksort adalah kecepatan dan efisiensinya dibandingkan algoritma pengurutan sederhana seperti Bubble Sort dan Selection Sort. Dengan kompleksitas rata-rata O(n log n)**,** algoritma ini sangat efektif untuk menangani data dalam jumlah besar. Namun, dalam kasus terburuk, ketika pemilihan pivot tidak optimal (misalnya selalu memilih elemen terbesar atau terkecil), kompleksitasnya dapat memburuk menjadi $O(n^2)$**.**

Tahapan utama dalam algoritma Quicksort:

1. Menentukan pivot (dapat berupa elemen pertama, terakhir, tengah, atau dipilih secara acak).
2. Memisahkan elemen lainnya menjadi dua bagian:
    - Elemen yang lebih kecil dari pivot.
    - Elemen yang lebih besar dari pivot.
3. Menggunakan rekursi untuk mengurutkan kedua bagian tersebut.
4. Menggabungkan hasilnya sehingga membentuk daftar yang telah terurut.

Dalam praktiknya, Quicksort banyak digunakan di berbagai aplikasi pemrograman, termasuk dalam pengelolaan data besar, sistem pencarian cepat, dan basis data**.** Algoritma ini juga sering diimplementasikan dalam pustaka sorting di berbagai bahasa pemrograman seperti C++, Python, dan Java**.**

Keunggulan utama Quicksort adalah sifatnya sebagai in-place sorting, yang berarti tidak membutuhkan banyak memori tambahan seperti Merge Sort. Namun, salah satu kelemahannya adalah performa yang kurang optimal pada data yang sudah hampir terurut, kecuali jika menggunakan metode pemilihan pivot yang lebih canggih, seperti randomized Quicksort**,** untuk mengurangi kemungkinan skenario terburuk.

B. Pseudocode QuickSort

1. Fungsi pengurutan_cepat

```
FUNGSI pengurutan_cepat(array, kiri, kanan):
JIKA kiri < kanan:
pivotIndex = partisi(array, kiri, kanan)
pengurutan_cepat(array, kiri, pivotIndex - 1)
pengurutan_cepat(array, pivotIndex + 1, kanan)
```

Penjelasan:

- Fungsi ini adalah fungsi utama untuk mengurutkan daftar menggunakan rekursi.
- Jika Kiri masih lebih kecil dari kanan, maka:
    1. Tentukan pivot dengan memanggil fungsi partisi, yang akan menempatkan pivot di posisi yang benar.
    2. Rekursi ke bagian kiri dari pivot (kiri hingga pivotIndex - 1).
    3. Rekursi ke bagian kanan dari pivot (pivotIndex + 1 hingga kanan).

2. Fungsi partisi

```
FUNGSI partisi(array, kiri, kanan):
    pivot = array[kanan]
    i = kiri - 1
    UNTUK j dari kiri ke kanan - 1:
        JIKA array[j] < pivot:
            i = i + 1
            TUKAR array[i] dan array[j]
    TUKAR array[i + 1] dan array[kanan]
    KEMBALIKAN i + 1
```

Penjelasan:

- Fungsi partisi bertanggung jawab untuk menempatkan pivot di posisi yang benar dalam array.
- Langkah-langkahnya:
    1. Pilih pivot → Elemen paling kanan (array[kanan]).
    2. Buat indeks **i** yang menyimpan posisi elemen lebih kecil dari pivot.
    3. Loop dari kiri ke kanan:
        - Jika elemen array[j] lebih kecil dari pivot, maka tukar dengan elemen di indeks i.
    4. Tukar pivot dengan elemen di **i + 1** → Ini menempatkan pivot di posisi yang benar.
    5. Kembalikan indeks pivot yang baru.

