

# **QUIZ**

## **PERANCANGAN DAN ANALISIS ALGORITMA**

### **“ALGORITMA SPACE AND TIME TRADE OFFS DENGAN MENGUNAKAN METODE SORTING BY COUNTING”**

“Disusun sebagai tugas pada mata kuliah Perancangan dan Analisis Algoritma , dengan  
Dosen Randi Proska Sandra, M.Sc dan Widya Darwin S.Pd., M.Pd.T”



**Disusun Oleh :**

**PUTRI MAHARANI || 21346018**

**PROGRAM STUDI S1 TEKNIK INFORMATIKA  
JURUSAN TEKNIK ELEKTRONIKA  
FAKULTAS TEKNIK  
UNIVERSITAS NEGERI PADANG  
TAHUN 2023**

## **A. METODE ALGORITMA SORTING BY COUNTING DALAM STRATEGI ALGORITMA SPACE AND TIME TRADE OFFS(KOMPROMI RUANG DAN WAKTU)**

Algoritma space and time trade-offs (kompromi ruang dan waktu) mengacu pada cara-cara untuk memilih antara penggunaan lebih banyak sumber daya komputer (seperti ruang memori) atau lebih banyak waktu komputasi dalam implementasi algoritma.

Dalam beberapa kasus, algoritma yang membutuhkan sedikit waktu komputasi mungkin memerlukan penggunaan memori yang besar, dan sebaliknya, algoritma yang menggunakan sedikit memori mungkin memerlukan waktu komputasi yang lebih lama. Oleh karena itu, untuk memilih algoritma yang tepat untuk sebuah masalah, kita perlu mempertimbangkan faktor-faktor seperti kecepatan, memori yang tersedia, dan efisiensi.

Untuk mengurangi kompleksitas algoritma dalam hal waktu, beberapa teknik dapat digunakan seperti memperbaiki kompleksitas waktu, mengurangi jumlah operasi, dan menggunakan teknik heuristik. Sedangkan, untuk mengurangi kompleksitas algoritma dalam hal memori, teknik-teknik seperti kompresi data, penggunaan struktur data yang lebih efisien, atau memori virtual dapat digunakan.

Dalam praktiknya, memilih algoritma terbaik untuk sebuah masalah melibatkan evaluasi trade-off antara penggunaan waktu dan memori. Kita harus mempertimbangkan sumber daya yang tersedia, ukuran masalah, dan persyaratan performa untuk memilih algoritma yang paling efisien.

Metode algoritma sorting by counting adalah salah satu metode pengurutan data yang sederhana dan efektif. Metode ini bekerja dengan cara menghitung jumlah kemunculan setiap nilai dalam kumpulan data yang akan diurutkan. Kemudian, nilai-nilai tersebut diurutkan secara berurutan berdasarkan jumlah kemunculannya.

Proses sorting by counting dimulai dengan membuat sebuah array kosong dengan ukuran sama dengan jumlah nilai unik yang terdapat dalam kumpulan data. Selanjutnya, untuk setiap nilai dalam kumpulan data, frekuensi kemunculannya dihitung dan disimpan di dalam array pada indeks yang sesuai dengan nilai tersebut. Setelah semua nilai dihitung, array yang terisi kemudian diurutkan dengan cara memindahkan nilai dari array ke kumpulan data yang telah diurutkan sesuai dengan urutan nilai dalam array.

Metode sorting by counting dapat bekerja dengan cepat dan efisien jika nilai-nilai dalam kumpulan data memiliki rentang nilai yang kecil dan terdistribusi merata. Namun, jika nilai-nilai dalam kumpulan data memiliki rentang nilai yang besar dan tidak terdistribusi merata, metode ini dapat menghasilkan kinerja yang buruk karena memerlukan alokasi memori yang besar.

Meskipun demikian, sorting by counting tetap menjadi pilihan yang baik dalam beberapa kasus, seperti dalam pengurutan data yang berisi nilai-nilai yang terbatas, seperti huruf, karakter, atau bilangan bulat kecil.

## B. CONTOH PSEUDO CODE DARI ALGORITMA SORTING BY COUNTING

Pseudo code Mengurutkan sebuah array dengan algoritma Sorting by Counting :

### Deklarasi :

```
def pengurutan_jumlah_kemunculan(arr):  
    n = len(arr)  
    maksimal = max(arr)  
    frekuensi = [0] * (maksimal+1)  
    urutkan = [0] * n
```

### Algoritma :

```
# menghitung frekuensi kemunculan setiap nilai pada array  
for i in range(n):  
    frekuensi[arr[i]] += 1  
  
# menjumlahkan frekuensi nilai-nilai sebelumnya pada array  
frekuensi  
for i in range(1, maksimal+1):  
    frekuensi[i] += frekuensi[i-1]  
  
# memindahkan nilai dari arr ke urutkan secara terurut  
for i in range(n-1, -1, -1):  
    urutkan[frekuensi[arr[i]]-1] = arr[i]  
    frekuensi[arr[i]] -= 1  
  
return urutkan
```

### Output :

```
data = [4, 2, 1, 4, 6, 5, 2] urutan =  
pengurutan_jumlah_kemunculan(data) print(urutan) # Output: [1,  
2, 2, 4, 4, 5, 6]
```

