LAPORAN PRAKTIKUM

STRUKTUR DATA

MODUL KE-6

SORTING PADA PYTHON

Disusun Oleh:

Nama	**:** Oktario Mufti Yudha
NPM	**:** 2320506044
Kelas	**:** 04 (Empat)

Program Studi S1 Teknologi Informasi

Fakultas Teknik, Universitas Tidar

Genap 2023/2024

I. Tujuan Praktikum

1. Mahasiswa mampu memahami sorting pada python
2. Mahasiswa mampu menerapkan sorting pada program python

II. Dasar Teori

Mengurutkan data merupakan fondasi utama yang menjadi dasar bagi banyak algoritma lainnya. Hal ini terkait dengan beberapa konsep menarik yang akan Anda temui sepanjang perjalanan karier pemrograman Anda. Memahami bagaimana algoritma pengurutan bekerja di dalam Python adalah langkah awal yang penting untuk menerapkan algoritma dengan tepat dan efisien dalam menyelesaikan masalah dunia nyata. Pengurutan merupakan salah satu algoritma yang sering dipelajari dalam ilmu komputer. Terdapat banyak implementasi dan aplikasi berbeda dari pengurutan yang dapat meningkatkan efisiensi dan efektivitas kode Anda.

Ada berbagai masalah yang dapat diselesaikan dengan pengurutan:

- Pencarian: Pencarian item dalam daftar menjadi lebih cepat jika daftar tersebut telah diurutkan.

- Seleksi: Memilih item dari daftar berdasarkan hubungannya dengan item lainnya menjadi lebih mudah dengan data yang diurutkan. Contohnya, mencari nilai k-th terbesar atau terkecil, atau mencari nilai median dari daftar, akan lebih mudah jika nilainilai tersebut sudah dalam urutan menaik atau menurun.

- Penghapusan Duplikat: Menemukan nilai duplikat pada daftar menjadi lebih cepat ketika daftar sudah diurutkan.

Tanda Tangan

III. Hasil dan Pembahasan

a. Penyortiran Bawaan Python

```
# Algoritma penyortiran bawaan python
arr = [5, 2, 9, 1, 6]

sorted_arr = sorted(arr)

print('Array yang telah diurutkan: ', sorted_arr)
```

✓ 0.0s

```
Array yang telah diurutkan:  [1, 2, 5, 6, 9]
```

Gambar 3.1: Penyortiran Bawaam Python

1. `Algoritma penyortiran bawaan python`: Ini adalah komentar yang memberikan penjelasan bahwa kode berikutnya akan menggunakan algoritma penyortiran bawaan Python.
2. `arr = [5, 2, 9, 1, 6]`: Baris ini mendefinisikan sebuah list dengan nama `arr` yang berisi beberapa bilangan.
3. `sorted_arr = sorted(arr)`: Pada baris ini, kita menggunakan fungsi `sorted()` untuk mengurutkan list `arr` yang telah dibuat sebelumnya. Hasil pengurutan disimpan dalam variabel `sorted_arr`.
4. `print('Array yang telah diurutkan: ', sorted_arr)`: Baris terakhir ini mencetak hasil pengurutan array `sorted_arr` ke dalam output, bersama dengan sebuah pesan yang memberikan informasi tentang array yang telah diurutkan.
5. kode tersebut menghasilkan output berupa array yang telah diurutkan dari terkecil hingga terbesar, yaitu [1, 2, 5, 6, 9]

Tanda Tangan

b. Bubble Sort

```python
# Bubble sort
def bubble_sort(arr):
    n = len(arr)
    for i in range(n):
        already_sorted = True
        for j in range(n - i - 1):
            if arr[j] > arr[j + 1]:
                arr[j], arr[j + 1] = arr[j + 1], arr[j]
                already_sorted = False
        if already_sorted:
            break
    return arr

arr = [64, 34, 25, 12, 22, 11, 90]

print('Array sebelum diurutkan: ', arr)
bubble_sort(arr)
print('Array setelah diurutkan: ', arr)
```

