



## MODUL PERKULIAHAN

# TFC251 – Praktikum Struktur Data

## Sorting Dalam Python

<b>Penyusun Modul</b>	: Suamanda Ika Novichasari, M.Kom
<b>Minggu/Pertemuan</b>	: 6
<b>Sub-CPMK/Tujuan Pembelajaran</b>	: <ol style="list-style-type: none"><li>1. Mahasiswa mampu menerapkan algoritma Sorting pada bahasa pemrograman python</li></ol>
<b>Pokok Bahasan</b>	: <ol style="list-style-type: none"><li>1. Bubble Sort</li><li>2. Insertion Sort</li><li>3. Merge Sort</li></ol>

**Program Studi Teknologi Informasi (S1)**  
**Fakultas Teknik**  
**Universitas Tidar**  
**Tahun 2024**



# Materi 6

## SEARCHING DALAM PYTHON

---

### Petunjuk Praktikum :

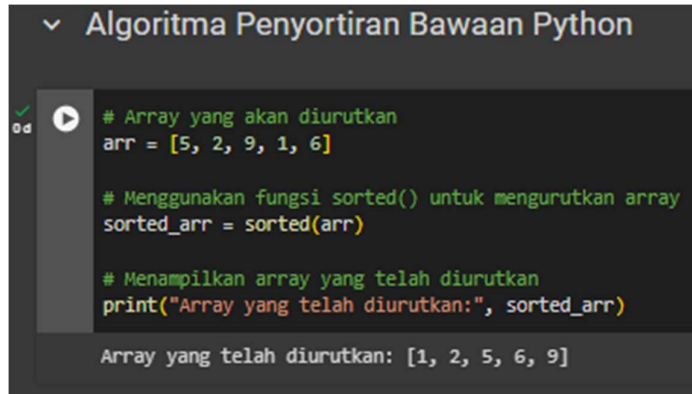
- Cobalah semua contoh penyelesaian kasus dengan kode program yang terdapat pada semua sub bab dalam modul ini menggunakan google colab.
- Dokumentasikan kegiatan dalam bentuk laporan studi kasus sesuai template laporan praktikum yang telah ditentukan.

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 nilai-nilai tersebut sudah dalam urutan menaik atau menurun.
- Penghapusan Duplikat: Menemukan nilai duplikat pada daftar menjadi lebih cepat ketika daftar sudah diurutkan.
- Analisis Distribusi: Menganalisis distribusi frekuensi item dalam suatu daftar akan lebih cepat jika daftar tersebut telah diurutkan. Contohnya, menemukan elemen yang paling sering atau paling jarang muncul menjadi lebih mudah dengan daftar yang telah diurutkan..

Dalam bahasa Python, seperti banyak bahasa pemrograman tingkat tinggi lainnya, Anda dapat mengurutkan data secara langsung menggunakan fungsi `sorted()`. Berikut adalah contoh pengurutan array integer :

A screenshot of a code editor window titled "Algoritma Penyortiran Bawaan Python". The code defines an array `arr = [5, 2, 9, 1, 6]`, uses `sorted(arr)` to sort it, and prints the result. The output shown at the bottom is "Array yang telah diurutkan: [1, 2, 5, 6, 9]".

```
✓ 0d ▶ # Array yang akan diurutkan
arr = [5, 2, 9, 1, 6]

# Menggunakan fungsi sorted() untuk mengurutkan array
sorted_arr = sorted(arr)

# Menampilkan array yang telah diurutkan
print("Array yang telah diurutkan:", sorted_arr)

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

## 6.1 Bubble Sort

Bubble Sort merupakan salah satu teknik pengurutan yang paling sederhana. Nama algoritma ini didasarkan pada prinsip kerjanya: dengan setiap iterasi, elemen terbesar dalam daftar "menggelembung" ke posisi yang seharusnya.

Bubble sort melibatkan pengulangan beberapa kali melalui daftar, membandingkan elemen satu per satu, dan menukar elemen yang tidak berdekatan yang berada pada urutan yang salah.

Kelebihan utama dari algoritma bubble sort adalah kemudahannya. Algoritma ini sangat sederhana untuk diimplementasikan dan dipahami. Mungkin itulah alasan utama mengapa banyak mata kuliah ilmu komputer memilih bubble sort sebagai topik pengantar dalam mempelajari pengurutan.

Namun, seperti yang telah Anda lihat sebelumnya, kelemahan bubble sort adalah kecepatannya yang lambat, dengan kompleksitas runtime  $O(n^2)$ . Hal ini membuatnya kurang praktis untuk mengurutkan array yang besar.

