ALGORITMA DAN STRUKTUR DATA

INF1083

LAPORAN PRAKTIKUM 4 : Analisis Algoritma (1)

Oleh :

Akhmad Qasim

2211102441237

Teknik Informatika
Fakultas Sains & Teknologi
Universitas Muhammadiyah Kalimantan Timur

Samarinda, 2023

Laporan Praktikum 4:
Analisis Algoritma (1)

Pokok Bahasan:

- Komparasi Algoritma
- Notasi Big O
- Algoritma *Checking Off*
- Algoritma *Sort & Compare*
- Algoritma *Count & Compare*

Tujuan Pembelajaran:

- ✓ Memahami cara menganalisis algoritma dengan komparasi hasil program.
- ✓ Memahami notasi Big O.
- ✓ Mengatasi masalah menggunakan algoritma *Checking Off, Sort & Compare, dan Count & Compare*.

Percobaan & Latihan:

1. **Komparasi Algoritma**

a) Berikan tampilan output kedua fungsi tersebut dan analisa!

```
akhmadqasim
def foo(joko):  # Membuat fungsi foo dengan parameter joko
    agus = 0  # Membuat variabel agus dengan nilai 0
    for dini in range(1, joko + 1):  # Membuat perulangan dengan nilai awal 1 dan nilai akhir joko + 1
        ayu = dini  # Membuat variabel ayu dengan nilai dini
        agus = agus + ayu  # Menghitung nilai agus dengan menjumlahkan nilai agus dengan nilai ayu

    return agus  # Mengembalikan nilai agus


print(foo(10))  # Menampilkan hasil dari fungsi foo dengan parameter 10


akhmadqasim
def sumOfN(n):  # Membuat fungsi sumOfN dengan parameter n
    theSum = 0  # Membuat variabel theSum dengan nilai 0
    for i in range(1, n + 1):  # Membuat perulangan dengan nilai awal 1 dan nilai akhir n + 1
        theSum = theSum + i  # Menghitung nilai theSum dengan menjumlahkan nilai theSum dengan nilai i

    return theSum  # Mengembalikan nilai theSum


print(sumOfN(10))  # Menampilkan hasil dari fungsi sumOfN dengan parameter 10
```

b) Tentukan fungsi mana yang terbaik dan berikan alasannya!
Fungsi sumOfN() lebih baik dibandingkan fungsi foo() karena fungsi sumOfN() lebih

mudah dibaca dan lebih mudah dipahami.

2. Komparasi Algoritma

a) Jalankan perintah iterasi berikut dan analisa!

```
16 for i in range(5):  # Membuat perulangan dengan nilai awal 0 dan nilai akhir 5
17     # Menampilkan hasil dari fungsi sumOfN2 dengan parameter 1000
18     print("Hasil disoal 4.2a adalah %d membutuhkan waktu %10.7f detik" % sumOfN2(10000))
```

```
Hasil disoal 4.2a adalah 50005000 membutuhkan waktu  0.0010004 detik
Hasil disoal 4.2a adalah 50005000 membutuhkan waktu  0.0000000 detik
Hasil disoal 4.2a adalah 50005000 membutuhkan waktu  0.0000000 detik
Hasil disoal 4.2a adalah 50005000 membutuhkan waktu  0.0009990 detik
Hasil disoal 4.2a adalah 50005000 membutuhkan waktu  0.0000000 detik
```

b) Jalankan perintah iterasi berikut dan analisa!

```
20 for i in range(5):  # Membuat perulangan dengan nilai awal 0 dan nilai akhir 5
21     # Menampilkan hasil dari fungsi sumOfN2 dengan parameter 100000
22     print("Hasil disoal 4.2b adalah %d membutuhkan waktu %10.7f detik" % sumOfN2(100000))
```

```
Hasil disoal 4.2b adalah 5000050000 membutuhkan waktu  0.0039992 detik
Hasil disoal 4.2b adalah 5000050000 membutuhkan waktu  0.0040002 detik
Hasil disoal 4.2b adalah 5000050000 membutuhkan waktu  0.0040002 detik
Hasil disoal 4.2b adalah 5000050000 membutuhkan waktu  0.0040002 detik
Hasil disoal 4.2b adalah 5000050000 membutuhkan waktu  0.0029993 detik
```

c) Jalankan perintah iterasi berikut dan analisa!

```
24 for i in range(5):  # Membuat perulangan dengan nilai awal 0 dan nilai akhir 5
25     # Menampilkan hasil dari fungsi sumOfN2 dengan parameter 1000000
26     print("Hasil disoal 4.2c adalah %d membutuhkan waktu %10.7f detik" % sumOfN2(1000000))
```

```
Hasil disoal 4.2c adalah 500000500000 membutuhkan waktu  0.0410006 detik
Hasil disoal 4.2c adalah 500000500000 membutuhkan waktu  0.0399995 detik
Hasil disoal 4.2c adalah 500000500000 membutuhkan waktu  0.0399997 detik
Hasil disoal 4.2c adalah 500000500000 membutuhkan waktu  0.0400028 detik
Hasil disoal 4.2c adalah 500000500000 membutuhkan waktu  0.0390213 detik
```