✓ 0.0s

```
Array sebelum diurutkan:  [64, 34, 25, 12, 22, 11, 90]
Array setelah diurutkan:  [11, 12, 22, 25, 34, 64, 90]
```

Gambar 3.2: Bubble Sort

1. `def bubble_sort(arr):` : Mendefinisikan sebuah fungsi bernama `bubble_sort` yang menerima sebuah argumen `arr`, yang akan diurutkan menggunakan algoritma bubble sort.
2. `n = len(arr)` : Menghitung panjang (jumlah elemen) dari array `arr` dan menyimpannya dalam variabel `n`.
3. `for i in range(n):` : Memulai iterasi untuk setiap elemen dalam array `arr` menggunakan loop `for`, dimulai dari indeks 0 hingga `n-1`.
4. `already_sorted = True` : Menginisialisasi variabel `already_sorted` sebagai True, yang akan digunakan untuk menandai apakah array sudah terurut atau belum pada setiap iterasi.
5. `for j in range(n - i - 1):` : Memulai iterasi kedua untuk membandingkan elemen-elemen berdekatan dalam array.
6. `if arr[j] > arr[j + 1]:` : Memeriksa apakah elemen saat ini lebih besar dari elemen berikutnya.
7. `arr[j], arr[j + 1] = arr[j + 1], arr[j]` : Melakukan pertukaran posisi antara elemen `arr[j]` dan `arr[j + 1]` jika perlu.

Tanda Tangan

8. `already_sorted = False`: Mengubah nilai `already_sorted` menjadi False jika terjadi pertukaran elemen, menandakan bahwa array belum terurut.
9. `if already_sorted: break`: Memeriksa apakah array sudah terurut. Jika sudah, maka keluar dari loop menggunakan pernyataan `break`.
10. `return arr`: Mengembalikan array yang telah diurutkan setelah selesai melakukan iterasi.
11. `arr = [64, 34, 25, 12, 22, 11, 90]`: Mendefinisikan sebuah array dengan nama `arr` yang akan diurutkan menggunakan fungsi `bubble_sort`.
12. `print('Array sebelum diurutkan: ', arr)`: Mencetak array sebelum diurutkan.
13. `bubble_sort(arr)`: Memanggil fungsi `bubble_sort` untuk mengurutkan array `arr` yang telah didefinisikan sebelumnya.
14. `print('Array setelah diurutkan: ', arr)`: Mencetak array setelah proses pengurutan selesai.
15. Output pertama mencetak array sebelum diurutkan, sedangkan output kedua mencetak array setelah diurutkan. Jadi, output pertama menunjukkan nilai awal array, sedangkan output kedua menunjukkan nilai array setelah proses pengurutan.

Tanda Tangan

c. Insortion Sort

```python
# Insortion sort
def insertion_sort(arr):
    for i in range(1, len(arr)):
        key_item = arr[i]
        j = i - 1
        while j >= 0 and arr[j] > key_item:
            arr[j + 1] = arr[j]
            j -= 1
        arr[j + 1] = key_item
    return arr

arr = [64, 34, 25, 12, 22, 11, 90]
print('Array sebelum diurutkan: ', arr)
insertion_sort(arr)
print('Array setelah diurutkan: ', arr)
```

✓ 0.0s

```
Array sebelum diurutkan:  [64, 34, 25, 12, 22, 11, 90]
Array setelah diurutkan:  [11, 12, 22, 25, 34, 64, 90]
```

