



Algoritma dan Pemrograman

#10 Meeting

Metode Sorting

Ferdian Bangkit Wijaya, S.Stat., M.Si
NIP. 199005202024061001

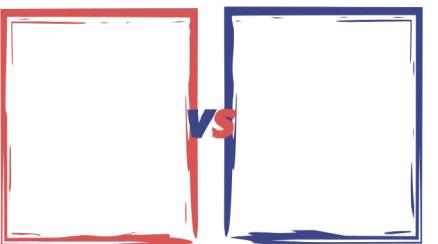




Pengertian

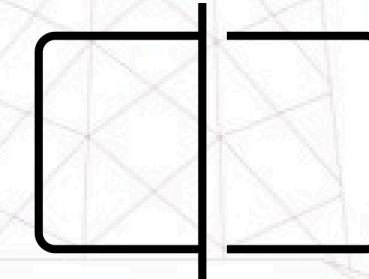
Kumpulan langkah sistematis untuk menyusun atau mengurutkan sekumpulan data/element ke dalam urutan tertentu (misalnya, dari terkecil ke terbesar)

Comparison Sort



Algoritma ini menggunakan perbandingan antar elemen untuk menentukan urutan

Non- Comparison Sort

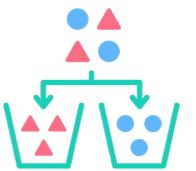


Algoritma ini mengurutkan elemen tanpa membandingkan antar elemen secara langsung.



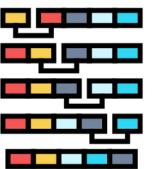
Comparison Sort

1. Bubble Sort



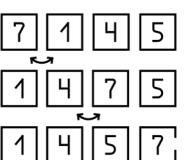
Membandingkan setiap elemen dengan elemen berikutnya, dan menukar posisinya jika urutannya salah. Proses ini diulang hingga semua elemen terurut.

2. Selection Sort

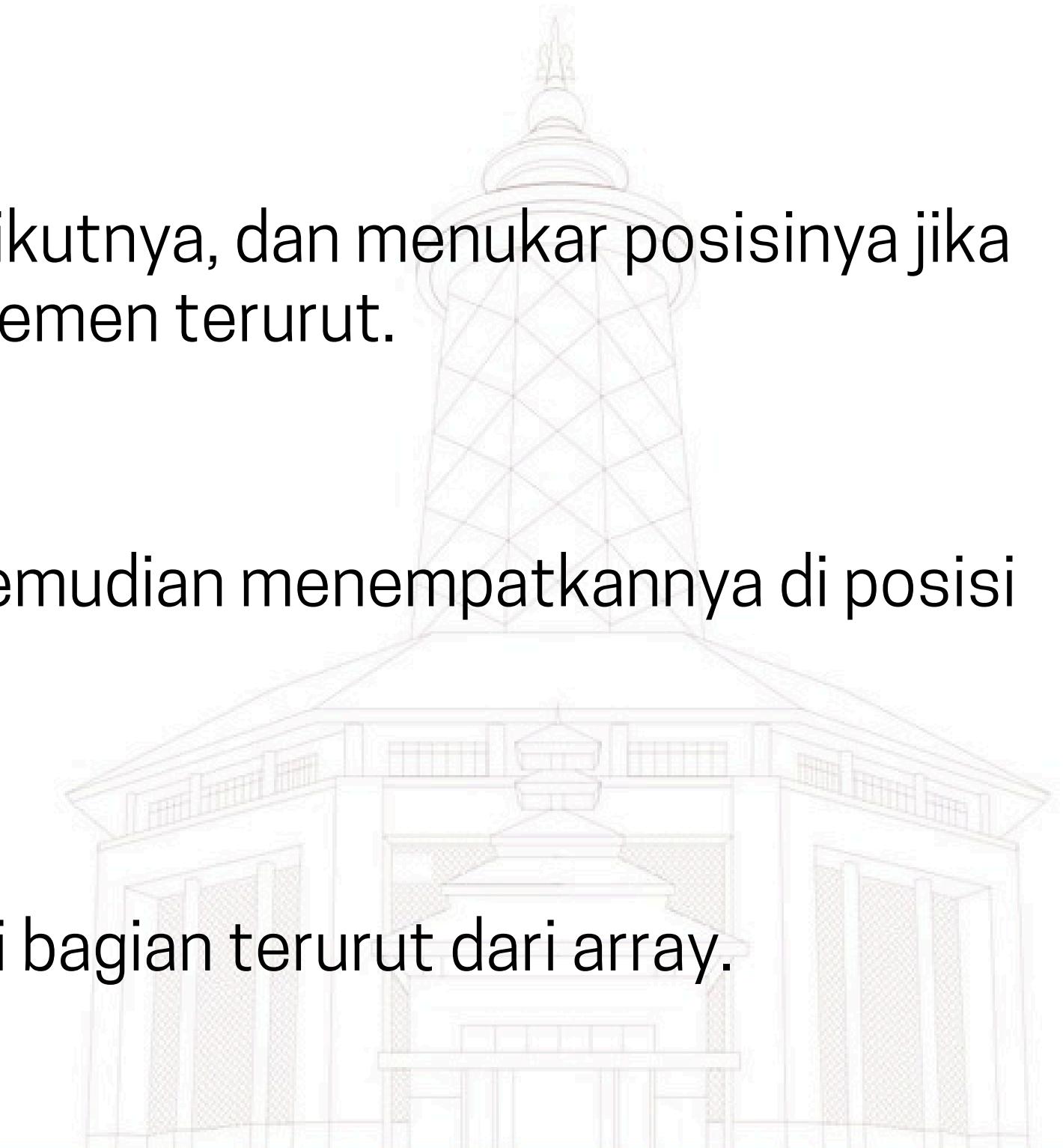


Memilih elemen terkecil dari bagian tidak terurut, kemudian menempatkannya di posisi yang sesuai pada bagian terurut.

3. Insertion Sort



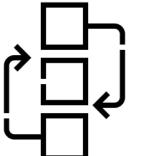
Memasukkan setiap elemen ke posisi yang benar di bagian terurut dari array.





Comparison Sort

4. Merge Sort

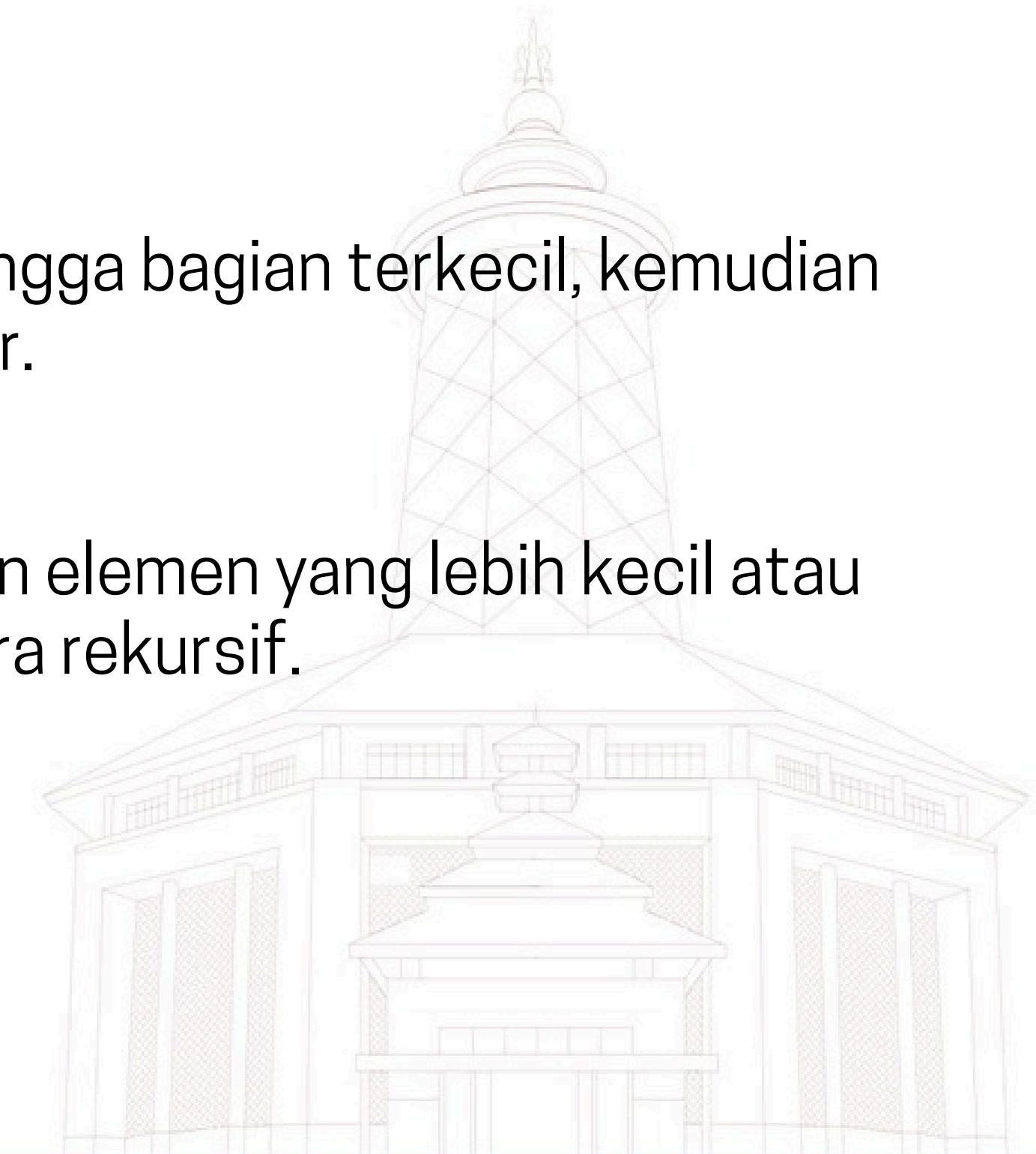


Membagi array menjadi dua bagian secara rekursif hingga bagian terkecil, kemudian menggabungkannya kembali dalam urutan yang benar.

5. Quick Sort



Memilih elemen pivot, lalu membagi array berdasarkan elemen yang lebih kecil atau lebih besar dari pivot, dan mengulang proses ini secara rekursif.