d) Bandingkan hasil analisa ketiga iterasi tersebut dan berikan keterangan!
Terdapat perbedaan waktu yang sangat signifikan antara fungsi sumOfN2 dengan fungsi sumOfN. Fungsi sumOfN2 membutuhkan waktu yang lebih lama daripada fungsi sumOfN, hal ini dikarenakan jumlah perulangan yang lebih banyak pada fungsi sumOfN2.

3. Komparasi Algoritma

a) Analisa perbandingan hasil iterasi dari fungsi diatas dengan hasil iterasi yang terdapat di Percobaan & Latihan 4.2a,b,c!
4.2 dengan 4.3 memiliki perbedaan dalam cara menghitung hasil iterasi. 4.2 menggunakan perulangan for untuk menghitung jumlah iterasi sedangkan 4.3 menggunakan rumus matematis untuk menghitung jumlah iterasi. Karena 4.3 menggunakan rumus matematis, maka waktu yang dibutuhkan untuk menghitung jumlah iterasi jauh lebih singkat dibandingkan dengan waktu yang dibutuhkan oleh 4.2. Karena itu, hasil iterasi pada kedua file tersebut akan sama, namun waktu yang dibutuhkan untuk menghitung iterasi pada 4.3 akan lebih singkat dibandingkan dengan 4.2.

b) Buatlah perbandingan hasil waktu dari fungsi sumOfN3(1000000) dengan hasil waktu dari Percobaan & Latihan 4.2c, analisa dan tentukan fungsi mana yang terbaik berdasarkan waktu proses!
4.2c

```
Hasil disoal 4.2c adalah 500000500000 membutuhkan waktu  0.0410006 detik
Hasil disoal 4.2c adalah 500000500000 membutuhkan waktu  0.0399995 detik
Hasil disoal 4.2c adalah 500000500000 membutuhkan waktu  0.0399997 detik
Hasil disoal 4.2c adalah 500000500000 membutuhkan waktu  0.0400028 detik
Hasil disoal 4.2c adalah 500000500000 membutuhkan waktu  0.0390213 detik
```

sumOfN3(1000000)

```
Hasil 1000000 iterasi adalah 500000500000 memerlukan  0.0148745 detik
```

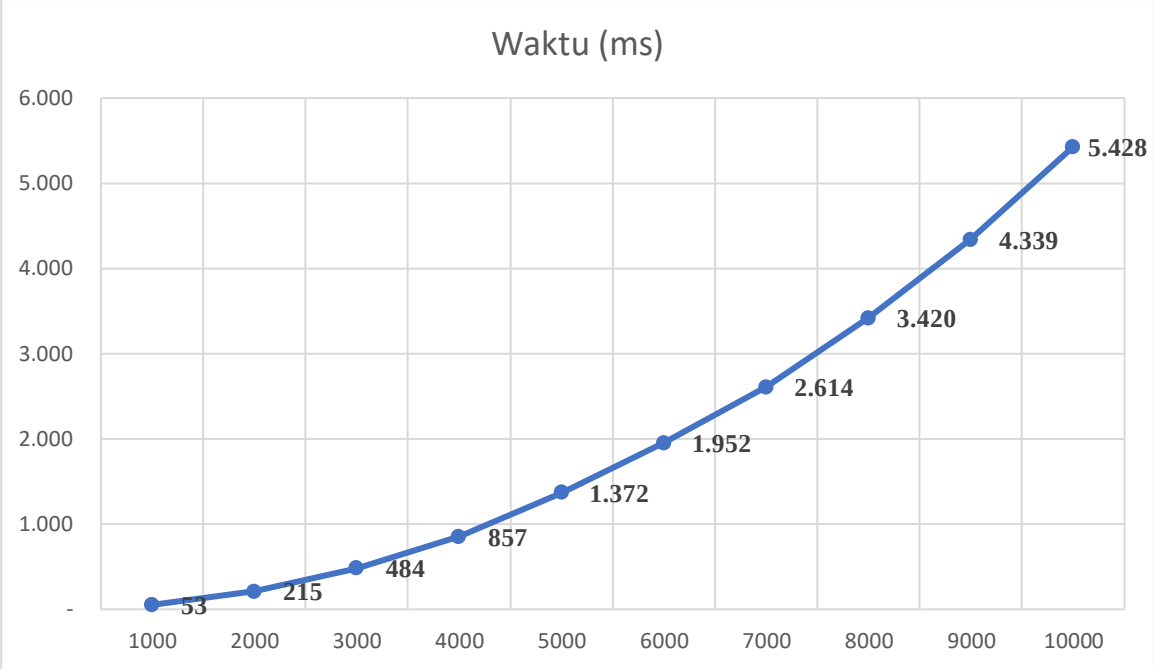
Jelas bahwa fungsi sumOfN3 lebih cepat daripada fungsi sumOfN2. Hal ini dikarenakan fungsi sumOfN3 hanya melakukan 1 perhitungan. Sedangkan fungsi sumOfN2 melakukan perhitungan sebanyak n + 1 menggunakan perulangan for.