```
▼ Bubble Sort

def bubble_sort(array):
    n = len(array)

    for i in range(n):
        # Buat tanda yang memungkinkan fungsi dihentikan lebih awal jika tidak ada lagi yang perlu disortir
        already_sorted = True

        # mulailah melihat setiap item dalam daftar satu per satu, bandingkan dengan nilai yang berdekatan.
        # Dengan setiap iterasi, porsi array yang Anda lihat menyusut karena item yang tersisa telah diurutkan.
        for j in range(n - i - 1):
            if array[j] > array[j + 1]:
                # Jika item yang Anda lihat lebih besar dari nilai di dekatnya, tukar item tersebut
                array[j], array[j + 1] = array[j + 1], array[j]
                # Karena Anda harus menukar dua elemen, setel tanda 'already_sorted' ke 'False',
                # agar algoritme tidak selesai sebelum waktunya
                already_sorted = False

        # Jika tidak ada swap selama iterasi terakhir, array sudah diurutkan, dan Anda dapat mengakhirinya
        if already_sorted:
            break

    return array

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

print("Array sebelum pengurutan:", arr)
bubble_sort(arr)
print("Array setelah pengurutan:", arr)

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

Mari kita ilustrasikan bagaimana algoritma Bubble Sort bekerja pada array dengan nilai [64, 34, 25, 12, 22, 11, 90]. Saya akan memberikan langkah-langkah visual dari setiap iterasi.

Langkah 1:

- Array awal: [64, 34, 25, 12, 22, 11, 90]
- Bandingkan 64 dengan 34, lalu tukar posisinya karena  $64 > 34$ .
- Array setelah iterasi pertama: [34, 64, 25, 12, 22, 11, 90]

Langkah 2:

- Bandingkan 64 dengan 25, lalu tukar posisinya karena  $64 > 25$ .
- Bandingkan 64 dengan 12, lalu tukar posisinya karena  $64 > 12$ .
- Bandingkan 64 dengan 22, lalu tukar posisinya karena  $64 > 22$ .
- Bandingkan 64 dengan 11, lalu tukar posisinya karena  $64 > 11$ .
- Array setelah iterasi kedua: [34, 25, 12, 22, 11, 64, 90]

Langkah 3:

- Bandingkan 34 dengan 25, lalu tukar posisinya karena  $34 > 25$ .
- Bandingkan 34 dengan 12, lalu tukar posisinya karena  $34 > 12$ .
- Bandingkan 34 dengan 22, lalu tukar posisinya karena  $34 > 22$ .
- Bandingkan 34 dengan 11, lalu tukar posisinya karena  $34 > 11$ .
- Array setelah iterasi ketiga: [25, 12, 22, 11, 34, 64, 90]

Langkah 4:

- Bandingkan 25 dengan 12, lalu tukar posisinya karena  $25 > 12$ .
- Bandingkan 25 dengan 22, lalu tukar posisinya karena  $25 > 22$ .
- Bandingkan 25 dengan 11, lalu tukar posisinya karena  $25 > 11$ .
- Array setelah iterasi keempat: [12, 22, 11, 25, 34, 64, 90]

Langkah 5:

- Bandingkan 22 dengan 11, lalu tukar posisinya karena  $22 > 11$ .
- Array setelah iterasi kelima: [12, 11, 22, 25, 34, 64, 90]

Langkah 6:

- Tidak ada pertukaran yang terjadi karena semua elemen sudah berada pada posisi yang benar.

Array akhir setelah semua iterasi: [11, 12, 22, 25, 34, 64, 90]

## 6.2 Insertion Sort

Seperti halnya bubble sort, algoritme penyisipan penyortiran juga mudah diterapkan dan dipahami. Namun, berbeda dengan bubble sort, algoritme ini membangun daftar yang terurut satu per satu dengan cara membandingkan setiap elemen dengan sisa daftar dan menyisipkannya ke posisi yang sesuai. Proses "penyisipan" ini menjadi nama algoritma tersebut.