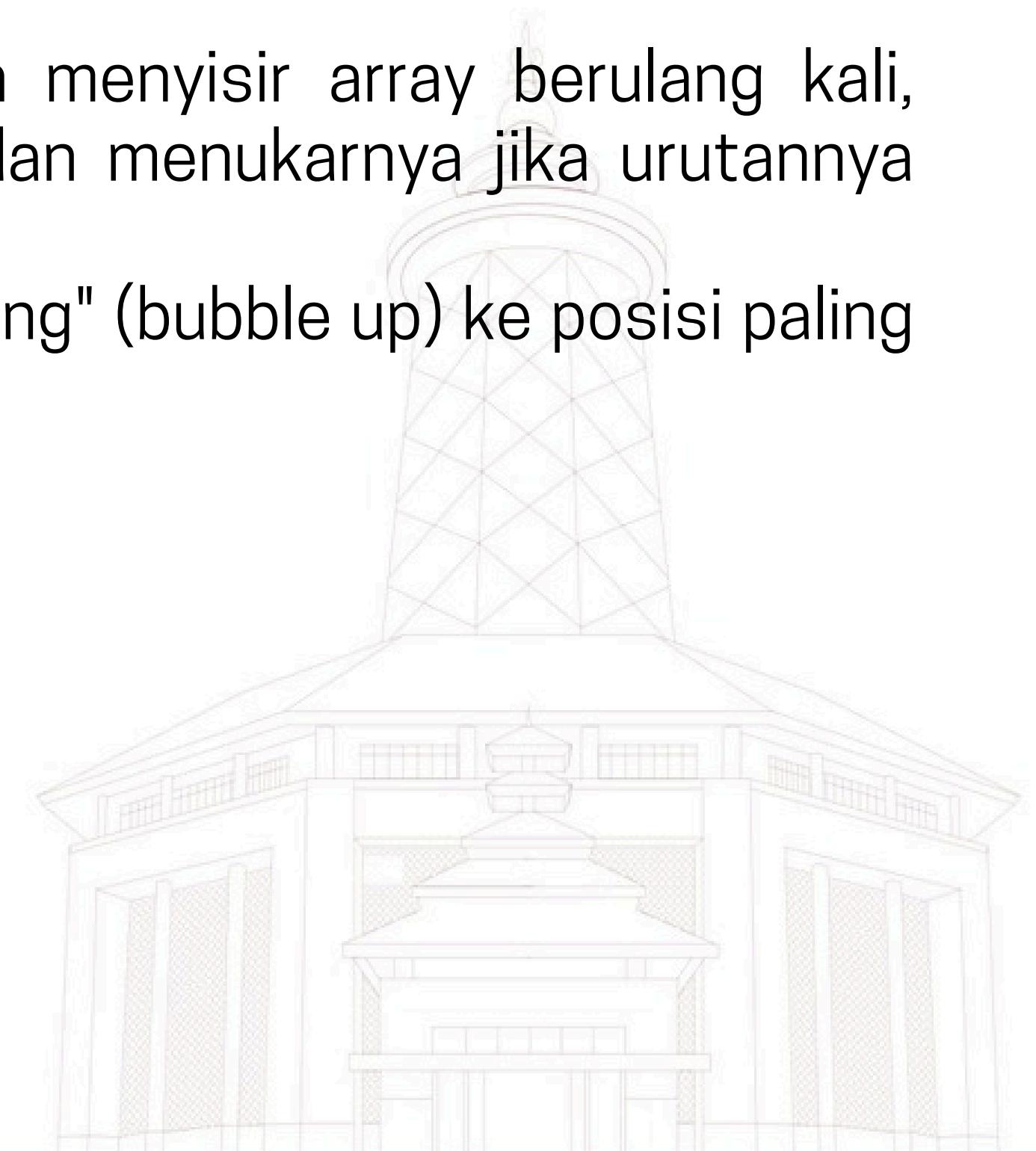




Bubble Sort

Algoritma sorting paling sederhana. Ideanya adalah menyisir array berulang kali, membandingkan dua elemen yang berdampingan, dan menukarnya jika urutannya salah (misal, kiri > kanan).

Proses ini membuat elemen terbesar "menggelembung" (bubble up) ke posisi paling kanan (posisi akhirnya) di setiap akhir iterasi.





Bubble Sort

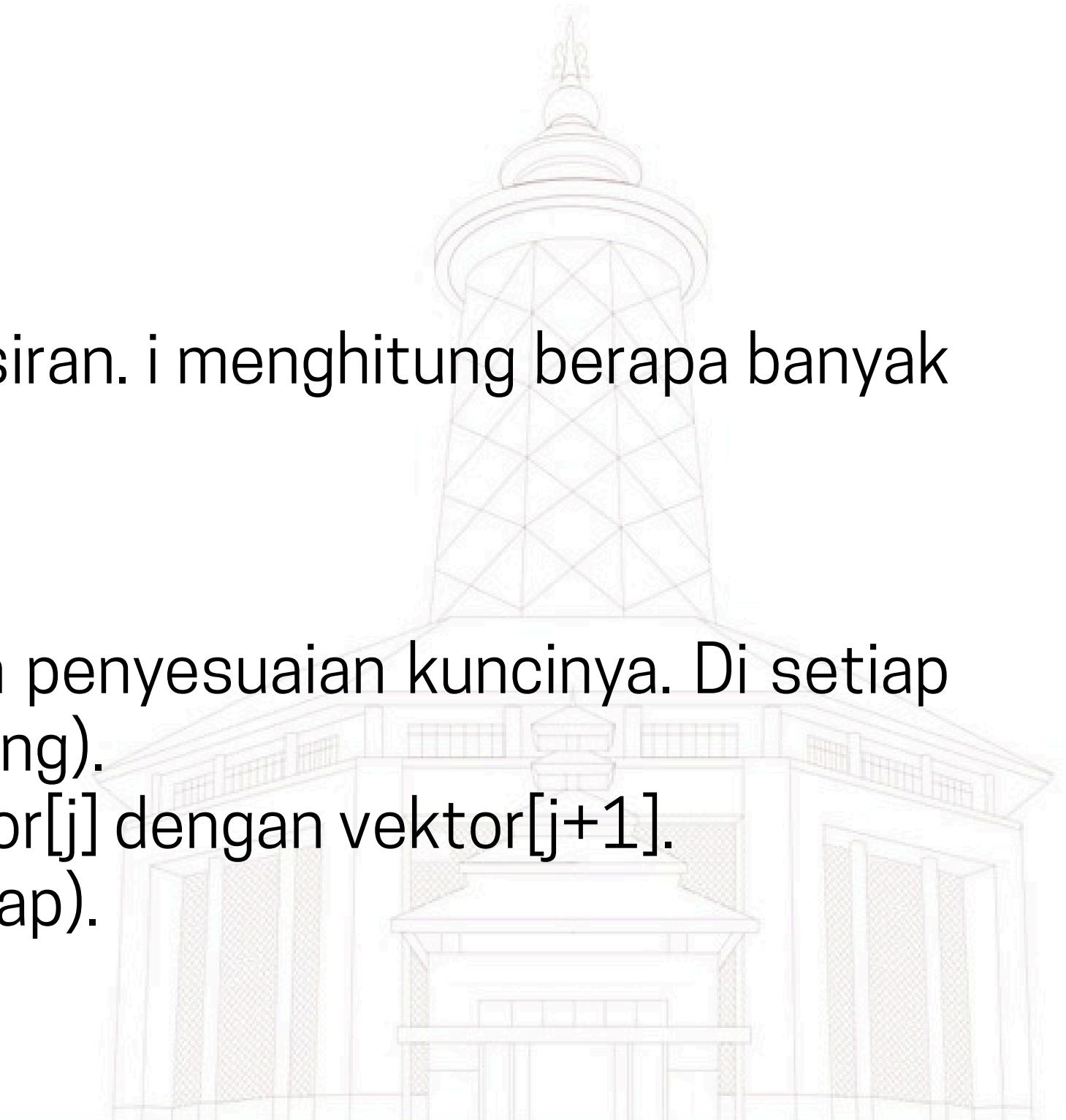
Langkah Penyelesaian Algoritma (Indeks 1)

1. Loop Luar (Iterasi - i):

- Loop ini berjalan dari $i = 1$ hingga $n-1$.
- Tugasnya adalah mengulangi proses penyisiran. i menghitung berapa banyak elemen yang sudah ter-sortir di akhir array.

2. Loop Dalam (Perbandingan & Tukar - j):

- Loop ini bertugas melakukan penyisiran.
- Ia berjalan dari $j = 1$ hingga $n - i$. (Ini adalah penyesuaian kuncinya. Di setiap iterasi i , jangkauan j berkurang 1 dari belakang).
- Di dalam loop ini, kita membandingkan vektor[j] dengan vektor[j+1].
- Jika vektor[j] > vektor[j+1], maka tukar (swap).





Bubble Sort (Min-Max)

Vektor Awal: [7, 5, 1, 9, 3] ($n = 5$)

Iterasi 1 ($i = 1$)

- $j=1$: [**7**, **5**, 1, 9, 3] (Bandingkan vektor[1] & vektor[2])
 - Apakah $7 > 5$? Ya. Tukar.
 - Hasil: [5, 7, 1, 9, 3]
- $j=2$: [5, **7**, **1**, 9, 3] (Bandingkan vektor[2] & vektor[3])
 - Apakah $7 > 1$? Ya. Tukar.
 - Hasil: [5, 1, 7, 9, 3]
- $j=3$: [5, 1, **7**, **9**, 3] (Bandingkan vektor[3] & vektor[4])
 - Apakah $7 > 9$? Tidak.
 - Hasil: [5, 1, 7, 9, 3]
- $j=4$: [5, 1, 7, **9**, **3**] (Bandingkan vektor[4] & vektor[5])
 - Apakah $9 > 3$? Ya. Tukar.
 - Hasil: [5, 1, 7, 3, 9]

Hasil Iterasi 1: [5, 1, 7, 3, 9] (Angka 9 di vektor[5] sudah benar)

Iterasi 2 ($i = 2$)

- $j=1$: [**5**, **1**, 7, 3, 9] (Bandingkan vektor[1] & vektor[2])
 - Apakah $5 > 1$? Ya. Tukar.
 - Hasil: [1, 5, 7, 3, 9]
- $j=2$: [1, **5**, **7**, 3, 9] (Bandingkan vektor[2] & vektor[3])
 - Apakah $5 > 7$? Tidak.
 - Hasil: [1, 5, 7, 3, 9]
- $j=3$: [1, 5, **7**, **3**, 9] (Bandingkan vektor[3] & vektor[4])
 - Apakah $7 > 3$? Ya. Tukar.
 - Hasil: [1, 5, 3, 7, 9]

Hasil Iterasi 2: [1, 5, 3, 7, 9] (Angka 7 di vektor[4] sudah benar)



Bubble Sort (Min-Max)

Iterasi 3 ($i = 3$)

- $j=1: [1, 5, 3, 7, 9]$ (Bandangkan vektor[1] & vektor[2])
 - Apakah $1 > 5$? Tidak.
 - Hasil: $[1, 5, 3, 7, 9]$
- $j=2: [1, 5, 3, 7, 9]$ (Bandangkan vektor[2] & vektor[3])
 - Apakah $5 > 3$? Ya. Tukar.
 - Hasil: $[1, 3, 5, 7, 9]$