Gambar 3.3: Insortion Sort

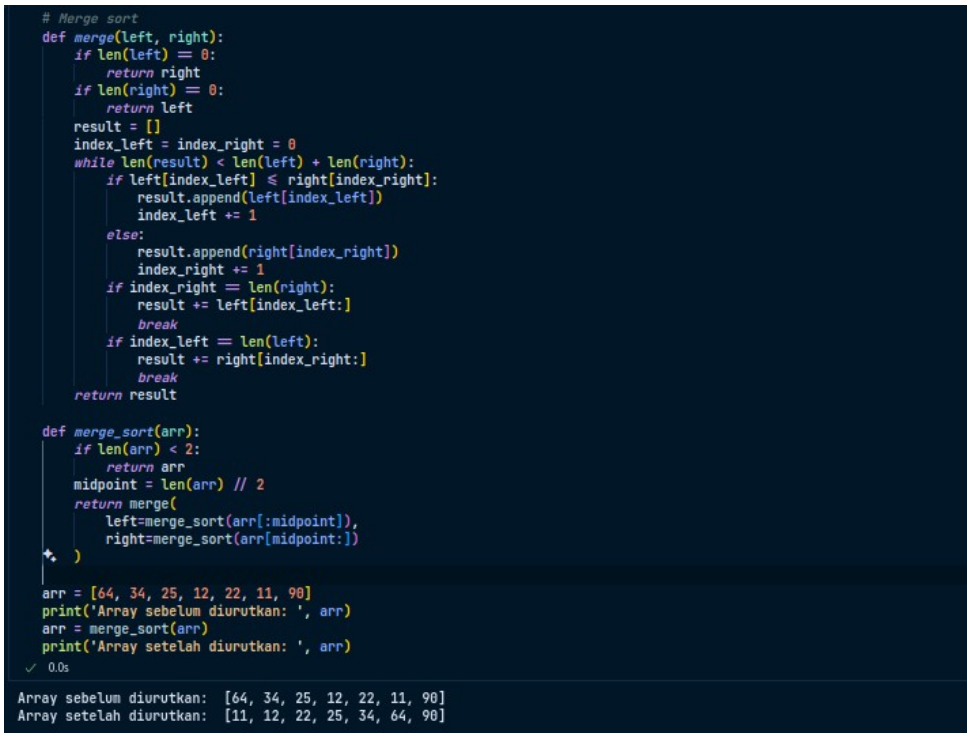
1. `def insertion_sort(arr):` : Mendefinisikan sebuah fungsi bernama `insertion_sort` yang menerima sebuah argumen `arr`, yang akan diurutkan menggunakan algoritma insertion sort.
2. `for i in range(1, len(arr)):` : Memulai iterasi dari indeks 1 hingga panjang array `arr`.
3. `key_item = arr[i]` : Menyimpan nilai dari elemen saat ini (yang sedang dipertimbangkan) dalam variabel `key_item`.
4. `j = i - 1` : Menginisialisasi variabel `j` dengan indeks sebelumnya untuk membandingkan elemen saat ini dengan elemen-elemen sebelumnya.
5. `while j >= 0 and arr[j] > key_item:` : Memulai loop while untuk memindahkan elemen-elemen yang lebih besar dari `key_item` ke kanan.
6. `arr[j + 1] = arr[j]` : Memindahkan elemen yang lebih besar dari `key_item` ke posisi kanan.
7. `j -= 1` : Mengurangi nilai `j` untuk memeriksa elemen sebelumnya dalam array.

Tanda Tangan

8. `arr[j + 1] = key_item`: Memasukkan `key_item` ke posisi yang benar setelah semua elemen yang lebih besar telah dipindahkan.
9. `return arr`: Mengembalikan array yang telah diurutkan setelah selesai melakukan iterasi.
10. `arr = [64, 34, 25, 12, 22, 11, 90]`: Mendefinisikan sebuah array dengan nama `arr` yang akan diurutkan menggunakan fungsi `insertion_sort`.
11. `print('Array sebelum diurutkan: ', arr)`: Mencetak array sebelum diurutkan.
12. `insertion_sort(arr)`: Memanggil fungsi `insertion_sort` untuk mengurutkan array `arr` yang telah didefinisikan sebelumnya.
13. `print('Array setelah diurutkan: ', arr)`: Mencetak array setelah proses pengurutan selesai.
14. Output pertama mencetak array sebelum diurutkan, sedangkan output kedua mencetak array setelah diurutkan. Jadi, output pertama menunjukkan nilai awal array, sedangkan output kedua menunjukkan nilai array setelah proses pengurutan.

