Latihan 1 : Desain Analisis & Algortima

Alfansyah Putra Raja Dinata

L0223002 – Sains Data B

1. **Minimum-Maximum Search**
    a. **Kode dengan algoritma Brute Force**

```
def f_min_max(arr):
    min_val = arr[0]
    max_val = arr[0]

    for num in arr[1:]:
        if num < min_val:
            min_val = num
        if num > max_val:
            max_val = num

    return min_val, max_val

array = [23, 15, 42, 4, 16, 8, 27]
min_val, max_val = f_min_max(array)
print(f"Nilai minimum: {min_val}, Nilai maksimum: {max_val}")
```

    b. **Pseudocode dan analisis algoritmanya**

```
Fungsi f_min_max(arr):
    min_val = arr[0]
    max_val = arr[0]

    Untuk setiap num dalam arr[1:]:
        Jika num < min_val:
            min_val = num
        Jika num > max_val:
            max_val = num

    Kembalikan min_val, max_val

array = [23, 15, 42, 4, 16, 8, 27]
min_val, max_val = f_min_max(array)
Cetak "Nilai minimum: ", min_val, " Nilai maksimum: ", max_val
```

Karena hanya melakukan looping untuk mencari elemen di array dari indeks 1 sampai ke n-1. Maka kompleksitas nya konstan di **O(n)**

2. **Sequential Search**
    a. **Sequential Code**

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

array = [12, 5, 7, 19, 23, 9, 14]
x = 19
index = seq_search(array, x)
print(f"Elemen {x} ditemukan pada indeks {index}" if index != -1 else
f"Elemen {x} tidak ditemukan dalam array")
```

    b. **Apakah hasilnya tetap efisien jika elemen yang dicari tidak ada dalam array?**

Hasil nya tetap akan efisien meskipun elemen tersebut tidak ada di dalam array karena skema algortima sequential search tetap melakukan *scanning* pada setiap elemen didalam array nya.

3. **Perkalian Matriks**
    a. **Brute Force Code**

```
def kali_matriks(A, B):
    n = len(A)
    m = len(A[0])
    p = len(B[0])

    C = [[0 for _ in range(p)] for _ in range(n)]

    for i in range(n):
        for j in range(p):
            for k in range(m):
                C[i][j] += A[i][k] * B[k][j]

    return C

A = [[1, 2], [3, 4]]
B = [[5, 6], [7, 8]]

C = kali_matriks(A, B)
print("Hasil perkalian matriks A dan B:")
for row in C:
    print(row)
```

b. Langkah-langkah

Langkah langkahnya seperti ini:

- Inisialisasi semua matriks;
- Iterasi setiap elemen yang ada dengan (C[i][j]);
- Untuk setiap elemen yang sudah diinput ke dalam (C[i][j]) lakukan perkalian elemen dari baris ke-i di Matriks A dengan elemen di baris j di Matriks B;
- Kembalikan nilai nya ke dalam Matriks C

4. **Tes Bilangan Prima**
   a. *Pseudocode* untuk Algotritma Brute Force:

```
def prima(n):
    if n <= 1:
        return False
    for i in range(2, n):
        if n % i == 0:
            return False
    return True

n = 29
if prima(n):
    print(f"{n} adalah bilangan prima")
else:
    print(f"{n} bukan bilangan prima")
```

b. Efisiensi

Di dalam algortima brute force kita melakukan pembagian sebanyak (n-2) kali (dari 2 hingga (n-1). Sehingga saat kita menaruh n di 29, dia akan melakukan pembagian di 29-2 = 27 kali.

Agar lebih efisien, kita bisa menambah fungsi untuk memeriksa akar kuadrat sehingga apabila setelah dibagi masih ada sisa, maka bisa disimpulkan bahwa angka tersebut merupakan bilangan prima. Untuk kode nya seperti ini:

```python
import math

def prima_eff(n):
    if n <= 1:
        return False
    for i in range(2, int(math.sqrt(n)) + 1):
        if n % i == 0:
            return False
    return True

n = 29
if prima_eff(n):
    print(f"{n} adalah bilangan prima")
else:
    print(f"{n} bukan bilangan prima")
```

Dengan menambah fungsi itu, yang awalnya harus melakukan pembagian sebanyak 27 kali, menjadi hanya 4 pembagian saja. Sehingga waktunya lebih cepat untuk pemorsesannya.

5. **Bubble Sort**
    a. **Algoritma Bubble Sort**

```python
def bubble_sort(arr):
    n = len(arr)
    for i in range(n):
        for j in range(0, n-i-1):
            if arr[j] > arr[j+1]:
                arr[j], arr[j+1] = arr[j+1], arr[j]
    return arr

array = [20, 5, 7, 12, 15, 3, 9]
sorted_array = bubble_sort(array)
print("Array yang diurutkan:", sorted_array)

output :
Array yang diurutkan: [3, 5, 7, 9, 12, 15, 20]
```

    b. **Pseudocode**

```
fungsi bubbleSort(arr):
    n = length(arr)
    for i = 0 to n-1:
        for j = 0 to n-i-2:
            if arr[j] > arr[j+1]:
                swap(arr[j], arr[j+1])
```

Analisis Kompleksitas Waktu:

- Kasus Terbaik (Best Case): O(n)
- Kasus Terburuk (Worst Case): O(n^2)

6. Selection Sort

a. Algoritma Selection Sort

```
def selection_sort(arr):
    n = len(arr)
    for i in range(n):
        min_idx = i
        for j in range(i+1, n):
            if arr[j] < arr[min_idx]:
                min_idx = j
        arr[i], arr[min_idx] = arr[min_idx], arr[i]
    return arr

array = [16, 2, 9, 14, 7, 3, 11]
sorted_array = selection_sort(array)
print("Array yang diurutkan:", sorted_array)

Output :
Array yang diurutkan: [2, 3, 7, 9, 11, 14, 16]
```

b. Pseudocode

```
function selectionSort(arr):
    n = length(arr)
    for i = 0 to n-1:
        min_idx = i
        for j = i+1 to n:
            if arr[j] < arr[min_idx]:
                min_idx = j
        swap(arr[i], arr[min_idx])
```

Algoritma Selection Sort Tidak Efisien untuk Dataset Besar, karena Selection Sort memiliki kompleksitas waktu O(n^2), yang berarti waktu eksekusi meningkat secara eksponensial dengan bertambahnya ukuran dataset. Ini membuatnya tidak efisien untuk dataset besar dibandingkan dengan algoritma lain seperti Quick Sort atau Merge Sort yang memiliki kompleksitas waktu O(n log n).

7. **Evaluasi Polinom**
    a. **Brute Force**

$$P(2) = 2(2)^3 + 3(2)^2 + 4(2) + 5$$

Hitung (2^3 = 8)

Kali hasilnya dengan 2: (2x8 = 16)
Hitung (2^2 = 4)
Kali hasilnya dengan 3: (3x4 = 12)
Kalikan 2 dengan 4: (4x 2 = 8)
Tambahkan semua hasil: (16 + 12 + 8 + 5 = 41)
Jadi, nilai (P(2) = 41).

    b. **Algoritma**

```python
def polynomial(x):
    result = 2 * x**3 + 3 * x**2 + 4 * x + 5
    return result

x = 2
print(f"P({x}) = { polynomial(x)}")
```

Kompleksitas Waktunya adalah O(1) dan Ruangnya juga O(1).

8. **Pencocokan String**
    a. **Brute Force Code**