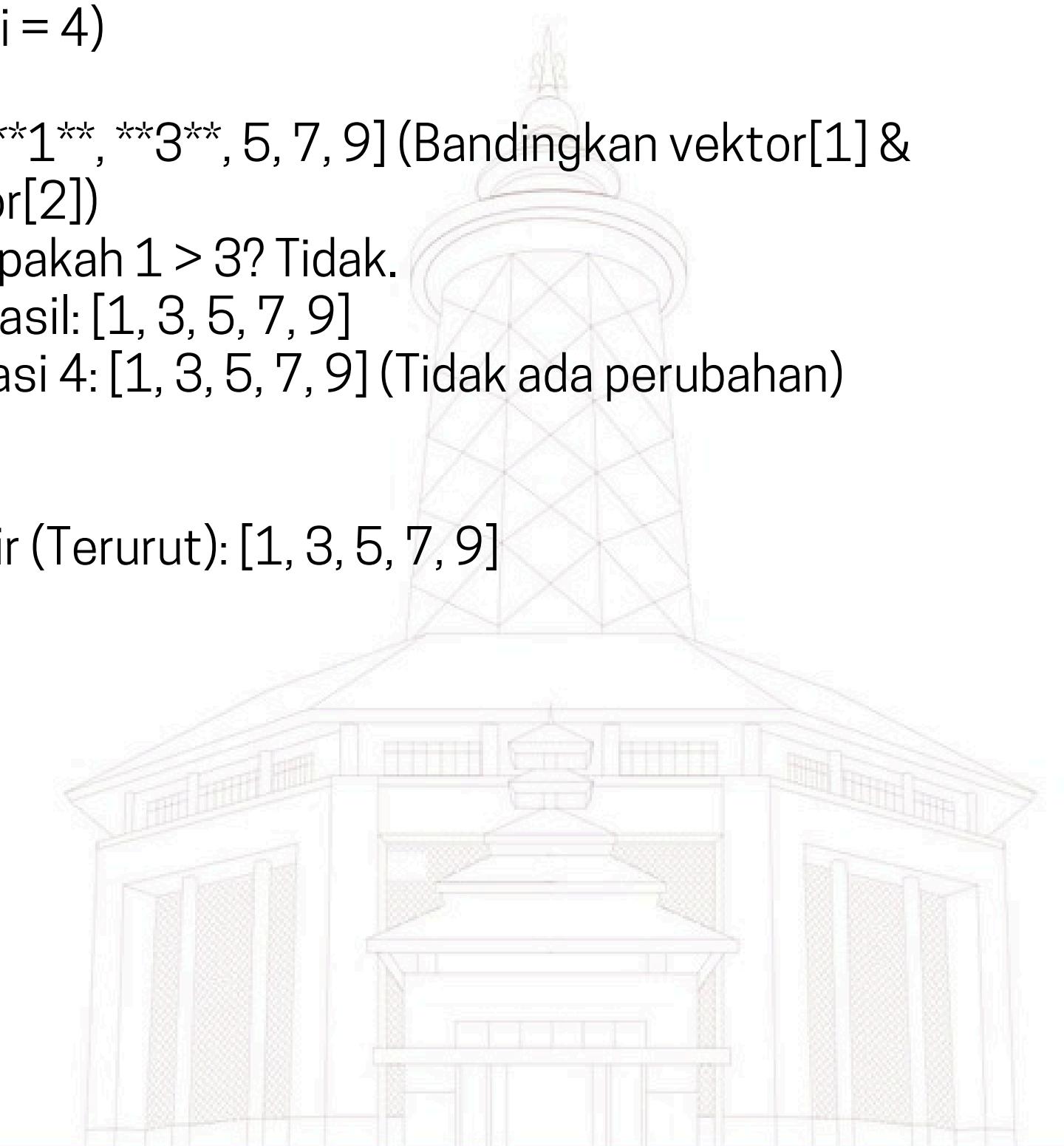
Hasil Iterasi 3: $[1, 3, 5, 7, 9]$ (Angka 5 di vektor[3] sudah benar)

Iterasi 4 ($i = 4$)

- $j=1: [1, 3, 5, 7, 9]$ (Bandangkan vektor[1] & vektor[2])
 - Apakah $1 > 3$? Tidak.
 - Hasil: $[1, 3, 5, 7, 9]$

Hasil Iterasi 4: $[1, 3, 5, 7, 9]$ (Tidak ada perubahan)

Hasil Akhir (Terurut): $[1, 3, 5, 7, 9]$





Bubble Sort (Max-Min)

Vektor Awal: [7, 5, 1, 9, 3] (n = 5)

Iterasi 1 (i = 1)

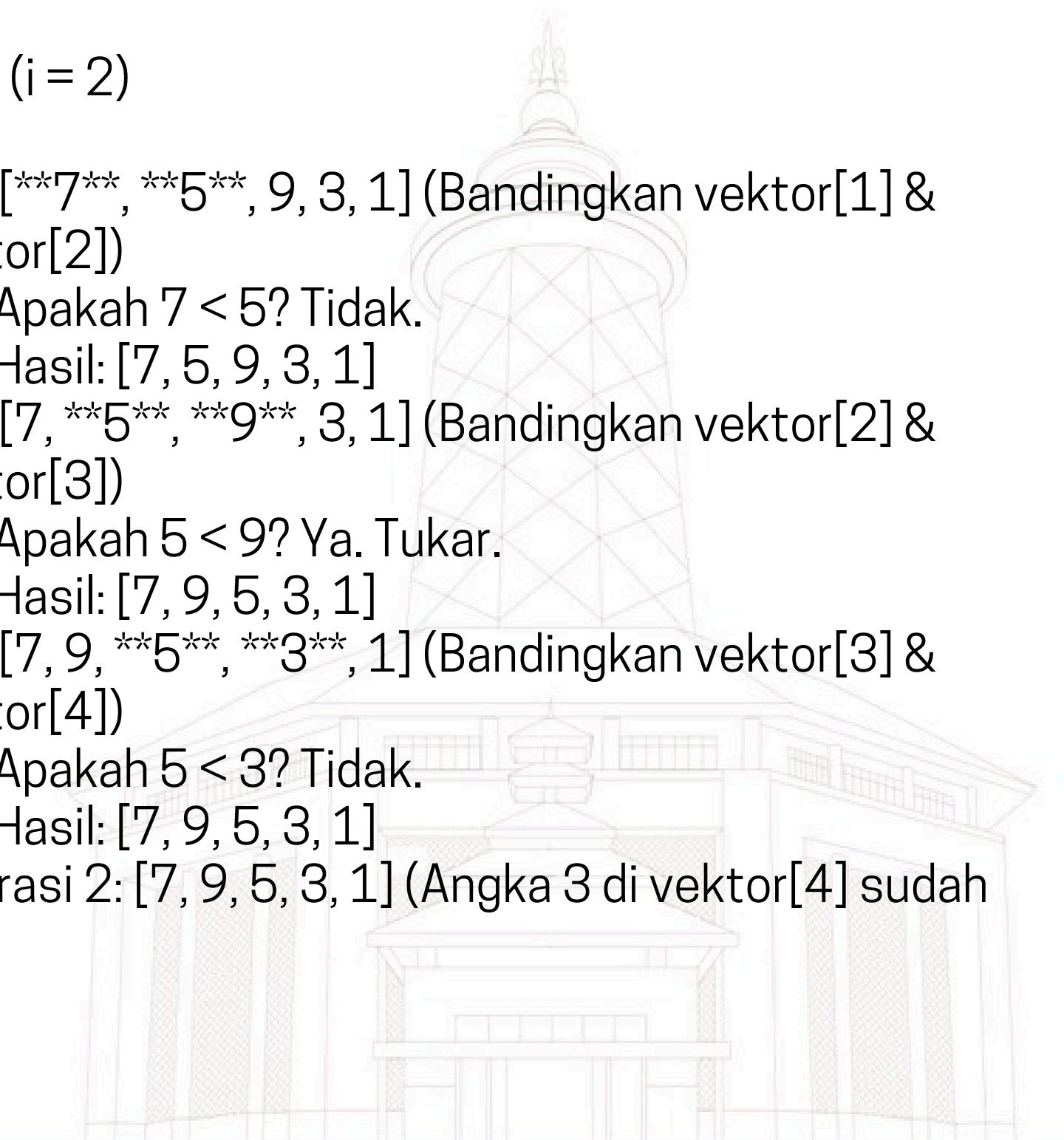
- j=1: [**7**, **5**, 1, 9, 3] (Bandingkan vektor[1] & vektor[2])
 - Apakah $7 < 5$? Tidak.
 - Hasil: [7, 5, 1, 9, 3]
- j=2: [7, **5**, **1**, 9, 3] (Bandingkan vektor[2] & vektor[3])
 - Apakah $5 < 1$? Tidak.
 - Hasil: [7, 5, 1, 9, 3]
- j=3: [7, 5, **1**, **9**, 3] (Bandingkan vektor[3] & vektor[4])
 - Apakah $1 < 9$? Ya. Tukar.
 - Hasil: [7, 5, 9, 1, 3]
- j=4: [7, 5, 9, **1**, **3**] (Bandingkan vektor[4] & vektor[5])
 - Apakah $1 < 3$? Ya. Tukar.
 - Hasil: [7, 5, 9, 3, 1]

Hasil Iterasi 1: [7, 5, 9, 3, 1] (Angka 1 di vektor[5] sudah benar)

Iterasi 2 (i = 2)

- j=1: [**7**, **5**, 9, 3, 1] (Bandingkan vektor[1] & vektor[2])
 - Apakah $7 < 5$? Tidak.
 - Hasil: [7, 5, 9, 3, 1]
- j=2: [7, **5**, **9**, 3, 1] (Bandingkan vektor[2] & vektor[3])
 - Apakah $5 < 9$? Ya. Tukar.
 - Hasil: [7, 9, 5, 3, 1]
- j=3: [7, 9, **5**, **3**, 1] (Bandingkan vektor[3] & vektor[4])
 - Apakah $5 < 3$? Tidak.
 - Hasil: [7, 9, 5, 3, 1]

Hasil Iterasi 2: [7, 9, 5, 3, 1] (Angka 3 di vektor[4] sudah benar)





Bubble Sort (Max-Min)

Iterasi 3 ($i = 3$)

- $j=1: [9, 7, 5, 3, 1]$ (Bandangkan vektor[1] & vektor[2])
 - Apakah $7 < 9$? Ya. Tukar.
 - Hasil: $[9, 7, 5, 3, 1]$
- $j=2: [9, 7, 5, 3, 1]$ (Bandangkan vektor[2] & vektor[3])
 - Apakah $7 < 5$? Tidak.
 - Hasil: $[9, 7, 5, 3, 1]$

Hasil Iterasi 3: $[9, 7, 5, 3, 1]$ (Angka 5 di vektor[3] sudah benar)

Iterasi 4 ($i = 4$)

- $j=1: [9, 7, 5, 3, 1]$ (Bandangkan vektor[1] & vektor[2])
 - Apakah $9 < 7$? Tidak.
 - Hasil: $[9, 7, 5, 3, 1]$