Tanda Tangan

d. Merge Sort



```
# Merge sort
def merge(left, right):
    if len(left) == 0:
        return right
    if len(right) == 0:
        return left
    result = []
    index_left = index_right = 0
    while len(result) < len(left) + len(right):
        if left[index_left] <= right[index_right]:
            result.append(left[index_left])
            index_left += 1
        else:
            result.append(right[index_right])
            index_right += 1
        if index_right == len(right):
            result += left[index_left:]
            break
        if index_left == len(left):
            result += right[index_right:]
            break
    return result

def merge_sort(arr):
    if len(arr) < 2:
        return arr
    midpoint = len(arr) // 2
    return merge(
        left=merge_sort(arr[:midpoint]),
        right=merge_sort(arr[midpoint:])
    )

arr = [64, 34, 25, 12, 22, 11, 90]
print('Array sebelum diurutkan: ', arr)
arr = merge_sort(arr)
print('Array setelah diurutkan: ', arr)
```

✓ 0.0s

```
Array sebelum diurutkan:  [64, 34, 25, 12, 22, 11, 90]
Array setelah diurutkan:  [11, 12, 22, 25, 34, 64, 90]
```

Gambr 3.4: Merge Sort

1. `def merge(left, right):` : Mendefinisikan fungsi `merge` yang digunakan untuk menggabungkan dua array yang telah diurutkan.
2. `if len(left) == 0:` dan `if len(right) == 0:` : Memeriksa apakah salah satu dari array yang akan digabungkan kosong.
3. `result = []` : Membuat list kosong yang akan digunakan untuk menyimpan hasil penggabungan.
4. `index_left = index_right = 0` : Menginisialisasi variabel `index_left` dan `index_right` sebagai indeks awal untuk array kiri dan kanan.
5. `while len(result) < len(left) + len(right):` : Memulai loop while untuk menggabungkan elemen-elemen dari kedua array hingga semua elemen telah digabungkan.
6. Di dalam loop while, dilakukan pengecekan untuk menentukan elemen mana yang akan dimasukkan ke dalam hasil penggabungan. Jika nilai pada `left[index_left]` lebih kecil dari atau sama dengan

Tanda Tangan

`right[index_right]`, maka elemen dari `left` dimasukkan ke dalam `result`, dan sebaliknya.

7. Setelah selesai memasukkan elemen, dilakukan pengecekan apakah sudah mencapai akhir dari salah satu array. Jika sudah, maka sisa elemen dari array yang masih memiliki elemen akan langsung dimasukkan ke dalam `result`.
8. `return result`: Mengembalikan array yang telah digabungkan dan diurutkan.
9. `def merge_sort(arr):`: Mendefinisikan fungsi `merge_sort` yang digunakan untuk melakukan sorting menggunakan algoritma merge sort.
10. `if len(arr) < 2:`: Memeriksa apakah panjang array `arr` kurang dari 2. Jika ya, maka array tersebut dianggap sudah terurut (karena satu elemen atau tidak ada elemen) dan langsung dikembalikan.
11. `midpoint = len(arr) // 2`: Menghitung titik tengah array `arr` untuk membagi array menjadi dua bagian.
12. `return merge(left=merge_sort(arr[:midpoint]), right=merge_sort(arr[midpoint:]))`: Mengembalikan hasil penggabungan dua array yang telah diurutkan menggunakan fungsi `merge`.
13. `arr = [64, 34, 25, 12, 22, 11, 90]`: Mendefinisikan sebuah array dengan nama `arr` yang akan diurutkan menggunakan fungsi `merge_sort`.
14. `print('Array sebelum diurutkan: ', arr)`: Mencetak array sebelum diurutkan.
15. `arr = merge_sort(arr)`: Memanggil fungsi `merge_sort` untuk mengurutkan array `arr` yang telah didefinisikan sebelumnya dan menyimpan hasilnya kembali ke dalam variabel `arr`.
16. `print('Array setelah diurutkan: ', arr)`: Mencetak array setelah proses pengurutan selesai.
17. Output pertama mencetak array sebelum diurutkan, sedangkan output kedua mencetak array setelah diurutkan. Jadi, output pertama menunjukkan nilai awal array, sedangkan output kedua menunjukkan nilai array setelah proses pengurutan.