4. Notasi Big O

a) Berikan tampilan output!

```
Hasill sumOfY(1000) adalah 249500250000 membutuhkan waktu      0.053 detik
Hasill sumOfY(2000) adalah 3996001000000 membutuhkan waktu     0.215 detik
Hasill sumOfY(3000) adalah 20236502250000 membutuhkan waktu    0.484 detik
Hasill sumOfY(4000) adalah 63968004000000 membutuhkan waktu    0.857 detik
Hasill sumOfY(5000) adalah 156187506250000 membutuhkan waktu   1.372 detik
Hasill sumOfY(6000) adalah 323892009000000 membutuhkan waktu   1.952 detik
Hasill sumOfY(7000) adalah 600078512250000 membutuhkan waktu   2.614 detik
Hasill sumOfY(8000) adalah 1023744016000000 membutuhkan waktu  3.420 detik
Hasill sumOfY(9000) adalah 1639885520250000 membutuhkan waktu  4.339 detik
Hasill sumOfY(10000) adalah 2499500025000000 membutuhkan waktu 5.428 detik
```

b) Buatlah grafik hasil fungsi diatas terhadap waktu (grafik silahkan menggunakan excel)!



c) Termasuk notasi big O apakah algoritma diatas?
Notasi O(n^2)

5. Algoritma Checking Off

a) Berikan tampilan output dan penjelasan tiap baris program!

```python
# Fungsi anagram merupakan fungsi untuk mengecek apakah dua string merupakan anagram atau tidak
# yang artinya kedua string tersebut memiliki karakter yang sama dan jumlahnya sama

akhmadqasim
def anagramSolution1(s1, s2):  # Fungsi untuk mengecek apakah dua string merupakan anagram
    alist = list(s2)  # Mengubah string kedalam list

    pos1 = 0  # Membuat variabel pos1 dengan nilai 0 untuk mengecek posisi string pertama
    stillOK = True  # Membuat variabel stillOK dengan nilai True untuk mengecek apakah masih sama

    # Membuat perulangan untuk mengecek apakah string pertama sama dengan string kedua
    while pos1 < len(s1) and stillOK:
        pos2 = 0  # Membuat variabel pos2 dengan nilai 0 untuk mengecek posisi string kedua
        found = False  # Membuat variabel found dengan nilai False untuk mengecek apakah ditemukan
        # Membuat perulangan untuk mengecek apakah string pertama sama dengan string kedua
        while pos2 < len(alist) and not found:
            if s1[pos1] == alist[pos2]:  # Mengecek apakah string pertama sama dengan string kedua
                found = True  # Jika sama maka found bernilai True
            else:  # Jika tidak sama
                pos2 = pos2 + 1  # pos2 ditambah 1

        if found:  # Jika found bernilai True
            alist[pos2] = None  # Membuat nilai list kedua menjadi None
        else:  # Jika found bernilai False
            stillOK = False  # Maka stillOK bernilai False

```

```
        pos1 = pos1 + 1  # pos1 ditambah 1

    return stillOK  # Mengembalikan nilai stillOK


print(anagramSolution1('abcd', 'dcba'))  # Menampilkan hasil dari fungsi anagramSolution1
print(anagramSolution1('abcd', 'abdc'))  # Menampilkan hasil dari fungsi anagramSolution1
print(anagramSolution1('abcd', 'defg'))  # Menampilkan hasil dari fungsi anagramSolution1
```

