**STRUKTUR DATA**

**FUNGSI REKURSI DAN ALGORITMA SORTING**



Oleh:

**DANUARY BIMA HAMMAM MAYFALAH**

**24091397007**

**Program Studi D4 Manajemen Informatika**

**Fakultas Vokasi**

**Universitas Negeri Surabaya**

**2025**

1. **TUGAS**
2. Rekursi dan Penerapannya dalam Pemecahan Masalah

Tujuan: Mahasiswa memahami konsep rekursi dan bagaimana rekursi digunakan untuk menyelesaikan berbagai masalah dalam pemrograman.

* Tulis dan analisis kode rekursif untuk menghitung faktorial (n = 15).
* Bandingkan implementasi rekursi dan iterasi dalam menghitung faktorial.
* Implementasikan rekursi untuk menghitung bilangan Fibonacci ke-25. • Analisis kompleksitas waktu dari algoritma rekursi Fibonacci.

1. Implementasi Rekursi dalam Python

Tujuan: Mahasiswa mampu mengimplementasikan rekursi dalam Python untuk berbagai permasalahan umum.

* Implementasikan fungsi rekursif menyelesaikan Tower of Hanoi dengan 2 cakram. • Ubah program Tower of Hanoi agar menerima input jumlah cakram dari user.

1. Algoritma Sorting

Tujuan: Mahasiswa memahami dan mengimplementasikan berbagai algoritma sorting dalam Python.

Data Input = [115, 18, 45, 29, 56, 1, 37]

* Implementasikan Bubble Sort dan analisis jumlah perbandingan serta pertukaran elemen yang terjadi.
* Implementasikan Selection Sort dan Bubble Sort. • Implementasikan juga Quick Sort, lalu bandingkan dengan sorting lainnya.

1. Implementasi Sorting dan Rekursi Menggunakan OOP

Tujuan: Mahasiswa mampu menerapkan konsep OOP dalam implementasi rekursi dan sorting.

* Modifikasi program Fibonacci menggunakan konsep class dan method.
* Implementasikan Class Sorting yang mencakup metode Bubble Sort,

Insertion Sort, dan Selection Sort dalam satu class.

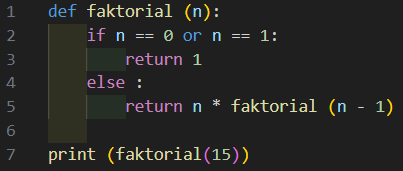
* Tambahkan Method tambahan dalam Class Sorting untuk mengurutkan dalam urutan sebaliknya, menurun (Descending).
* Bandingkan waktu eksekusi antara algoritma sorting menggunakan OOP dan tanpa OOP.

1. Perbandingan Efisiensi Algoritma Sorting

Tujuan: Mahasiswa membandingkan efisiensi berbagai algoritma sorting berdasarkankompleksitas waktu dan ruang.

* Gunakan Library Python time untuk mengukur waktu eksekusi dari setiap algoritma sorting pada data.
* Buat visualisasi grafik perbandingan efisiensi sorting menggunakan pustaka matplotlib.
* Analisis kompleksitas waktu dari masing-masing algoritma sorting dan simpulkan kapan sebaiknya menggunakan masing-masing algoritma.

1. **JAWABAN**
2. Source Code

****

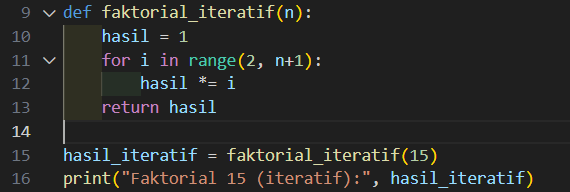
Penjelasan:

* Basis kasus: Jika n == 0 atau n == 1, kembalikan 1.
* Rekurens: Untuk n > 1, hasil = n \* faktorial\_rekursif(n-1).