### C. CONTOH PROGRAM TENTANG ALGORITMA SORTING BY COUNTING

```
def pengurutan_jumlah_kemunculan(arr):
    n = len(arr)
    maksimal = max(arr)
    frekuensi = [0] * (maksimal+1)
    urutan = [0] * n

    # menghitung frekuensi kemunculan setiap nilai pada array
    for i in range(n):
        frekuensi[arr[i]] += 1

    # menjumlahkan frekuensi nilai-nilai sebelumnya pada array frekuensi
    for i in range(1, maksimal+1):
        frekuensi[i] += frekuensi[i-1]

    # memindahkan nilai dari arr ke urutan secara terurut
    for i in range(n-1, -1, -1):
        urutan[frekuensi[arr[i]]-1] = arr[i]
        frekuensi[arr[i]] -= 1

    return urutan

# contoh penggunaan
data = [4, 2, 1, 4, 6, 5, 2]
urutan = pengurutan_jumlah_kemunculan(data)
print("Urutan dari data tersebut adalah",urutan) # Output: [1, 2, 2, 4, 4, 5, 6]
```

Program di atas adalah sebuah implementasi dari algoritma sorting by counting dengan menggunakan bahasa pemrograman Python. Algoritma sorting by counting adalah salah satu algoritma pengurutan data yang memiliki kompleksitas waktu  $O(n+k)$ , di mana  $n$  adalah jumlah data yang akan diurutkan, dan  $k$  adalah range nilai data.

Pada program di atas, terdapat sebuah fungsi bernama **`pengurutan_jumlah_kemunculan`** yang menerima sebuah array sebagai parameter, dan mengembalikan array yang telah diurutkan secara terurut berdasarkan frekuensi kemunculan setiap nilai pada array tersebut.

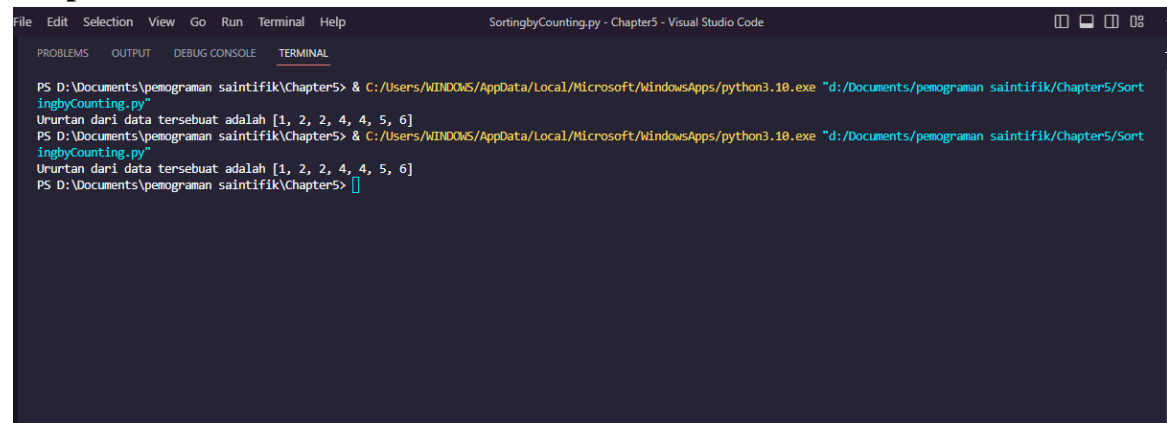
Langkah-langkah algoritma sorting by counting pada program di atas adalah sebagai berikut:

1. Mencari nilai maksimum pada array, untuk mengetahui range nilai yang akan digunakan untuk membuat array frekuensi.
2. Membuat array frekuensi dengan panjang sebanyak nilai maksimum pada array ditambah satu, dan menginisialisasi semua elemennya dengan nilai nol.
3. Menghitung frekuensi kemunculan setiap nilai pada array, dan menyimpan frekuensi tersebut pada indeks yang sesuai pada array frekuensi.

4. Menjumlahkan frekuensi nilai-nilai sebelumnya pada array frekuensi, sehingga setiap nilai pada array frekuensi menunjukkan jumlah nilai yang kurang dari atau sama dengan nilai tersebut.
5. Memindahkan nilai dari array ke array baru secara terurut berdasarkan frekuensi kemunculannya, dimulai dari akhir array, dan indeks dimulai dari nilai pada array frekuensi.
6. Mengembalikan array yang telah diurutkan.

Pada contoh penggunaan program di atas, terdapat array [4, 2, 1, 4, 6, 5, 2], kemudian diurutkan menggunakan fungsi **pengurutan\_jumlah\_kemunculan**, sehingga menghasilkan array [1, 2, 2, 4, 4, 5, 6].

### Output :



```
File Edit Selection View Go Run Terminal Help
SortingbyCounting.py - Chapter5 - Visual Studio Code

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

PS D:\Documents\pemograman saintifik\Chapter5> & C:/Users/WINDOWS/AppData/Local/Microsoft/WindowsApps/python3.10.exe "d:/Documents/pemograman saintifik/Chapter5/SortingbyCounting.py"
Urutan dari data tersebut adalah [1, 2, 2, 4, 4, 5, 6]
PS D:\Documents\pemograman saintifik\Chapter5> & C:/Users/WINDOWS/AppData/Local/Microsoft/WindowsApps/python3.10.exe "d:/Documents/pemograman saintifik/Chapter5/SortingbyCounting.py"
Urutan dari data tersebut adalah [1, 2, 2, 4, 4, 5, 6]
PS D:\Documents\pemograman saintifik\Chapter5> 
```

***LINK GITHUB :***

***[https://github.com/Putrimaharani9/Tugas\\_PerancanganAnalisisAlgoritma](https://github.com/Putrimaharani9/Tugas_PerancanganAnalisisAlgoritma)***