Output:

```
True
True
False
```

6. Algoritma Sort & Compare

a) Berikan tampilan output dan penjelasan tiap baris program!

```
akhmadqasim
def anagramSolution2(s1, s2):  # Fungsi untuk mengecek apakah dua string merupakan anagram
    alist1 = list(s1)  # Mengubah string kedalam list
    alist2 = list(s2)  # Mengubah string kedalam list
    alist1.sort()  # Mengurutkan list pertama
    alist2.sort()  # Mengurutkan list kedua
    pos = 0  # Membuat variabel pos dengan nilai 0 untuk mengecek posisi string
    matches = True  # Membuat variabel matches dengan nilai True untuk mengecek apakah sama
    while pos < len(s1) and matches:  # Membuat perulangan untuk mengecek apakah string pertama sama
        if alist1[pos] == alist2[pos]:  # Mengecek apakah string pertama sama dengan string kedua
            pos = pos + 1  # pos ditambah 1
        else:  # Jika tidak sama
            matches = False  # Maka matches bernilai False
    return matches  # Mengembalikan nilai matches


print(anagramSolution2('abcd', 'edcba'))  # Menampilkan hasil dari fungsi anagramSolution2
print(anagramSolution2('abcde', 'edcb'))  # Menampilkan hasil dari fungsi anagramSolution2
print(anagramSolution2('abcde', 'edfba'))  # Menampilkan hasil dari fungsi anagramSolution2
```

Output:

```
True
False
False
```

7. Algoritma Count & Compare

a) Berikan tampilan output dan penjelasan tiap baris program!

```
akhmadqasim
def anagramSolution2(s1, s2):  # Fungsi untuk mengecek apakah dua string merupakan anagram
    c1 = [0] * 26  # Membuat list c1 dengan panjang 26
    c2 = [0] * 26  # Membuat list c2 dengan panjang 26

    for i in range(len(s1)):  # Membuat perulangan untuk mengecek apakah string pertama sama dengan string kedua
        pos = ord(s1[i]) - ord('a')  # Mengubah string kedalam list
        c1[pos] = c1[pos] + 1  # Mengurutkan list pertama

    for i in range(len(s2)):  # Membuat perulangan untuk mengecek apakah string pertama sama dengan string kedua
        pos = ord(s2[i]) - ord('a')  # Mengubah string kedalam list
        c2[pos] = c2[pos] + 1  # Mengurutkan list kedua

    j = 0  # Membuat variabel j dengan nilai 0
    stillOK = True  # Membuat variabel stillOK dengan nilai True
    while j < 26 and stillOK:  # Membuat perulangan untuk mengecek apakah string pertama sama dengan string kedua
        if c1[j] == c2[j]:  # Mengecek apakah string pertama sama dengan string kedua
            j = j + 1  # j ditambah 1
        else:  # Jika tidak sama
            stillOK = False  # Maka stillOK bernilai False

    return stillOK  # Mengembalikan nilai stillOK


print(anagramSolution2('apple', 'pleap'))  # Menampilkan hasil dari fungsi anagramSolution2 yaiitu True
print(anagramSolution2('orange', 'ngerao'))  # Menampilkan hasil dari fungsi anagramSolution2 yaiitu True
print(anagramSolution2('apple', 'durian'))  # Menampilkan hasil dari fungsi anagramSolution2 yaiitu False
```

Output:

```
True
True
False
```

Tugas Mandiri:

O(n):

```
def pencarian_array(arr, x):
    for i in range(len(arr)):
        if arr[i] == x:
            return i
    return -1
```

O(n3):

```
def perkalian_matriks(A, B):
    m, n = len(A), len(B[0])
    C = [[0] * n for _ in range(m)]
    for i in range(m):
        for j in range(n):
            for k in range(len(B)):
                C[i][j] += A[i][k] * B[k][j]
    return C
```

Kesimpulan:

Analisis algoritma adalah proses mempelajari performa suatu algoritma dalam menyelesaikan sebuah masalah. Hal ini dilakukan dengan menganalisis waktu dan ruang yang dibutuhkan untuk menyelesaikan masalah dengan algoritma tersebut.

Notasi Big O adalah salah satu cara untuk mengukur kompleksitas waktu dari sebuah algoritma. Notasi Big O memberikan estimasi atas jumlah operasi yang diperlukan oleh sebuah algoritma dalam menyelesaikan sebuah masalah dalam rentang waktu tertentu. Dalam notasi Big O, waktu diukur sebagai fungsi dari ukuran input.

Anagram adalah sebuah istilah dalam bahasa Inggris yang merujuk pada sebuah kata atau frasa yang dibentuk dari huruf yang sama namun memiliki urutan yang berbeda. Contohnya adalah kata "debit card" dan "bad credit". Anagram dapat digunakan dalam pengkodean kata sandi atau dalam keamanan data, karena memiliki kemampuan untuk menyembunyikan arti dari sebuah pesan yang dikirim.

Kesimpulannya, analisis algoritma adalah proses untuk mempelajari performa suatu algoritma dalam menyelesaikan masalah, notasi Big O adalah cara untuk mengukur kompleksitas waktu dari sebuah algoritma, dan anagram adalah istilah yang merujuk pada kata atau frasa yang dibentuk dari huruf yang sama namun dengan urutan yang berbeda.