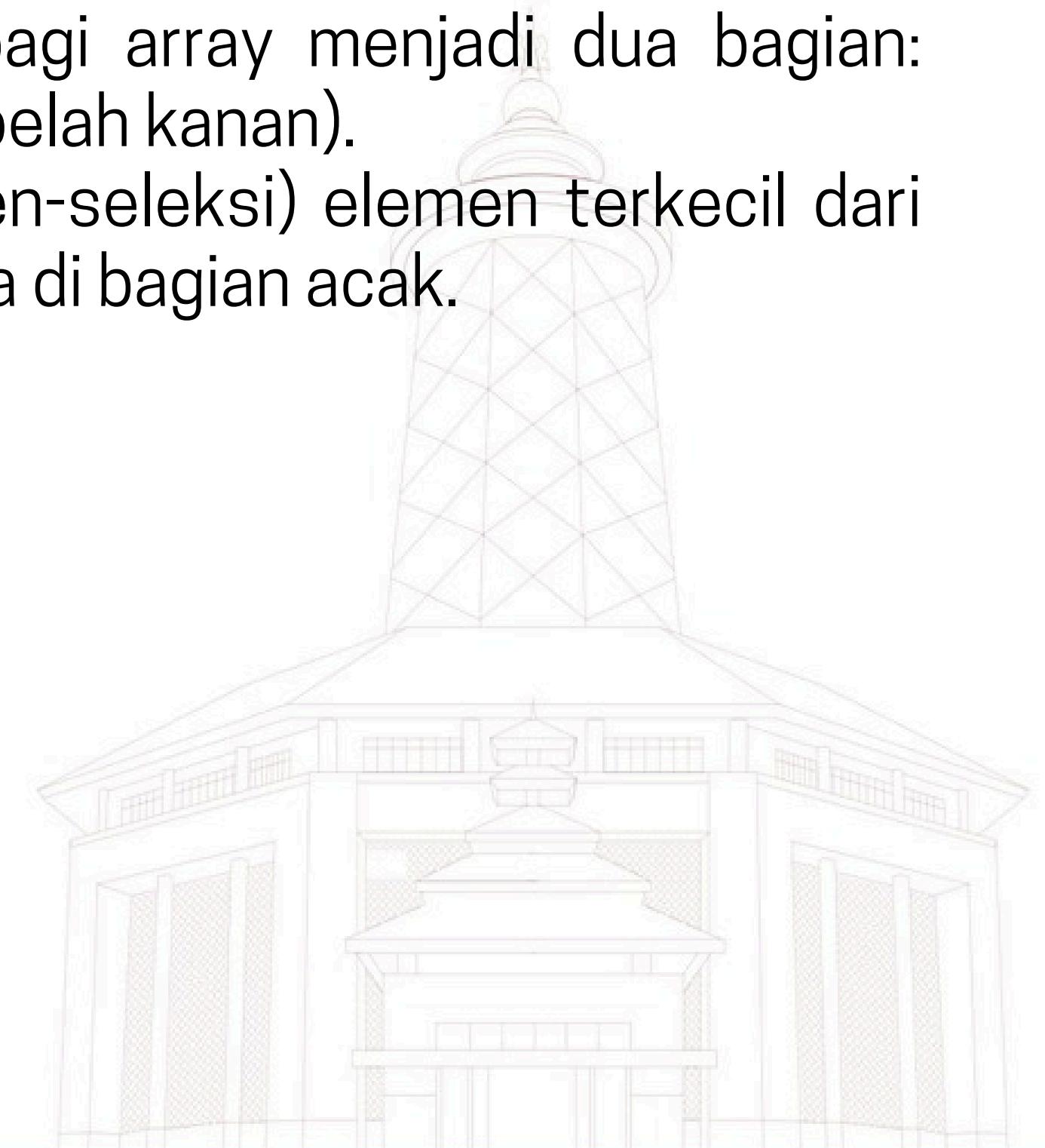
Hasil Iterasi 4: $[9, 7, 5, 3, 1]$ (Angka 7 di vektor[2] sudah benar)

Hasil Akhir (Terurut): $[9, 7, 5, 3, 1]$



Selection Sort

Selection Sort (Sortir Pilih) bekerja dengan membagi array menjadi dua bagian: bagian terurut (di sebelah kiri) dan bagian acak (di sebelah kanan). Idenya adalah, pada setiap iterasi, kita mencari (men-seleksi) elemen terkecil dari bagian acak, lalu menukarnya dengan elemen pertama di bagian acak.





Selection Sort

Langkah Penyelesaian Algoritma (Indeks 1)

1. Loop Luar (Iterasi - i):

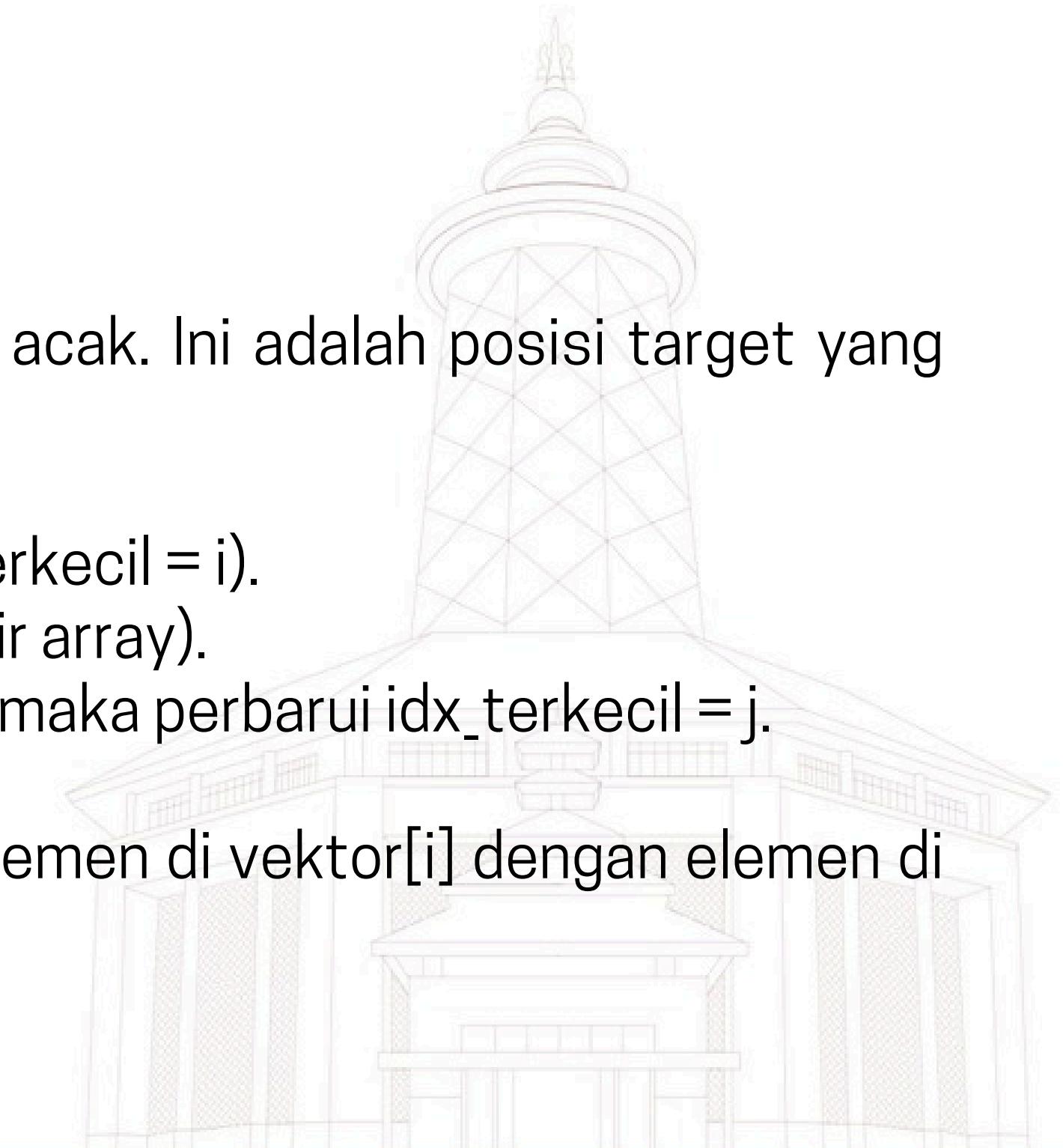
- Loop ini berjalan dari $i = 1$ hingga $n-1$.
- i adalah indeks batas antara bagian terurut dan acak. Ini adalah posisi target yang ingin kita isi.

2. Loop Dalam (Pencarian - j):

- Asumsikan elemen di i adalah yang terkecil ($idx_terkecil = i$).
- Loop ini berjalan dari $j = i + 1$ hingga n (sampai akhir array).
- Kita mencari: Jika $vektor[j] < vektor[idx_terkecil]$, maka perbarui $idx_terkecil = j$.

3. Tukar (Swap):

- Setelah loop dalam (j) selesai, kita tukar posisi elemen di $vektor[i]$ dengan elemen di $vektor[idx_terkecil]$.



Selection Sort (Min-Max)

Vektor Awal: [7, 5, 1, 9, 3] (n = 5)

Iterasi 1 (i = 1)

- Posisi Target: i = 1.
- Vektor: [**7**, 5, 1, 9, 3] (Indeks: [1, 2, 3, 4, 5])
- Proses:
 - Asumsikan idx_terkecil = 1 (nilai 7).
 - Cari di sisa array (indeks 2 s/d 5):
 - j=2: vektor[2](5) < vektor[1](7)? Ya.
idx_terkecil = 2.
 - j=3: vektor[3](1) < vektor[2](5)? Ya.
idx_terkecil = 3.
 - j=4: vektor[4](9) < vektor[3](1)? Tidak.
 - j=5: vektor[5](3) < vektor[3](1)? Tidak.
 - Loop dalam selesai. Elemen terkecil ada di idx_terkecil = 3 (nilai 1).
- Tukar: Tukar vektor[1] (nilai 7) dengan vektor[3] (nilai 1).

Hasil Iterasi 1: [1, 5, 7, 9, 3]

Iterasi 2 (i = 2)

- Posisi Target: i = 2.
- Vektor: [1 | **5**, 7, 9, 3]
- Proses:
 - Asumsikan idx_terkecil = 2 (nilai 5).
 - Cari di sisa array (indeks 3 s/d 5):
 - j=3: vektor[3](7) < vektor[2](5)? Tidak.
 - j=4: vektor[4](9) < vektor[2](5)? Tidak.
 - j=5: vektor[5](3) < vektor[2](5)? Ya.
idx_terkecil = 5.
 - Loop dalam selesai. Elemen terkecil ada di idx_terkecil = 5 (nilai 3).
- Tukar: Tukar vektor[2] (nilai 5) dengan vektor[5] (nilai 3).

Hasil Iterasi 2: [1, 3, 7, 9, 5]



Selection Sort (Min-Max)

Iterasi 3 ($i = 3$)

- Posisi Target: $i = 3$.
- Vektor: $[1, 3 | **7**, 9, 5]$
- Proses:
 - Asumsikan $idx_terkecil = 3$ (nilai 7).
 - Cari di sisa array (indeks 4 s/d 5):
 - $j=4$: $vektor[4](9) < vektor[3](7)$? Tidak.
 - $j=5$: $vektor5 < vektor[3](7)$? Ya.
 $idx_terkecil = 5$.
 - Loop dalam selesai. Elemen terkecil ada di $idx_terkecil = 5$ (nilai 5).
- Tukar: Tukar $vektor[3]$ (nilai 7) dengan $vektor[5]$ (nilai 5).