```python
def brute_force_search(text, pattern):
    n = len(text)
    m = len(pattern)
    for i in range(n - m + 1):
        j = 0
        while j < m and text[i + j] == pattern[j]:
            j += 1
        if j == m:
            return i  # Mengembalikan indeks awal dari substring yang cocok
    return -1  # Mengembalikan -1 jika tidak ditemukan

text = "abcdabc"
pattern = "abc"
index = brute_force_search(text, pattern)
print(f"Pattern found at index {index}" if index != -1 else "Pattern not
found")
```

b. Penjelasan Algoritma dan Analisis Kompleksitas Waktu

Cara Kerja Algoritma

- Hitung panjang string text dan pattern.

- Lakukan iterasi dari indeks 0 hingga n - m (di mana n adalah panjang text dan m adalah panjang pattern).

- Untuk setiap posisi, bandingkan karakter pattern dengan substring text yang sesuai.

- Jika semua karakter cocok, kembalikan indeks awal dari substring yang cocok.

- Jika tidak ada kecocokan, kembalikan -1.

Kompleksitas Waktu

Worst Case: (O(n x m)), di mana n adalah panjang text dan m adalah panjang pattern1.

Best Case: (O(m)).

9. Mencari Pasangan Titik Terdekat

a. Brute Force

```
import math

points = [(1, 2), (4, 6), (5, 8), (7, 3)]

def bfdistance(point1, point2):
    return math.sqrt((point2[0] - point1[0])**2 + (point2[1] -
point1[1])**2)

def closest_pair_brute_force(points):
    min_distance = float('inf')
    closest_pair = None
    n = len(points)

    for i in range(n):
        for j in range(i + 1, n):
            distance = bfdistance(points[i], points[j])
            if distance < min_distance:
                min_distance = distance
                closest_pair = (points[i], points[j])

    return closest_pair, min_distance
```

```
closest_pair, min_distance = closest_pair_brute_force(points)
print(f"Pasangan titik terdekat: {closest_pair} dengan jarak
{min_distance}")
```

b. Euclidean

```
import math

def euclidean(point1, point2):
    return math.sqrt((point2[0] - point1[0])**2 + (point2[1] -
point1[1])**2)

point1 = (1, 2)
point2 = (4, 6)
distance = euclidean(point1, point2)
print(f"Jarak Euclidean antara {point1} dan {point2} adalah {distance}")
```

10. Masalah Kombinatronik

a. Algoritma

```
def generate_subsets(s):
    subsets = [[]]
    for element in s:
        new_subsets = [subset + [element] for subset in subsets]
        subsets.extend(new_subsets)
    return subsets

s = [1, 2, 3]
subsets = generate_subsets(s)
print("Semua subset dari himpunan {1, 2, 3}:")
for subset in subsets:
    print(subset)
```

b. Total Subset

Jumlah total subset yang dapat dibentuk dari himpunan berukuran (n) adalah (2^n). Ini karena setiap elemen dalam himpunan dapat ada atau tidak ada dalam subset, sehingga ada dua pilihan untuk setiap elemen.

11. Travelling Salesman Problem (TSP)

a. Algoritma

```
from itertools import permutations

cities = ['A', 'B', 'C', 'D']
distances = {
    ('A', 'B'): 10,
    ('A', 'C'): 15,
    ('A', 'D'): 20,
    ('B', 'C'): 35,
    ('B', 'D'): 25,
    ('C', 'D'): 30,
    ('B', 'A'): 10,
    ('C', 'A'): 15,
    ('D', 'A'): 20,
    ('C', 'B'): 35,
    ('D', 'B'): 25,
    ('D', 'C'): 30
}

def tsp(route):
    total_distance = 0
    for i in range(len(route) - 1):
        total_distance += distances[(route[i], route[i + 1])]
    total_distance += distances[(route[-1], route[0])]  # Kembali ke kota
asal
    return total_distance

def shortest(cities):
    shortest_route = None
    min_distance = float('inf')
    for route in permutations(cities):
        current_distance = tsp(route)
        if current_distance < min_distance:
            min_distance = current_distance
            shortest_route = route
    return shortest_route, min_distance

shortest_route, min_distance = shortest(cities)
print(f"Rute terpendek: {shortest_route} dengan jarak {min_distance}")
```

b. Banyak Rute

Untuk TSP dengan (n) kota, jumlah rute yang harus dicoba dengan pendekatan brute force adalah (n!) karena kita harus mempertimbangkan semua permutasi dari tiap kota.

Kompleksitas Waktu

Kompleksitas waktu dari algoritma brute force untuk TSP adalah (O(n!)), karena kita harus menghitung jarak untuk setiap permutasi dari tiap kota.

12. 1/0 Knapsack Problem

a. Algoritma