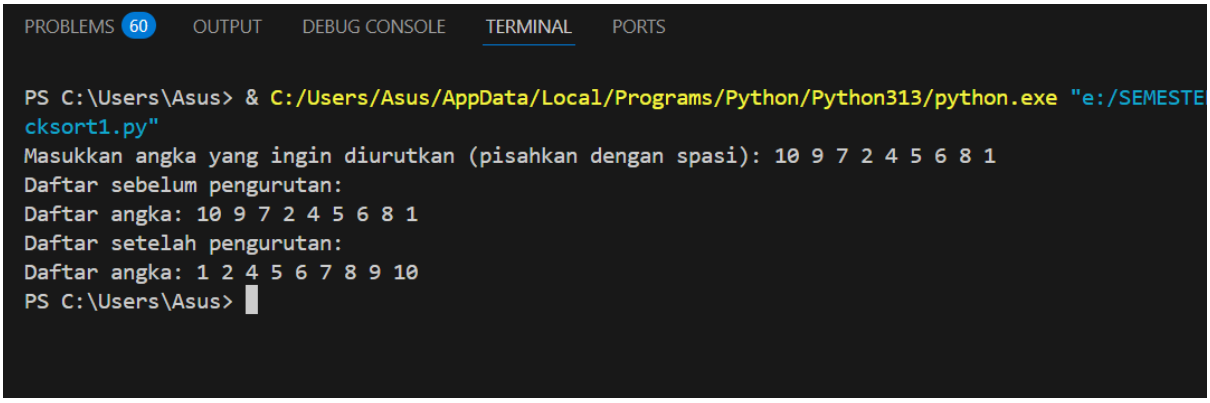
C. Sourcode QuickSort

```
def partisi(daftar, rendah, tinggi):

    """

    Fungsi untuk melakukan partisi pada daftar.

    Elemen yang lebih kecil dari pivot akan dipindahkan ke kiri,

    sedangkan yang lebih besar akan tetap di kanan.

    """
```

```
    pivot = daftar[tinggi]  # Memilih pivot

    i = rendah - 1  # Indeks untuk elemen yang lebih kecil



    for j in range(rendah, tinggi):

        if daftar[j] < pivot:

            i += 1

            daftar[i], daftar[j] = daftar[j], daftar[i]  # Tukar
elemen



    daftar[i + 1], daftar[tinggi] = daftar[tinggi], daftar[i + 1]  #
Tukar pivot ke posisi yang benar

    return i + 1



def pengurutan_cepat(daftar, rendah, tinggi):

    """

    Fungsi rekursif untuk mengurutkan daftar menggunakan algoritma
QuickSort.

    """

    if rendah < tinggi:

        pi = partisi(daftar, rendah, tinggi)  # Mendapatkan indeks
pivot setelah partisi

        pengurutan_cepat(daftar, rendah, pi - 1)  # Rekursi pada
bagian kiri

        pengurutan_cepat(daftar, pi + 1, tinggi)  # Rekursi pada
bagian kanan



def cetak_daftar(daftar):

    """

    Fungsi untuk mencetak daftar dalam format yang lebih mudah
dibaca.

    """
```

```
    print("Daftar angka:", " ".join(map(str, daftar)))



# Program utama

def main():

    """

    Fungsi utama untuk menerima input dari pengguna dan menjalankan
algoritma QuickSort.

    """

    try:

        daftar = list(map(int, input("Masukkan angka yang ingin
diurutkan (pisahkan dengan spasi): ").split()))

        if not daftar:

            raise ValueError("Daftar angka tidak boleh kosong.")



        print("Daftar sebelum pengurutan:")

        cetak_daftar(daftar)



        pengurutan_cepat(daftar, 0, len(daftar) - 1)



        print("Daftar setelah pengurutan:")

        cetak_daftar(daftar)

    except ValueError:

        print("Harap masukkan angka yang valid!")



if __name__ == "__main__":

    main()
```

Screnshot Output:



D. Analisis Kebutuhan Waktu

1. Analisis Berdasarkan Operasi/Intruksi yang Dieksekusi

   Operasi Yang dilakukan

   - Perbandingan: Setiap Elemen dalam sub array dibandingkan dengan elemen pivot
   - Pertukaran Elemen: Jika elemen lebih kecil dari pivot,elemen tersebut akan dipindahkan ke kiri.jika lebih besar, tetap di kanan
   - Pemanggil Rekursi: Setelah partisi, fungsi QuickSort dipanggil lagi untuk dua bagian (kiri dan kanan) dari array

2. Proses Partisi

   Proses ini memindahkan elemen yang kecil ke satu sisi pivot. Ini dilakukan dengan iterasi(perulangan) untuk setiap elemen dalam subarray

3. Analisis Berdasarkan Kasus

   - Best-case : O(n log n)

     Contoh: Jika kita memiliki array [3,1,4,2] dan memilih pivot sebagai 2, maka kita mendapatkan dua bagian: 1dan 3,4.

   - Wort-Case : O(n2)

     Contoh: Jika kita memiliki array [1,2,3,4,5] dan selalu memilij elemen pertama sebagai pivot, maka kita akan mendapatkan [] dan [2,3,4,5]. Ini menyebabkan banyak pemanggilan rekursi yang tidak efesien.

- Average-case : O(n log n)

  Contoh: Jika kita memiliki array [5,3,8,4,2] dan memilih pivot sebagai 4, kita mendapatkan dua bagian: [3,2] dan [5,8]. Ini adalah pembagian yang lebih seimbang.

E. Referensi

1. Levitin, Anany. *Introduction to the Design & Analysis of Algorithms* (3rd Edition)
2. Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. *Introduction to Algorithms* (3rd Edition)

F. Link Github

https://github.com/ketherinfr14/Quicksort