Hasil Iterasi 3: $[1, 3, 5, 9, 7]$

Iterasi 4 ($i = 4$)

- Posisi Target: $i = 4$.
- Vektor: $[1, 3, 5 | **9**, 7]$
- Proses:
 - Asumsikan $idx_terkecil = 4$ (nilai 9).
 - Cari di sisa array (indeks 5):
 - $j=5$: $vektor[5](7) < vektor[4](9)$? Ya.
 $idx_terkecil = 5$.
 - Loop dalam selesai. Elemen terkecil ada di $idx_terkecil = 5$ (nilai 7).
- Tukar: Tukar $vektor[4]$ (nilai 9) dengan $vektor[5]$ (nilai 7).

Hasil Iterasi 4: $[1, 3, 5, 7, 9]$



Insertion Sort

Insertion Sort (Sortir Sisip) bekerja dengan membagi array menjadi dua bagian: bagian terurut (di kiri) dan bagian acak (di kanan). Algoritma ini mengambil elemen pertama dari bagian acak, lalu menyisipkannya (insert) ke posisi yang tepat di dalam bagian terurut.

Untuk melakukan ini, ia menggeser (shift) elemen-elemen di bagian terurut yang lebih besar dari elemen yang akan disisipkan.



Insertion Sort

Langkah Penyelesaian Algoritma (Indeks 1)

1. Loop Luar (Iterasi - i):

- Loop ini dimulai dari elemen kedua ($i = 2$) hingga n . (Kita asumsikan vektor[1] sudah "terurut" dengan sendirinya).
- i adalah penunjuk elemen yang akan kita sisipkan.

2. Simpan key:

- $key = vektor[i]$. Simpan nilai yang akan disisipkan.
- $j = i - 1$. j adalah penunjuk untuk menyisir bagian terurut (di sebelah kiri i), dari kanan ke kiri.

3. Loop Dalam (Pergeseran - while):

- Selama $j > 0$ (kita belum sampai ujung kiri) DAN $vektor[j] > key$ (elemen di bagian terurut lebih besar dari key):
 - Geser (Shift): $vektor[j + 1] = vektor[j]$. (Geser elemen yang lebih besar itu ke kanan untuk memberi ruang).
 - Mundur: $j = j - 1$. (Pindah untuk mengecek elemen di kirinya lagi).

4. Sisipkan (Insert):

- Setelah loop while berhenti, kita telah menemukan posisi yang tepat.
- $vektor[j + 1] = key$. Masukkan key ke "lubang" yang sudah kita buat.



Insertion Sort (Min-Max)

Vektor Awal: [7, 5, 1, 9, 3] (n = 5)

Iterasi 1 (i = 2)

- Vektor: [7 | **5**, 1, 9, 3]
- key = vektor[2] (yaitu 5). j mulai dari 1.
- Proses while:
 - j=1: Apakah vektor[1](7) > key (5)? Ya.
 - Geser: vektor[2] = vektor[1]. Vektor menjadi: [7, 7, 1, 9, 3]
 - j menjadi 0.
- Loop while berhenti (karena j tidak > 0).
- Sisipkan: vektor[j + 1] (yaitu vektor[1]) = key (5).

Hasil Iterasi 1: [5, 7, 1, 9, 3]

Iterasi 2 (i = 3)

- Vektor: [5, 7 | **1**, 9, 3]
- key = vektor[3] (yaitu 1). j mulai dari 2.
- Proses while:
 - j=2: Apakah vektor[2](7) > key (1)? Ya.
 - Geser: vektor[3] = vektor[2]. Vektor menjadi: [5, 7, 7, 9, 3]
 - j menjadi 1.
 - j=1: Apakah vektor[1](5) > key (1)? Ya.
 - Geser: vektor[2] = vektor[1]. Vektor menjadi: [5, 5, 7, 9, 3]
 - j menjadi 0.
- Loop while berhenti.
- Sisipkan: vektor[j + 1] (yaitu vektor[1]) = key (1).

Hasil Iterasi 2: [1, 5, 7, 9, 3]



Insertion Sort (Min-Max)

Iterasi 3 ($i = 4$)

- Vektor: $[1, 5, 7 | **9**, 3]$
- key = vektor[4] (yaitu 9). j mulai dari 3.
- Proses while:
 - $j=3$: Apakah vektor[3] (7) > key (9)? Tidak.
- Loop while langsung berhenti.
- Sisipkan: vektor[j + 1] (yaitu vektor[4]) = key (9).

Hasil Iterasi 3: $[1, 5, 7, 9, 3]$ (Tidak ada pergeseran)

Iterasi 4 ($i = 5$)

- Vektor: $[1, 5, 7, 9 | **3**]$
- key = vektor[5] (yaitu 3). j mulai dari 4.
- Proses while:
 - $j=4$: Apakah vektor[4] (9) > key (3)? Ya.
 - Geser: vektor[5] = vektor[4]. Vektor menjadi: $[1, 5, 7, 9, 9]$
 - j menjadi 3.
 - $j=3$: Apakah vektor[3] (7) > key (3)? Ya.
 - Geser: vektor[4] = vektor[3]. Vektor menjadi: $[1, 5, 7, 7, 9]$
 - j menjadi 2.
 - $j=2$: Apakah vektor[2] (5) > key (3)? Ya.
 - Geser: vektor[3] = vektor[2]. Vektor menjadi: $[1, 5, 5, 7, 9]$
 - j menjadi 1.
 - $j=1$: Apakah vektor[1] (1) > key (3)? Tidak.
- Loop while berhenti.
- Sisipkan: vektor[j + 1] (yaitu vektor[2]) = key (3).

Hasil Iterasi 4: $[1, 3, 5, 7, 9]$



Merge Sort

Merge Sort adalah algoritma Divide and Conquer (D&C). Logikanya adalah memecah (Divide) array secara rekursif menjadi sub-array yang lebih kecil hingga setiap sub-array hanya memiliki satu elemen (ini adalah base case). Kemudian, ia menggabungkan (Combine/Merge) sub-array yang sudah terurut itu kembali, satu per satu, hingga menjadi satu array utuh yang terurut.



Merge Sort

Langkah Penyelesaian Algoritma (Indeks 1)

Algoritma ini pada dasarnya adalah dua fungsi:

1. Fungsi Utama (merge_sort):

- Divide: Menerima sebuah vektor. Jika panjang vektor ($n > 1$):
 - Tentukan titik tengah: $tengah = \text{floor}(n / 2)$.
 - Buat sub-vektor kiri: $kiri = \text{vektor}[1:tengah]$.
 - Buat sub-vektor kanan: $kanan = \text{vektor}[(tengah + 1):n]$.
- Conquer: Panggil merge_sort secara rekursif untuk kedua bagian:
 - $kiri_terurut = \text{merge_sort}(kiri)$
 - $kanan_terurut = \text{merge_sort}(kanan)$
- Combine: Panggil fungsi merge untuk menggabungkan hasilnya:
 - $hasil = \text{merge}(kiri_terurut, kanan_terurut)$
- Kembalikan hasil.
- Jika $n = 1$, kembalikan vektor itu sendiri (karena sudah terurut).

2. Fungsi Pembantu (merge):

- Menerima dua array yang sudah terurut (kiri dan kanan).
- Membuat satu array hasil yang kosong.
- Membandingkan elemen pertama kiri dan kanan. Ambil yang terkecil, masukkan ke hasil, dan geser penunjuk (indeks) di array tersebut.
- Ulangi terus hingga salah satu array habis.
- Masukkan semua sisa elemen dari array yang belum habis ke hasil.
- Kembalikan hasil.



Merge Sort (Min-Max)

Vektor Awal: [7, 5, 1, 9, 3] (n = 5)

Tahap 1: Divide (Memecah)

Proses ini terjadi top-down hingga base case (1 elemen).

1. [7, 5, 1, 9, 3] (n=5) → tengah = floor(5/2) = 2
2. Pecah jadi kiri = [7, 5] dan kanan = [1, 9, 3]
3. [7, 5] (n=2) → tengah = 1
4. Pecah jadi kiri = [7] (Base case) dan kanan = [5] (Base case)
5. [1, 9, 3] (n=3) → tengah = 1
6. Pecah jadi kiri = [1] (Base case) dan kanan = [9, 3]
7. [9, 3] (n=2) → tengah = 1
8. Pecah jadi kiri = [9] (Base case) dan kanan = [3] (Base case)

Tahap 2: Combine (Menggabungkan)

Proses ini terjadi bottom-up, mengurutkan sambil menggabung.

1. Combine [7] dan [5]:
 - Bandingkan 7 dan 5. 5 lebih kecil.
 - Hasil: [5, 7]
2. Combine [9] dan [3]:
 - Bandingkan 9 dan 3. 3 lebih kecil.
 - Hasil: [3, 9]
3. Combine [1] dan [3, 9]:
 - Bandingkan 1 dan 3. 1 lebih kecil.
 - Array [1] habis. Masukkan sisa [3, 9].
 - Hasil: [1, 3, 9]



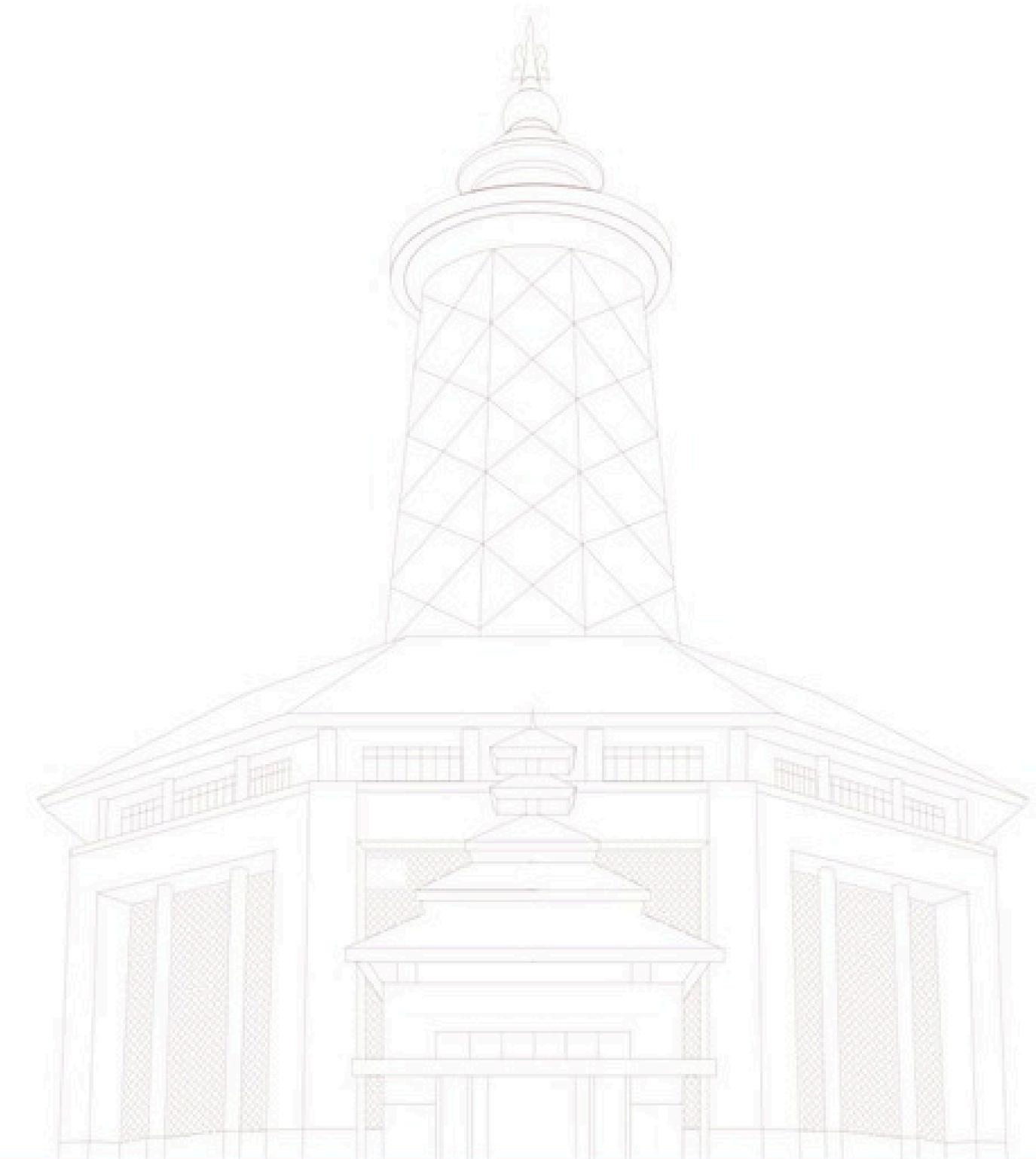
Merge Sort (Min-Max)

