

---

## EXERCISE 6.1: STRATEGI DIVIDE-AND-CONQUER DAN TEOREMA MASTER

dikerjakan di rumah, sebelum perkuliahan pertemuan 6

---

### Aturan pengerjaan tugas:

1. Buatlah kelompok diskusi beranggotakan 3 orang.
2. Kerjakan soal yang ada secara singkat, padat, dan jelas. Anda disarankan mengerjakan soal secara terurut, karena setiap soal terhubung satu sama lain dengan level pemahaman materi yang naik.
3. Tugas boleh diketik/ditulis tangan (pastikan bisa dibaca), boleh menggunakan Bahasa Indonesia/Inggris. Hindari menggunakan tinta merah. Pastikan bisa dibaca dengan baik.
4. Tugas dikumpulkan dalam bentuk *hardcopy*.
5. Setiap anggota kelompok **wajib** memahami hasil diskusi dan solusi yang dituliskan oleh kelompoknya.

\_\_\_\_\_  
*Dengan ini, Anda menyatakan bahwa Anda siap menerima segala konsekuensi  
jika nantinya ditemukan adanya kecurangan dalam pengerjaan tugas ini.*  
\_\_\_\_\_

# 1 Merge sort dengan DnC

## 1. (Algoritma merge sort dengan DnC)

Cermati dan jelaskan bagaimana proses yang terjadi pada algoritma merge sort berikut. Untuk memudahkan Anda, lihatlah contoh pada slide 6.1, halaman 36 dan 37.

---

### Algorithm 1 Merge Sort

```

1: procedure MERGESORT( $A$ : ordoable array,  $i, j$ : integer)
2:   if  $i = j$  then
3:     return  $A[i]$ 
4:   end if
5:    $k \leftarrow (i + j) \text{ div } 2$ 
6:   MERGESORT( $A, i, k$ )
7:   MERGESORT( $A, k + 1, j$ )
8:   MERGE( $A, i, k, j$ )
9: end procedure

```

▷  $i$ : starting index,  $j$ : last index, initialization:  $i = 0, j = n - 1$  (i.e. the whole array  $A$ )  
 ▷  $\text{length}(A) = 1$   
 ▷ Divide the array into two  
 ▷ Sort the sub-array  $A[i..k]$   
 ▷ Sort the sub-array  $A[k + 1..j]$   
 ▷ Merge sorted  $A[i..k]$  and  $A[k + 1..j]$  into the sorted  $A[i..j]$

---

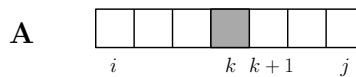


Figure 1: Ilustrasi *indexing* pada slicing array

Prosedur MERGE yang digunakan di Algoritma 1 tersebut adalah sebagai berikut.

---

### Algorithm 2 Prosedur “Merge” in MERGESORT

```

1: function MERGE( $A, i, k, j$ )
2:   declaration
3:      $B$  : temporary array to store the merged values
4:   end declaration
5:    $p \leftarrow i; q \leftarrow k + 1; r \leftarrow i$ 
6:   while  $p \leq k$  and  $q \leq j$  do
7:     if  $A[p] \leq A[q]$  then
8:        $B[r] \leftarrow A[p]$ 
9:        $p \leftarrow p + 1$ 
10:    else
11:       $B[r] \leftarrow A[q]$ 
12:       $q \leftarrow q + 1$ 
13:    end if
14:     $r \leftarrow r + 1$ 
15:  end while
16:  while  $p \leq k$  do
17:     $B[r] \leftarrow A[p]$ 
18:     $p \leftarrow p + 1$ 
19:     $r \leftarrow r + 1$ 
20:  end while
21:  while  $q \leq j$  do
22:     $B[r] \leftarrow A[q]$ 
23:     $q \leftarrow q + 1$ 
24:     $r \leftarrow r + 1$ 
25:  end while
26:  for  $r \leftarrow i$  to  $j$  do
27:     $A[r] \leftarrow B[r]$ 
28:  end for
29:  return  $A$ 
30: end function