Salah satu cara untuk menjelaskan jenis penyisipan ini adalah dengan analogi mengenai cara Anda menyusun setumpuk kartu. Bayangkan Anda memiliki sejumlah kartu di tangan dan Anda ingin menyusunnya secara berurutan. Anda akan membandingkan setiap kartu dengan yang lain secara berurutan, mencari posisi yang tepat untuk setiap kartu. Ketika Anda menemukan posisi yang benar, Anda akan

menyisipkan kartu tersebut di tempatnya yang sesuai, kemudian melanjutkan proses yang sama dengan kartu berikutnya hingga seluruh tumpukan terurut.

Seperti halnya bubble sort, algoritma insertion sort juga sangat mudah untuk diterapkan. Meskipun pengurutan penyisipan memiliki kompleksitas waktu  $O(n^2)$ , dalam praktiknya jauh lebih efisien dibandingkan dengan implementasi algoritma lain yang memiliki kompleksitas kuadrat seperti bubble sort.

Meskipun ada algoritma yang lebih efisien, seperti pengurutan gabungan dan Quicksort, implementasi ini seringkali bersifat rekursif dan seringkali kalah efisien saat menangani daftar yang kecil. Bahkan, beberapa implementasi Quicksort menggunakan penyisipan penyortiran secara internal jika daftar yang diurutkan cukup kecil, karena ini dapat memberikan hasil yang lebih cepat secara keseluruhan. Timsort, sebagai contoh, juga menggunakan pengurutan penyisipan secara internal untuk mengurutkan sebagian kecil dari array input.