(Lanjutan) Tahap 2: Combine (Menggabungkan)

4. Combine [5, 7] dan [1, 3, 9] (Langkah Final):

- Bandingkan 5 dan 1. 1 lebih kecil. → Hasil: [1]
- Bandingkan 5 dan 3. 3 lebih kecil. → Hasil: [1, 3]
- Bandingkan 5 dan 9. 5 lebih kecil. → Hasil: [1, 3, 5]
- Bandingkan 7 dan 9. 7 lebih kecil. → Hasil: [1, 3, 5, 7]
- Array [5, 7] habis. Masukkan sisa [9].
- Hasil: [1, 3, 5, 7, 9]

Hasil Akhir (Terurut): [1, 3, 5, 7, 9]





Quick Sort

Quick Sort bekerja dengan memilih satu elemen sebagai Pivot. Kemudian, ia melakukan partisi (Divide), yaitu menyusun ulang array sehingga semua elemen yang lebih kecil dari Pivot berada di kirinya, dan semua elemen yang lebih besar berada di kanannya. Pivot itu sendiri akan berada di posisi finalnya. Proses ini diulang secara rekursif untuk sub-array kiri dan kanan.

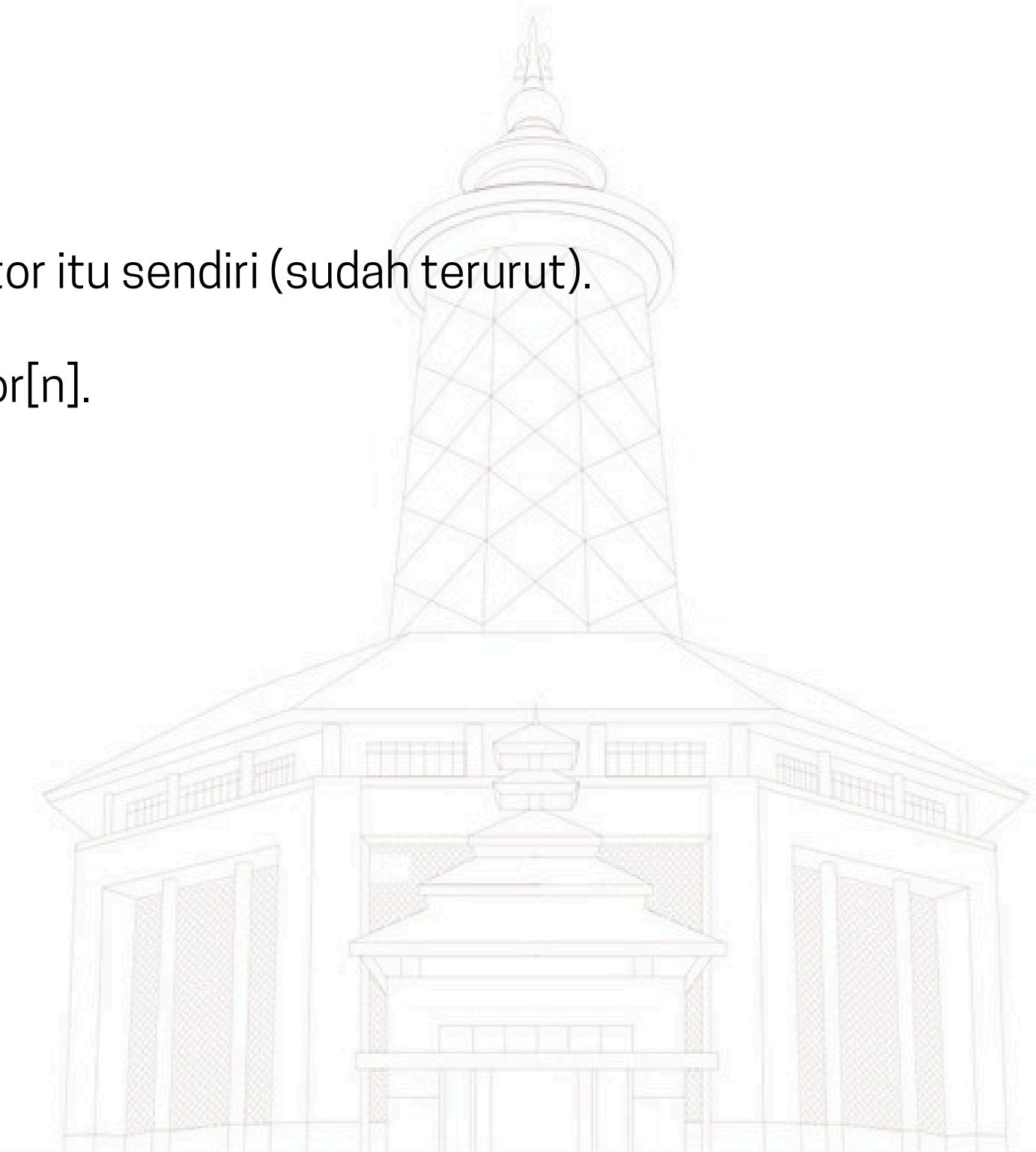


Quick Sort

Langkah Penyelesaian Algoritma (Indeks 1)

1. Fungsi Utama (quick_sort):

- Menerima sebuah vektor.
- Base Case: Jika panjang vektor (n) adalah 0 atau 1, kembalikan vektor itu sendiri (sudah terurut).
- Divide (Partition):
 - Pilih Pivot: Kita pilih elemen terakhir sebagai pivot. $\text{pivot} = \text{vektor}[n]$.
 - Buat 2 vektor kosong: kiri dan kanan.
 - Iterasi i dari 1 hingga $n-1$ (semua elemen selain pivot):
 - Jika $\text{vektor}[i] < \text{pivot}$, masukkan ke kiri.
 - Jika $\text{vektor}[i] \geq \text{pivot}$, masukkan ke kanan.
- Conquer:
 - Panggil quick_sort secara rekursif untuk kedua bagian:
 - $\text{kiri_terurut} = \text{quick_sort}(\text{kiri})$
 - $\text{kanan_terurut} = \text{quick_sort}(\text{kanan})$
- Combine: Gabungkan hasilnya. Pivot diletakkan di tengah.
 - Kembalikan $c(\text{kiri_terurut}, \text{pivot}, \text{kanan_terurut})$



Quick Sort (Min-Max)

Vektor Awal: [7, 5, 1, 9, 3]

Kita akan menelusuri panggilan rekursifnya. Q(...) adalah panggilan fungsi quick_sort.

1. Panggil $Q([7, 5, 1, 9, 3])$

- Pivot: 3 (elemen terakhir).

- Partisi:

- $7 \geq 3 \rightarrow$ kanan = [7]
- $5 \geq 3 \rightarrow$ kanan = [7, 5]
- $1 < 3 \rightarrow$ kiri = [1]
- $9 \geq 3 \rightarrow$ kanan = [7, 5, 9]

- Struktur Rekursif: $c(Q([1]), 3, Q([7, 5, 9]))$

2. Panggil $Q([1])$ (dari kiri)

- Base Case ($n=1$). Kembalikan [1].

3. Panggil $Q([7, 5, 9])$ (dari kanan)

- Pivot: 9.

- Partisi:

- $7 < 9 \rightarrow$ kiri = [7]
- $5 < 9 \rightarrow$ kiri = [7, 5]

- Struktur Rekursif: $c(Q([7, 5]), 9, Q([]))$

-
- 4. Panggil $Q([])$ (dari kanan)
 - Base Case ($n=0$). Kembalikan [].
- 5. Panggil $Q([7, 5])$ (dari kiri)
 - Pivot: 5.
 - Partisi:
 - $7 \geq 5 \rightarrow$ kanan = [7]
 - Struktur Rekursif: $c(Q([]), 5, Q([7]))$
- 6. Panggil $Q([])$ (dari kiri)
 - Base Case ($n=0$). Kembalikan [].
- 7. Panggil $Q([7])$ (dari kanan)
 - Base Case ($n=1$). Kembalikan [7].