Output

****

* Perbandingan Rekursi vs Iterasi dalam Menghitung Faktorial



Output

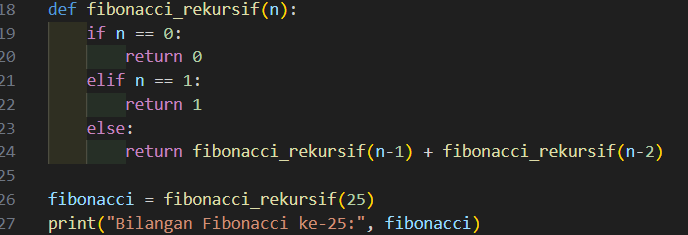


Perbandingan:

|  |  |  |
| --- | --- | --- |
| Aspek | Rekursi | Iterasi |
| Kelebihan | Lebih elegan dan dekat dengan definisi matematis. | Umumnya lebih cepat dan hemat memori. |
| Kekurangan | Stack memory bertambah dengan setiap pemanggilan, berisiko stack overflow untuk n besar. | Harus secara eksplisit mengelola perulangan. |
| Komplektibitas Waktu | O(n) | O(n) |
| Komplektibitas Ruang | O(n) (karena stack rekursif) | O(1) |

* Implementasi Rekursif untuk Menghitung Bilangan Fibonacci ke-25

Source Code



Output



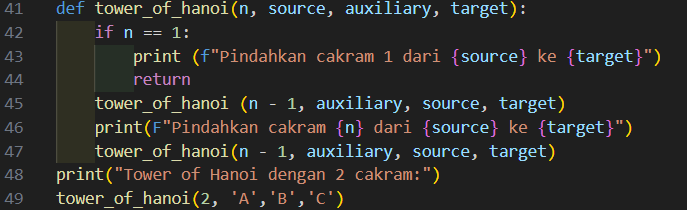
* Analisis Kompleksitas Waktu Rekursi Fibonacci

Analisis:

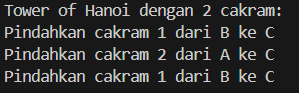
* Setiap pemanggilan fibonacci\_rekursif(n) memanggil 2 fungsi: fibonacci\_rekursif(n-1) dan fibonacci\_rekursif(n-2).
* Ini membentuk pohon biner di mana jumlah simpul ≈ 2^n.
* Kompleksitas waktu = O(2^n)  
  (sangat lambat untuk nilai n besar seperti n = 25 atau lebih).

1. Fungsi Rekursif Menyelesaikan Tower of Hanoi dengan 2 Cakram

Source code

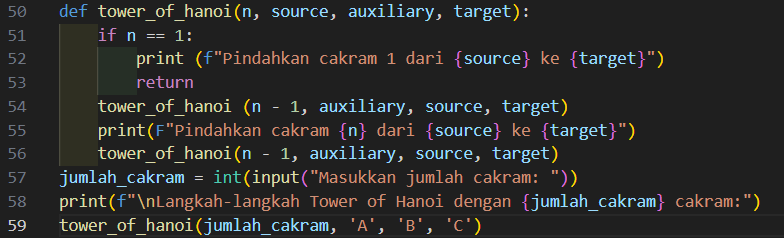


Output

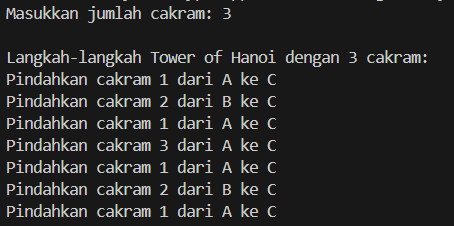


* Modifikasi Tower of Hanoi agar Menerima Input Jumlah Cakram dari User

Source code



Output



Alur Rekursi Tower of Hanoi:

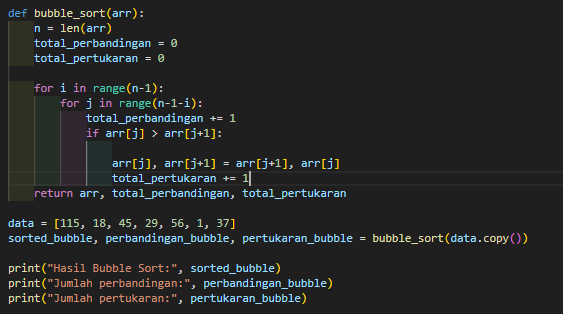
* Jika hanya ada satu cakram, langkahnya langsung dipindahkan ke tiang tujuan.