Meskipun demikian, pengurutan penyisipan tidak praktis untuk array yang besar, sehingga memberikan kesempatan bagi algoritma lain yang dapat mengatasi masalah dengan cara yang lebih efisien dalam skala yang lebih besar.

```
▼ Insertion Sort

def insertion_sort(array):
    # Ulangi dari elemen kedua array hingga elemen terakhir
    for i in range(1, len(array)):
        # Ini adalah elemen yang ingin kita posisikan pada tempat yang benar
        key_item = array[i]

        # Inisialisasi variabel yang akan digunakan untuk menemukan posisi yang benar dari elemen yang direferensikan oleh 'key_item'
        j = i - 1

        # Jalankan daftar item (bagian kiri array) dan temukan posisi yang benar dari elemen yang direferensikan oleh 'key_item'.
        # Lakukan ini hanya jika 'key_item' lebih kecil dari nilai di dekatnya.
        while j >= 0 and array[j] > key_item:
            # Geser nilai satu posisi ke kiri dan ubah posisi j untuk menunjuk ke elemen berikutnya (dari kanan ke kiri)
            array[j + 1] = array[j]
            j -= 1

        # Setelah selesai menggeser elemen, Anda dapat memposisikan 'key_item' di lokasi yang benar
        array[j + 1] = key_item

    return array

# Contoh penggunaan
arr = [64, 34, 25, 12, 22, 11, 90]
print("Array sebelum pengurutan:", arr)
insertion_sort(arr)
print("Array setelah pengurutan:", arr)

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

Berbeda dengan pengurutan gelembung, penerapan pengurutan penyisipan ini menghasilkan daftar yang diurutkan dengan cara memindahkan item yang lebih kecil ke kiri. Mari kita jelaskan `insertion_sort()` baris per baris:

- Baris 3 menyiapkan loop yang menentukan posisi `key_item` selama setiap iterasi. Penting untuk dicatat bahwa perulangan dimulai dengan item kedua dalam daftar dan berlanjut hingga item terakhir.
- Baris 5 menginisialisasi `key_item` dengan nilai item yang akan ditempatkan oleh fungsi.
- Baris 8 menginisialisasi variabel yang akan secara berurutan menunjuk ke setiap elemen di sebelah kiri `key_item`. Variabel ini akan digunakan untuk membandingkan nilai-nilai dengan `key_item`.
- Baris 12 membandingkan `key_item` dengan setiap nilai di sebelah kirinya menggunakan loop `while`, menggeser elemen-elemen untuk memberikan ruang bagi `key_item`.
- Pada baris 27, `key_item` berada di posisi yang tepat setelah algoritma menggeser semua nilai yang lebih besar ke kanan.

Mari kita ilustrasikan langkah-langkah pengurutan penyisipan pada array dengan nilai [64, 34, 25, 12, 22, 11, 90]:

Langkah 1:

- Array awal: [64, 34, 25, 12, 22, 11, 90]
- Pertama, kita memiliki `key_item` yang berisi nilai pertama di array, yaitu 64.
- Karena ini adalah elemen pertama, tidak ada elemen di sebelah kiri yang perlu dibandingkan.

Langkah 2:

- Array setelah langkah pertama: [34, 64, 25, 12, 22, 11, 90]
- Sekarang, `key_item` adalah nilai kedua di array, yaitu 34.
- Kita membandingkan 34 dengan 64. Karena 34 lebih kecil dari 64, kita tukar posisinya.
- Kita sekarang memiliki nilai 34 pada posisi yang benar, dan tidak ada nilai di sebelah kiri yang perlu dibandingkan.

Langkah 3:

- Array setelah langkah kedua: [25, 34, 64, 12, 22, 11, 90]
- `key_item` saat ini adalah nilai ketiga di array, yaitu 25.

- Kita membandingkan 25 dengan 64. Karena 25 lebih kecil dari 64, kita tukar posisinya.
- Kita lanjutkan dengan membandingkan 25 dengan 34. Karena 25 juga lebih kecil dari 34, kita tukar posisinya.
- Sekarang, nilai 25 berada di posisi yang benar di sebelah kiri, dan tidak ada nilai yang lebih besar di sebelah kanan.

Langkah 4:

- Array setelah langkah ketiga: [12, 25, 34, 64, 22, 11, 90]
- `key_item` saat ini adalah nilai keempat di array, yaitu 12.
- Kita membandingkan 12 dengan 64, 34, dan 25, dan karena 12 lebih kecil dari ketiganya, kita geser semua nilai ke kanan.
- Sekarang, nilai 12 berada di posisi yang benar di sebelah kiri.

Langkah 5:

- Array setelah langkah keempat: [12, 22, 25, 34, 64, 11, 90]
- `key_item` saat ini adalah nilai kelima di array, yaitu 22.
- Kita membandingkan 22 dengan 64, 34, dan 25. Karena 22 lebih kecil dari ketiganya, kita geser semua nilai ke kanan.
- Sekarang, nilai 22 berada di posisi yang benar di sebelah kiri.

Langkah 6:

- Array setelah langkah kelima: [11, 12, 22, 25, 34, 64, 90]
- `key_item` saat ini adalah nilai keenam di array, yaitu 11.
- Kita membandingkan 11 dengan semua nilai sebelumnya. Karena 11 adalah nilai terkecil, ia tetap berada di posisi awalnya.
- Array sekarang terurut

### 6.3 Merge Sort

Merge sort merupakan algoritma pengurutan yang sangat efisien, yang didasarkan pada pendekatan divide-and-conquer, suatu teknik algoritmik canggih yang digunakan untuk memecahkan masalah yang kompleks.



Untuk memahami divide-and-conquer dengan benar, konsep rekursi harus dipahami terlebih dahulu. Rekursi melibatkan pembagian masalah menjadi sub-masalah yang lebih kecil hingga ukuran masalah tersebut cukup kecil untuk diatasi. Dalam pemrograman, rekursi sering kali dinyatakan dengan menggunakan pemanggilan fungsi itu sendiri.

Implementasi algoritma pengurutan gabungan melibatkan dua komponen utama:

- Sebuah fungsi yang membagi input secara rekursif menjadi dua bagian.
- Sebuah fungsi yang menggabungkan dua bagian tersebut untuk menghasilkan array yang terurut.

Dengan kompleksitas runtime  $O(n \log 2 n)$ , pengurutan gabungan merupakan algoritme yang sangat efisien dan dapat diskalakan dengan baik seiring dengan bertambahnya ukuran larik masukan. Memparalelkan pengurutan gabungan juga relatif mudah karena array input dapat dibagi menjadi beberapa bagian yang dapat didistribusikan dan diproses secara paralel jika diperlukan.

Meskipun demikian, untuk daftar yang kecil, biaya waktu rekursi memungkinkan algoritme seperti pengurutan gelembung dan pengurutan penyisipan menjadi lebih cepat. Kelemahan lain dari merge sort adalah bahwa ia membuat salinan array ketika memanggil dirinya secara rekursif, dan juga membuat daftar baru di dalam fungsi `merge()` untuk mengurutkan dan menggabungkan kedua bagian masukan. Hal ini menyebabkan pengurutan gabungan menggunakan lebih banyak memori daripada pengurutan gelembung dan pengurutan penyisipan, yang keduanya dapat mengurutkan daftar pada tempatnya.

Karena keterbatasan ini, pengurutan gabungan mungkin tidak ideal untuk mengurutkan daftar besar di perangkat keras dengan memori terbatas.

## ▼ Merge Sort