Sekarang, proses Combine (menggabungkan dari bawah ke atas):

- Combine Level 5: $c([], 5, [7]) \rightarrow$ menghasilkan [5, 7]
- Combine Level 3: $c([5, 7], 9, []) \rightarrow$ menghasilkan [5, 7, 9]
- Combine Level 1: $c([1], 3, [5, 7, 9]) \rightarrow$ menghasilkan [1, 3, 5, 7, 9]

Hasil Akhir (Terurut): [1, 3, 5, 7, 9]



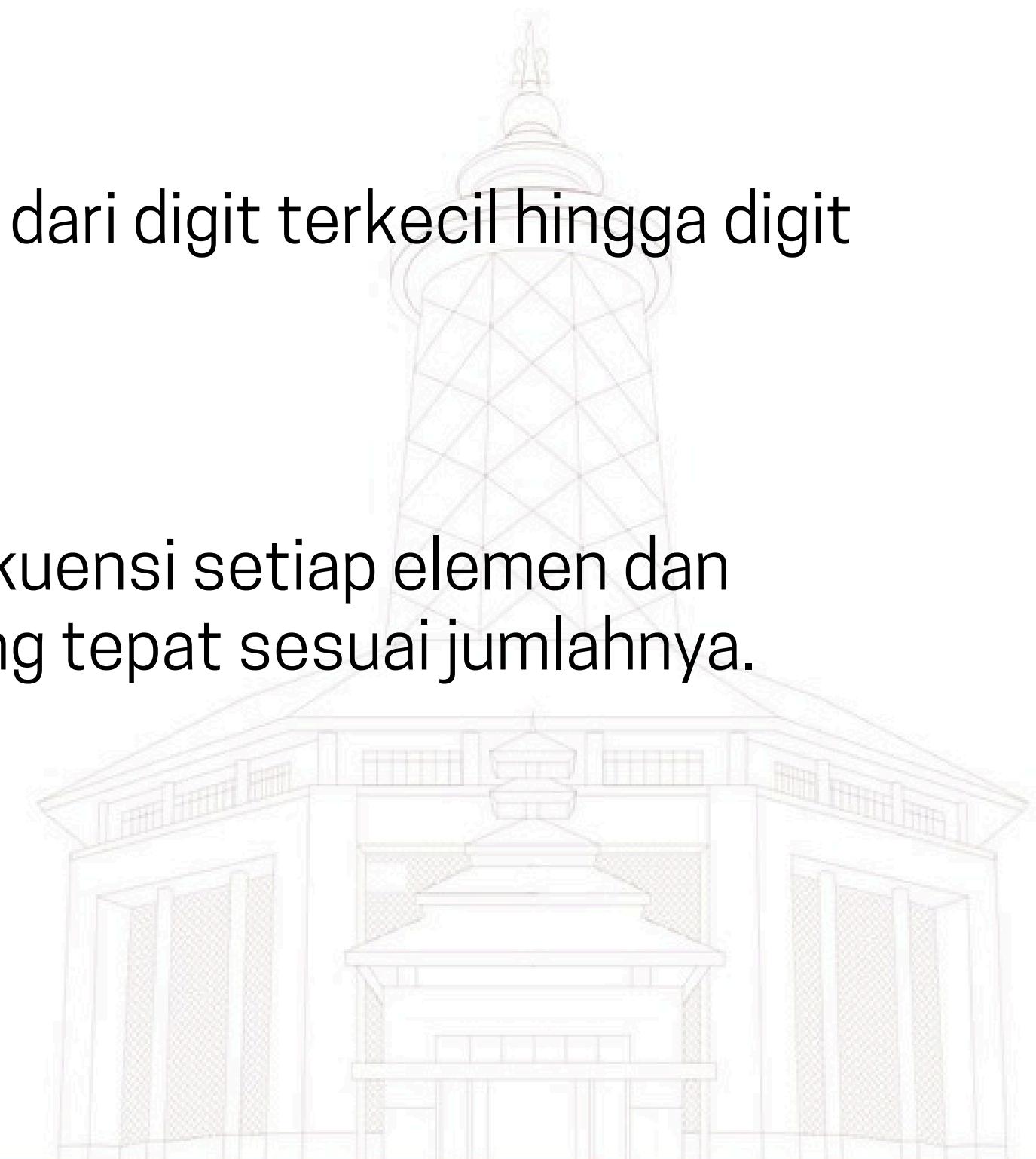
Non-Comparison Sort

1. Radix Sort

Mengurutkan angka berdasarkan digit per digit, mulai dari digit terkecil hingga digit terbesar

2. Counting Sort

Menggunakan array tambahan untuk menghitung frekuensi setiap elemen dan menempatkan elemen-elemen tersebut ke posisi yang tepat sesuai jumlahnya.





Radix Sort

Radix Sort (Sortir Digit) adalah algoritma yang mengurutkan data dengan mengelompokkannya berdasarkan digit per digit. Kita mulai dari digit yang paling tidak signifikan (LSD - Least Significant Digit / satuan), lalu ke puluhan, ratusan, dan seterusnya.



Radix Sort

Langkah Penyelesaian Algoritma (Indeks 1)

1. Cari Digit Maksimal: Temukan angka dengan digit terbanyak (misal, 99 atau 100). Ini menentukan berapa kali kita harus mengulang proses.
2. Loop (Digit): Lakukan loop untuk setiap digit (eksponen), mulai dari 1 (satuan), lalu 10 (puluhan), 100 (ratusan), dst.
3. Sortir per Digit: Di dalam setiap loop digit, urutkan seluruh array menggunakan Counting Sort (atau algoritma stabil lainnya), tetapi berdasarkan nilai digit saat itu.
 - (Misal, saat di digit satuan, 17 dan 2 akan dikelompokkan berdasarkan 7 dan 2. Saat di digit puluhan, 17 dan 02 akan dikelompokkan berdasarkan 1 dan 0).
4. Setelah loop untuk digit terbesar selesai, array akan terurut sempurna.



Radix Sort (Min-Max)

Vektor Awal: [170, 45, 75, 90, 802, 24, 2, 66]

Iterasi 1: Sortir berdasarkan Digit Satuan (eksponen 1)

- Kita lihat digit terakhirnya: [...0, ...5, ...5, ...0, ...2, ...4, ...2, ...6]
- Kita lakukan Counting Sort berdasarkan digit-digit ini:
 - 0 (dari 170, 90)
 - 2 (dari 802, 2)
 - 4 (dari 24)
 - 5 (dari 45, 75)
 - 6 (dari 66)

Hasil Iterasi 1: [170, 90, 802, 2, 24, 45, 75, 66](Penting: 802 tetap di depan 2 karena 170 di depan 90 - ini disebut stabil)

Iterasi 2: Sortir berdasarkan Digit Puluhan (eksponen 10)

- Kita lihat digit tengahnya: [...7..., ...9..., ...0..., ...0..., ...2..., ...4..., ...7..., ...6...]
- Kita lakukan Counting Sort pada array hasil iterasi 1 berdasarkan digit puluhan:
 - 0 (dari 802, 02)
 - 2 (dari 24)
 - 4 (dari 45)
 - 6 (dari 66)
 - 7 (dari 170, 75)
 - 9 (dari 90)

Hasil Iterasi 2: [802, 2, 24, 45, 66, 170, 75, 90](Lihat, 802 dan 2 (002) sekarang di depan. 170 di depan 75 karena urutan di iterasi 1)



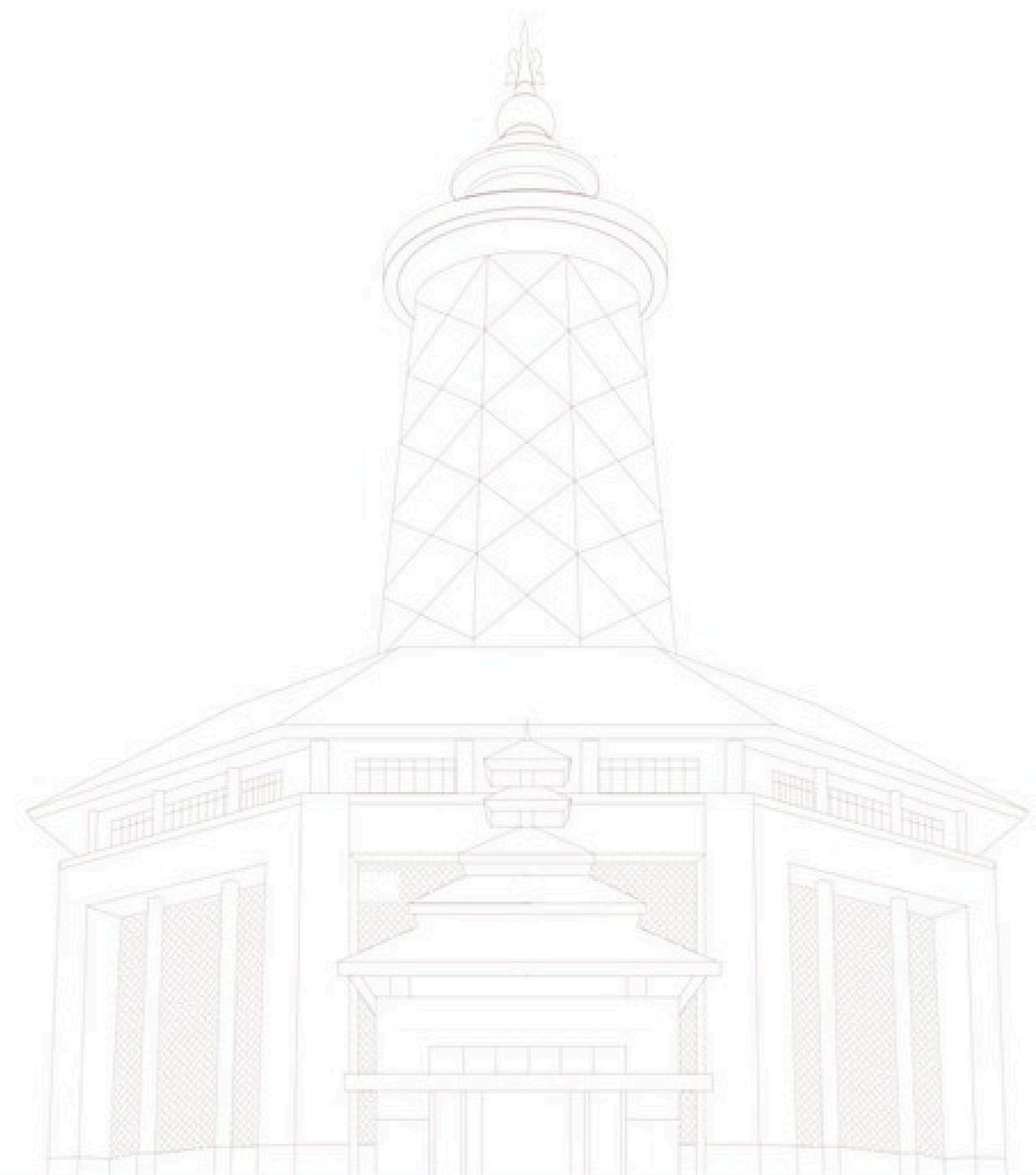
Radix Sort (Min-Max)

Iterasi 3: Sortir berdasarkan Digit Ratusan (eksponen 100)

- Kita lihat digit pertamanya: [...8..., ...0..., ...0..., ...0..., ...0...,
...1..., ...0..., ...0...]
- Kita lakukan Counting Sort pada array hasil iterasi 2
berdasarkan digit ratusan:
 - 0 (dari 002, 024, 045, 066, 075, 090)
 - 1 (dari 170)
 - 8 (dari 802)

Hasil Iterasi 3: [2, 24, 45, 66, 75, 90, 170, 802]

Hasil Akhir (Terurut): [2, 24, 45, 66, 75, 90, 170, 802]





Counting Sort

Counting Sort (Sortir Hitung) adalah algoritma yang sangat efisien, tetapi hanya bisa digunakan jika kita tahu rentang nilai (range) dari data kita (misal, semua angka antara 0-9, atau 1-100).