Tanda Tangan

IV. Tugas

a. Latihan

```python
# Latihan
def partisi(arr, low, high):
    i = (low-1)
    pivot = arr[high]

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

    arr[i+1], arr[high] = arr[high], arr[i+1]
    return (i+1)

def quickSort(arr, low, high):
    if len(arr) == 1:
        return arr
    if low < high:
        pi = partisi(arr, low, high)

        quickSort(arr, low, pi-1)
        quickSort(arr, pi+1, high)

arr = [64, 34, 25, 12, 22, 11, 90]
n = len(arr)
quickSort(arr, 0, n-1)
print("Array yang telah diurutkan adalah:", arr)
```

✓ 0.0s

```
Array yang telah diurutkan adalah: [11, 12, 22, 25, 34, 64, 90]
```

Gambr 4.1: Latihan

1. `def partisi(arr, low, high):` : Mendefinisikan fungsi `partisi` untuk membagi array menjadi dua bagian (partisi) berdasarkan nilai pivot.
2. `i = (low-1)` : Menginisialisasi variabel `i` sebagai indeks untuk elemen terakhir dari partisi yang lebih kecil dari pivot.
3. `pivot = arr[high]` : Memilih pivot dari array, yang dalam kasus ini diambil sebagai elemen terakhir (`arr[high]`).
4. `for j in range(low, high):` : Memulai iterasi untuk setiap elemen dari `low` hingga `high`.
5. `if arr[j] <= pivot:` : Memeriksa apakah nilai elemen saat ini kurang dari atau sama dengan pivot.
6. `i = i+1` dan `arr[i], arr[j] = arr[j], arr[i]` : Jika kondisi terpenuhi, maka nilai `i` ditingkatkan dan dilakukan pertukaran nilai antara elemen ke-i dan elemen saat ini (`arr[j]`).
7. Setelah selesai iterasi, dilakukan pertukaran pivot ke posisi yang benar dalam array.

Tanda Tangan

8. `def quickSort(arr, low, high):` : Mendefinisikan fungsi `quickSort` untuk melakukan sorting menggunakan algoritma Quick Sort.
9. `if len(arr) == 1:` : Memeriksa jika panjang array `arr` hanya 1, maka array tersebut dianggap sudah terurut dan langsung dikembalikan.
10. `pi = partisi(arr, low, high)` : Memanggil fungsi `partisi` untuk membagi array menjadi dua bagian dan mendapatkan indeks pivot (`pi`).
11. `quickSort(arr, low, pi-1)` dan `quickSort(arr, pi+1, high)` : Memanggil rekursif fungsi `quickSort` untuk melakukan sorting pada dua bagian array yang lebih kecil dari pivot dan dua bagian array yang lebih besar dari pivot.
12. `arr = [64, 34, 25, 12, 22, 11, 90]` : Mendefinisikan sebuah array dengan nama `arr` yang akan diurutkan menggunakan fungsi `quickSort`.
13. `n = len(arr)` : Menghitung panjang array `arr`.
14. `quickSort(arr, 0, n-1)` : Memanggil fungsi `quickSort` untuk mengurutkan array `arr` mulai dari indeks 0 hingga indeks terakhir (`n-1`).
15. `print("Array yang telah diurutkan adalah:", arr)` : Mencetak array setelah proses pengurutan selesai.
16. Outputnya adalah `11` `12` `22` `25` `34` `64` `90` Ini adalah array setelah diurutkan menggunakan algoritma Quick Sort.

Tanda Tangan

V. Kesimpulan

Dari praktikum ini, dapat disimpulkan bahwa kita telah mempelajari berbagai cara sorting dalam pemrograman, seperti bubble sort, insortion sort, dan merge sort. Setiap teknik memiliki keunggulan dan kelemahan masing-masing. Contoh kode yang telah dijelaskan memberikan pemahaman tentang strategi sorting yang digunakan, penerapan algoritma, dan cara mengintegrasikan fungsi-fungsi sorting dalam kode yang lebih besar. Dengan memahami dan menguasai teknik-teknik sorting tersebut, kita dapat mengembangkan pemahaman yang kuat dalam pemrograman dan mampu menyelesaikan berbagai masalah secara efisien.

Tanda Tangan