Jika lebih dari satu cakram:

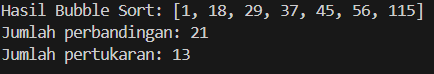
* + Pertama, pindahkan n-1 cakram ke tiang bantu.
  + Lalu, pindahkan cakram terbesar ke tiang tujuan.
  + Akhirnya, pindahkan kembali n-1 cakram dari tiang bantu ke tiang tujuan.

1. Implementasi Bubble Sort

Source Code

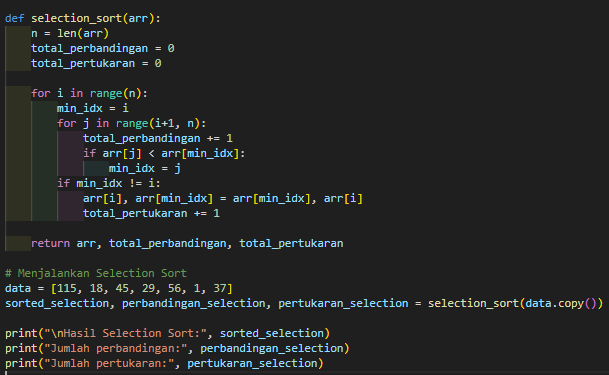


Output

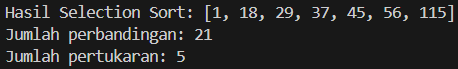


* Implementasi Selection Sort

Source Code

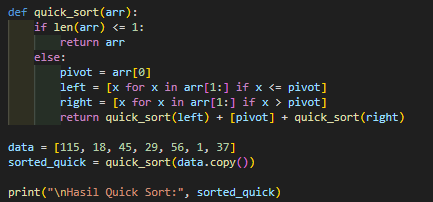


Output



* Implementasi quick sort

Source Code



Output

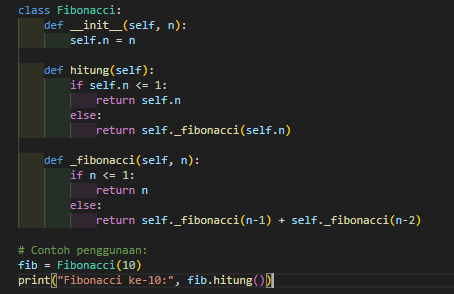


Perbandingan:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Metode | Waktu Rata-rata | Perbandingan | Pertukaran | Keterangan |
| Bubble Sort | O(n²) | Banyak | Banyak | Lambat Jika data besar |
| Selection Sort | O(n²) | Banyak | Lebih sedikit | Tetap lambat untuk data besar |
| Quick Sort | O(n log n) | Optimal | Optimal | Cepat untuk data besar |