```

▷  $A[i..k]$  and  $A[k + 1..j]$  are sorted (ascending)  
 ▷ **output:** Array  $A[i..j]$  sorted (ascending)  
 ▷ while the left-array and the right-array are not finished  
 ▷  $B$  is a temporary array to store the merged array; assign  $A[p]$  (of left array) to  $B$   
 ▷ Assign  $A[q]$  (of right array) to  $B$   
 ▷ At this point,  $p > k$  or  $q > j$   
 ▷ If the left-array is not finished, copy the rest of left-array  $A$  to  $B$  (if any)  
 ▷ If the right-array is not finished, copy the rest of right-array  $A$  to  $B$  (if any)  
 ▷ Assign back all elements of  $B$  to  $A$   
 ▷  $A$  is in ascending order

---

## 2. Kompleksitas waktu algoritma merge sort dengan DnC

Sekarang, hitunglah kompleksitas waktu algoritma tersebut, dengan cara yang serupa dengan metode penghitungan kompleksitas waktu algoritma rekursif.

- Operasi apakah yang harus kita hitung pada algoritma ini? Lambangkan dengan  $T(n)$ : banyaknya operasi yang terjadi ketika input berukuran  $n$ .
- Pertama-tama tentukan fungsi rekursif dari  $T(n)$
- Hitunglah fungsi eksplisit dari  $T(n)$  dengan metode substitusi berulang. Untuk memudahkan kalkulasi, asumsikan bahwa  $n = 2^k$  untuk suatu bilangan bulat positif  $k$ .
- Nyatakan kompleksitas waktunya dalam notasi  $\mathcal{O}$ .
- Bandingkan kompleksitas waktu algoritma ini dengan kompleksitas waktu Merge Sort versi brute-force.

## 2 Teorema Master

### 1. Definisi Teorema Master

Mungkin Anda menyadari bahwa proses penghitungan kompleksitas waktu dari algoritma rekursif maupun algoritma DnC cukup kompleks. Oleh sebab itu, sekarang diperkenalkan metode penghitungan fungsi kompleksitas waktu yang disebut dengan **Teorema Master**.

**Teorema 2.1** (Teorema Master). *Misalkan fungsi kompleksitas waktu dalam bentuk fungsi rekursifnya adalah:*

$$T(n) = aT(n/b) + f(n)$$

dimana  $a, b \in \mathbb{Z}^+$ .

Jika  $f(n) \in \mathcal{O}(n^d)$  dimana  $d \geq 0$ , maka:

$$T(n) \in \begin{cases} \mathcal{O}(n^d), & \text{jika } a < b^d \\ \mathcal{O}(n^d \log n), & \text{jika } a = b^d \\ \mathcal{O}(n^{\log_b a}), & \text{jika } a > b^d \end{cases}$$

**Catatan:** Hasil yang serupa juga berlaku untuk notasi  $\Omega$  dan  $\Theta$ .

### 2. Perhatikan beberapa contoh berikut untuk membantu Anda memahami Teorema Master di atas.

**Contoh 2.2.** Misalkan  $T(n) = 2T(\frac{n}{4}) + \sqrt{n} + 42$ . Tentukan parameter  $a$ ,  $b$ , dan  $d$  seperti pada teorema.

**Jawab:**

Nilai  $a = 2$ ;  $b = 4$ ;  $d = \frac{1}{2}$

Karena  $2 = 4^{\frac{1}{2}}$ , kasus kedua Teorema Master berlaku. Karenanya,

$$T(n) \in \mathcal{O}(n^d \log n) = \mathcal{O}(\sqrt{n} \log n)$$

### 3. Penerapan Teorema Master pada Algoritma Merge Sort sebelumnya

Coba Anda terapkan Teorema Master untuk menghitung fungsi kompleksitas waktu dari algoritma Merge Sort pada contoh sebelumnya (nomor 1). Apakah hasil yang Anda dapatkan sama dengan hasil yang penghitungan pada kalkulasi sebelumnya?