```
items = [(3, 25), (4, 40), (5, 50)]
max_capacity = 10

def knapsack(items, max_capacity):
    n = len(items)
    max_value = 0
    best_combination = []

    for i in range(1 << n):
        total_weight = 0
        total_value = 0
        combination = []
        for j in range(n):
            if i & (1 << j):
                total_weight += items[j][0]
                total_value += items[j][1]
                combination.append(items[j])
        if total_weight <= max_capacity and total_value > max_value:
            max_value = total_value
            best_combination = combination

    return best_combination, max_value

best_combination, max_value = knapsack(items, max_capacity)
print(f"Kombinasi terbaik: {best_combination} dengan nilai total
{max_value}")
```

b. Pseudocode

```
function knapsack(items, max_capacity):
    n = length(items)
    max_value = 0
    best_combination = []

    for i from 0 to (2^n - 1):
        total_weight = 0
        total_value = 0
        combination = []
        for j from 0 to (n - 1):
            if i & (1 << j):
                total_weight += items[j].weight
                total_value += items[j].value
                combination.append(items[j])
        if total_weight <= max_capacity and total_value > max_value:
            max_value = total_value
            best_combination = combination

    return best_combination, max_value
```

Pendekatan exhaustive search tidak efisien untuk jumlah barang yang lebih besar karena kompleksitas waktunya adalah (O(2^n)), di mana (n) adalah jumlah barang.

13. Permutasi dan Kombinasi

a. Algoritma

```
from itertools import permutations

array = [1, 2, 3]

permutasi = list(permutations(array))

print("Semua permutasi dari array [1, 2, 3]:")
for perm in permutasi:
    print(perm)
```

b. Kompleksitas

Kompleksitas waktu dari pencarian permutasi dengan pendekatan exhaustive search adalah (O(n!)), di mana (n) adalah panjang array. Ini karena kita harus menghasilkan semua permutasi yang mungkin dari array, dan jumlah permutasi dari array berukuran (n) adalah (n!) (n faktorial).

14. Partition Problem

a. Algoritma

```
def partisi(nums):
    total_sum = sum(nums)
    if total_sum % 2 != 0:
        return False

    target = total_sum // 2
    n = len(nums)

    def dfs(index, current_sum):
        if current_sum == target:
            return True
        if current_sum > target or index >= n:
            return False
        return dfs(index + 1, current_sum + nums[index]) or dfs(index + 1,
current_sum)

    return dfs(0, 0)

nums = [1, 5, 11, 5]
result = partisi(nums)
print(f"Dapat dipartisi: {result}")
```

b. Efisiensi

Algoritma ini menggunakan tahapan seperti berikut:

- Hitung jumlah total elemen dalam array. Jika jumlah total ganjil, maka tidak mungkin untuk membagi array menjadi dua subset dengan jumlah yang sama.
- Tentukan target yang merupakan setengah dari jumlah total.
- DFS (Depth-First Search): Gunakan DFS untuk mencoba semua kombinasi subset. Jika subset dengan jumlah yang sama dengan target ditemukan, kembalikan *True*.

Dengan kompleksitas waktu sebesar O(2^n) dimana n adalah jumlah elemen dalam array. Maka Pendekatan ini tidak efisien untuk jumlah elemen yang besar karena jumlah kombinasi yang harus diperiksa tumbuh secara eksponensial dengan bertambahnya jumlah elemen. Untuk array berukuran besar, waktu komputasi menjadi sangat besar dan tidak praktis untuk digunakan.