1. Modifikasi Program Fibonacci menggunakan Konsep Class dan Method

Source Code

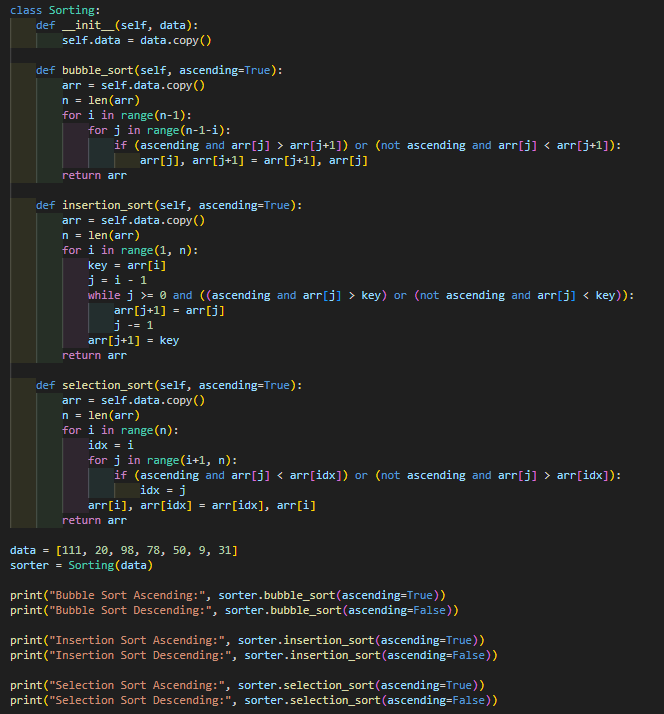


Output

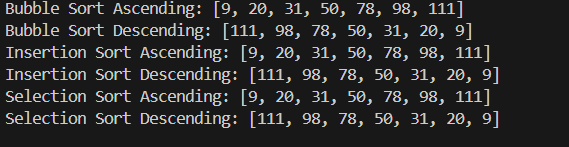


* Implementasi Class Sorting dengan Bubble Sort, Insertion Sort, dan Selection Sort

Source Code

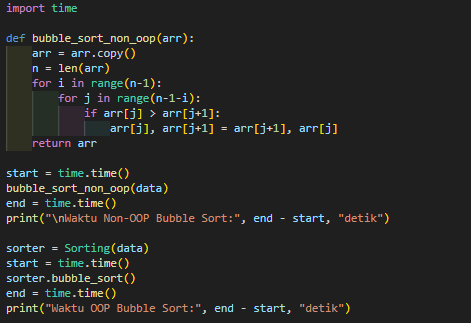


Output



* Bandingkan Waktu Eksekusi: OOP vs Non-OOP

Source Code



Output

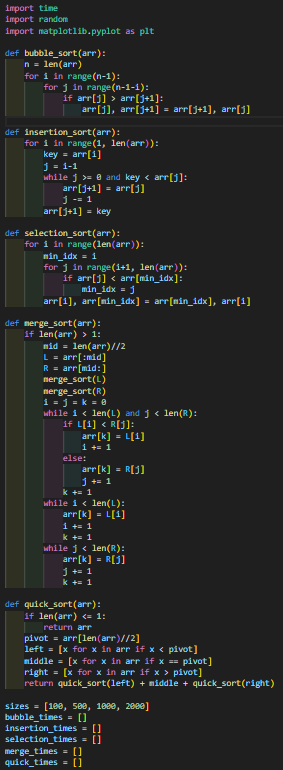
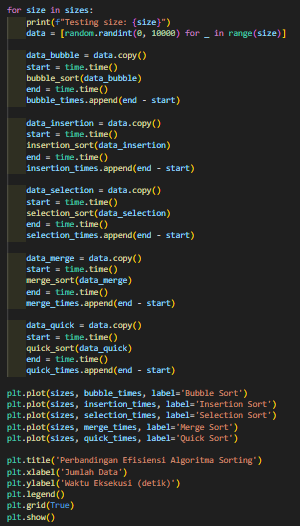


Perbandingan:

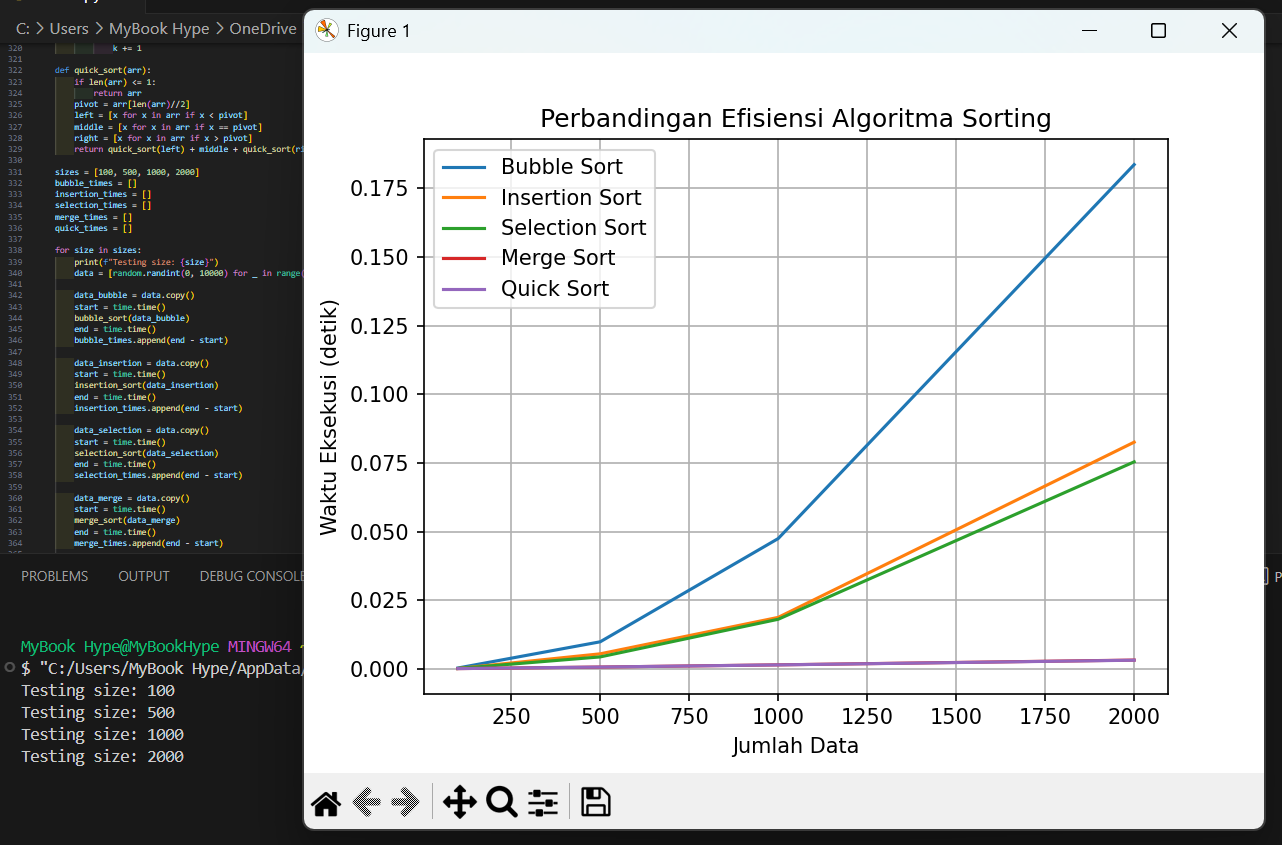
|  |  |  |
| --- | --- | --- |
| Aspek | Non-oop | Oop |
| Struktur | Lebih simpel | Lebih terorganisasi (rapih) |
| Penulisan | Sedikit kode | Sedikit lebih panjang |
| Waktu Eksekusi | Lebih cepat sedikit | Sedikit lebih lambat (karena overhead class) |
| Kelebihan | Mudah, cepat dibuat | Mudah dikembangkan, maintenance lebih bagus |

1. Mengukur Waktu Eksekusi Setiap Algoritma Sorting

Source Code

Output



Perbandingan:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Algoritma | Kompleksitas Terburuk | Kompleksitas Terbaik | Kompleksitas  Rata-rata | Catatan Penggunaan |
| Bubble Sort | O(n²) | O(n) | O(n²) | Gunakan hanya untuk dataset kecil atau hampir terurut |
| Insertion Sort | O(n²) | O(n) | O(n²) | Lebih baik untuk dataset kecil atau hampir terurut |
| Selection Sort | O(n²) | O(n²) | O(n²) | Stabil tapi lambat; dipakai kalau ingin minim pertukaran |
| Quick Sort | O(n²) | O(n log n) | O(n log n) | Sangat efisien untuk data besar, tapi kurang stabil |
| Merge Sort | O(n log n) | O(n log n) | O(n log n) | Sangat efisien untuk data besar dan stabil |