```
def merge(left, right):
    # Jika array pertama kosong, maka tidak ada yang perlu digabungkan,
    # dan Anda dapat mengembalikan array kedua sebagai hasilnya
    if len(left) == 0:
        return right
    # Jika array kedua kosong, maka tidak ada yang perlu digabungkan,
    # dan Anda dapat mengembalikan array pertama sebagai hasilnya
    if len(right) == 0:
        return left
    result = []
    index_left = index_right = 0
    # Sekarang telusuri kedua array hingga semua elemen berhasil masuk ke dalam
    # array yang dihasilkan
    while len(result) < len(left) + len(right):
        # Elemen perlu diurutkan untuk menambahkannya ke array yang dihasilkan,
        # jadi Anda perlu memutuskan apakah akan mendapatkan elemen berikutnya
        # dari array pertama atau kedua
        if left[index_left] <= right[index_right]:
            result.append(left[index_left])
            index_left += 1
        else:
            result.append(right[index_right])
            index_right += 1
        # Jika Anda mencapai akhir dari salah satu array, maka Anda dapat menambahkan
        # elemen yang tersisa dari array lain ke hasil dan memutus perulangan
        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(array):
    # Jika array masukan berisi kurang dari dua elemen, kembalikan sebagai hasil fungsinya
    if len(array) < 2:
        return array
    midpoint = len(array) // 2
    # Urutkan array dengan membagi input secara rekursif menjadi dua bagian yang sama,
    # mengurutkan masing-masing bagian dan menggabungkannya menjadi hasil akhir
    return merge(
        left=merge_sort(array[:midpoint]),
        right=merge_sort(array[midpoint:]))

# Contoh penggunaan
arr = [64, 34, 25, 12, 22, 11, 90]
print("Array sebelum pengurutan:", arr)
merge_sort(arr)
print("Array setelah pengurutan:", arr)
```

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

Berikut adalah ilustrasi langkah demi langkah dari pengurutan gabungan (merge sort) pada array dengan nilai [64, 34, 25, 12, 22, 11, 90]:

#### Langkah 1: Pembagian Array Menjadi Subarray yang Lebih Kecil

- Array awal: [64, 34, 25, 12, 22, 11, 90]
- Array dibagi menjadi dua bagian:
  - Bagian 1: [64, 34, 25, 12]
  - Bagian 2: [22, 11, 90]

#### Langkah 2: Pengurutan Subarray

- Pengurutan dilakukan pada setiap subarray secara terpisah.
  - Bagian 1 terurut: [12, 25, 34, 64]
  - Bagian 2 terurut: [11, 22, 90]

#### Langkah 3: Penggabungan Subarray yang Terurut

- Subarray yang terurut digabungkan menjadi array yang utuh.
  - Array terurut: [11, 12, 22, 25, 34, 64, 90]

## Latihan

---

Lakukan pengurutan array dengan nilai [64, 34, 25, 12, 22, 11, 90] menggunakan algoritma quicksort dengan bahasa pemrograman python dengan ilustrasi sebagai berikut :

#### Langkah 1: Pemilihan Pivotal

- Kita memilih elemen pivot. Dalam contoh ini, kita akan menggunakan elemen terakhir dari array sebagai pivot, yaitu 90.

#### Langkah 2: Pembagian Array Menggunakan Pivot

- Array dibagi menjadi dua bagian, dengan elemen-elemen yang lebih kecil dari pivot berada di sebelah kiri, dan elemen-elemen yang lebih besar berada di sebelah kanan.
- Array setelah pembagian: [34, 25, 12, 22, 11, 64, 90]

#### Langkah 3: Pengurutan Rekursif

- Kita ulangi proses ini pada kedua bagian array yang terbagi. Kita menerapkan langkah-langkah yang sama untuk setiap subarray hingga semua elemen terurut.

#### Langkah 4: Penggabungan Subarray

- Setelah semua subarray terurut, kita menggabungkan mereka menjadi array yang utuh.
- Array terurut: [11, 12, 22, 25, 34, 64, 90]