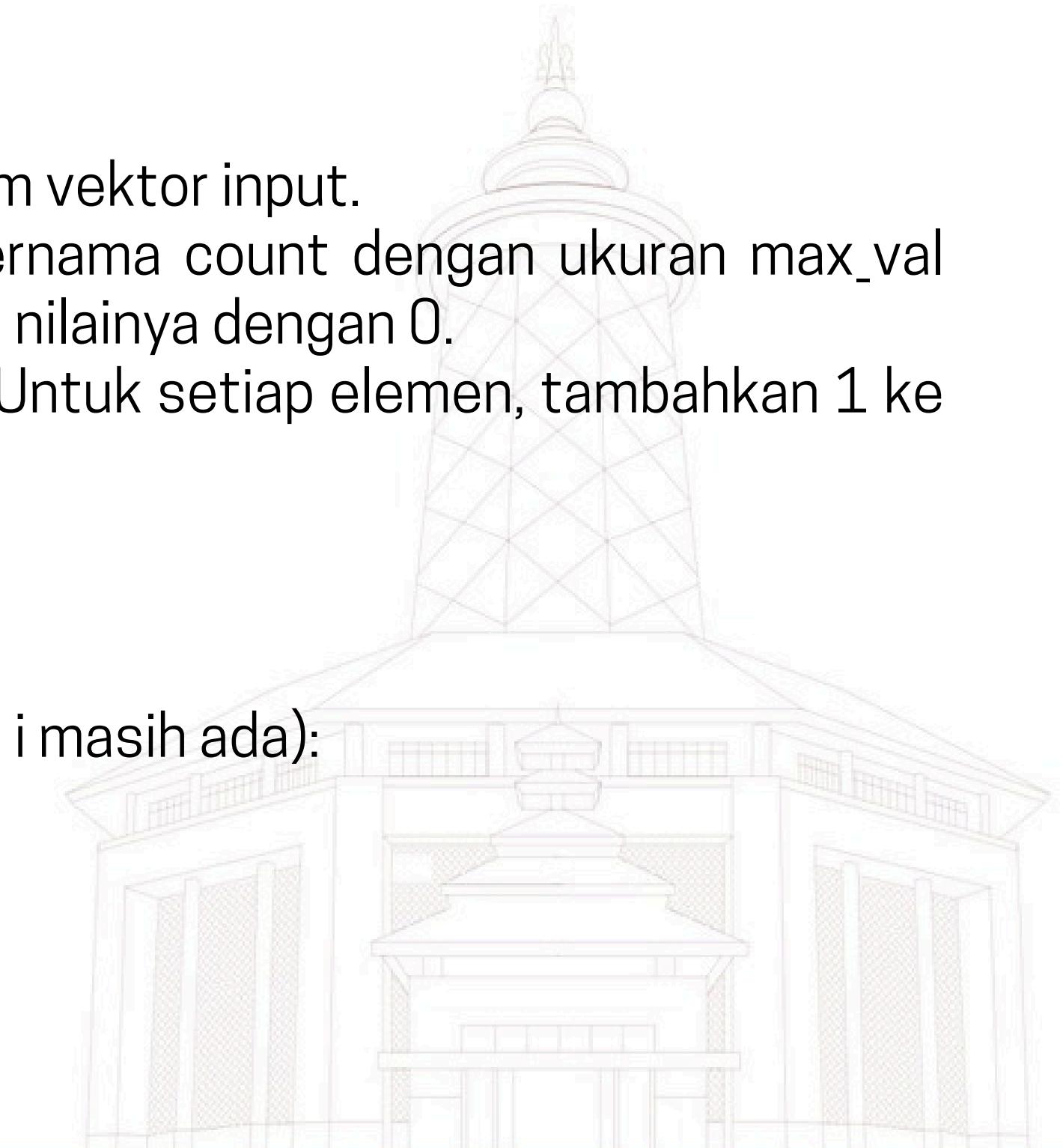
Idenya adalah membuat "ember" (array baru) untuk menghitung frekuensi (jumlah kemunculan) dari setiap elemen. Kemudian, kita membangun kembali array hasil berdasarkan hitungan tersebut.



Counting Sort

Langkah Penyelesaian Algoritma (Indeks 1)

1. Cari Nilai Maksimal: Temukan nilai terbesar (max_val) dalam vektor input.
2. Buat Array Hitung (Count): Buat sebuah vektor baru bernama count dengan ukuran max_val (atau max_val + 1 jika data mulai dari 0). Inisialisasi semua nilainya dengan 0.
3. Hitung Frekuensi: Iterasi melalui vektor input (vektor[i]). Untuk setiap elemen, tambahkan 1 ke "ember"-nya: $\text{count}[\text{vektor}[i]] = \text{count}[\text{vektor}[i]] + 1$
4. Susun Ulang Array (Combine):
 - Buat vektor hasil yang kosong.
 - Iterasi melalui array count, dari $i = 1$ hingga max_val:
 - while ($\text{count}[i] > 0$) (Selama hitungan untuk angka i masih ada):
 - Masukkan i ke dalam hasil.
 - $\text{count}[i] = \text{count}[i] - 1$ (kurangi hitungannya).
 - Kembalikan hasil.

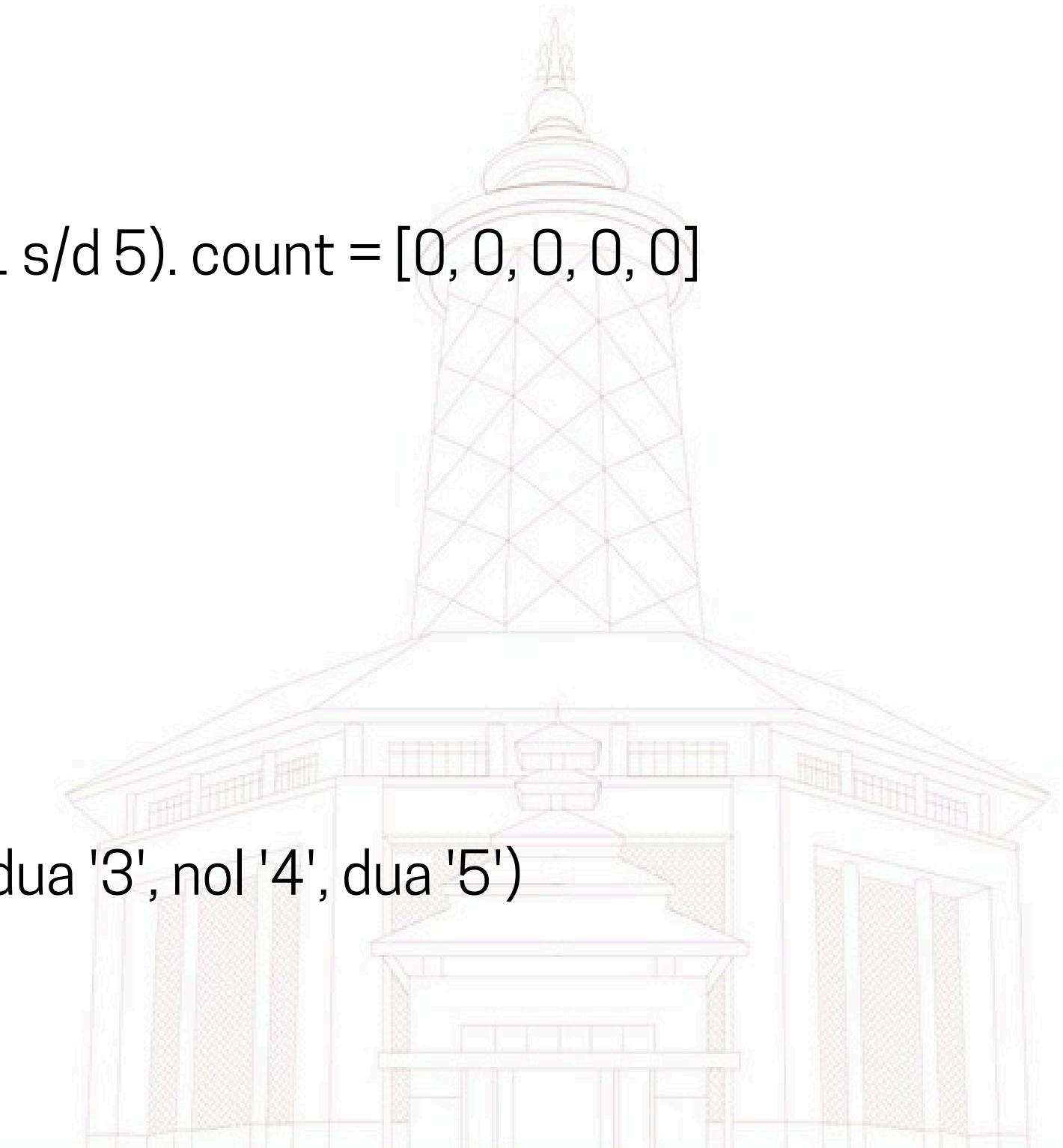




Counting Sort (Min-Max)

Vektor Awal: [5, 3, 1, 5, 3, 1, 1] ($n = 7$)

1. Cari Nilai Maksimal: $\text{max_val} = 5$.
2. Buat Array Hitung: Buat count dengan 5 "ember" (indeks 1 s/d 5). $\text{count} = [0, 0, 0, 0, 0]$
3. Hitung Frekuensi: Iterasi melalui [5, 3, 1, 5, 3, 1, 1].
 - vektor[1] adalah 5 → $\text{count}[5]$ jadi 1.
 - vektor[2] adalah 3 → $\text{count}[3]$ jadi 1.
 - vektor[3] adalah 1 → $\text{count}[1]$ jadi 1.
 - vektor[4] adalah 5 → $\text{count}[5]$ jadi 2.
 - vektor[5] adalah 3 → $\text{count}[3]$ jadi 2.
 - vektor[6] adalah 1 → $\text{count}[1]$ jadi 2.
 - vektor[7] adalah 1 → $\text{count}[1]$ jadi 3.
 - Hasil count: [3, 0, 2, 0, 2] (Artinya: ada tiga '1', nol '2', dua '3', nol '4', dua '5')





Counting Sort (Min-Max)



4. Susun Ulang Array: Buat hasil = [].

- i = 1: count[1] adalah 3.
 - Masukkan '1'. hasil = [1]. count[1] jadi 2.
 - Masukkan '1'. hasil = [1, 1]. count[1] jadi 1.
 - Masukkan '1'. hasil = [1, 1, 1]. count[1] jadi 0.
- i = 2: count[2] adalah 0. Lewati.
- i = 3: count[3] adalah 2.
 - Masukkan '3'. hasil = [1, 1, 1, 3]. count[3] jadi 1.
 - Masukkan '3'. hasil = [1, 1, 1, 3, 3]. count[3] jadi 0.
- i = 4: count[4] adalah 0. Lewati.
- i = 5: count[5] adalah 2.
 - Masukkan '5'. hasil = [1, 1, 1, 3, 3, 5]. count[5] jadi 1.
 - Masukkan '5'. hasil = [1, 1, 1, 3, 3, 5, 5]. count[5] jadi 0.

Hasil Akhir (Terurut): [1, 1, 1, 3, 3, 5, 5]



SEE YOU NEXT WEEK !

Ferdian Bangkit Wijaya, S.Stat., M.Si

NIP. 199005202024061001

ferdian.bangkit@untirta.ac.id

