EXERCISE 7.1: STRATEGI DIVIDE-AND-CONQUER DAN TEOREMA MASTER

dikerjakan di rumah, sebelum perkuliahan pertemuan 7

Petunjuk: Kerjakan secara berkelompok dengan anggota 3 orang.

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: ordorable array, i, j: integer)   ▷ i: starting index, j: last index, initialization: i = 0, j = n − 1 (i.e. the whole array A)
2:     if i = j then                                        ▷ length(A) = 1
3:         return A[i]
4:     end if
5:     k ← (i + j) div 2                                    ▷ Divide the array into two
6:     MERGESORT(A, i, k)                                   ▷ Sort the sub-array A[i..k]
7:     MERGESORT(A, k + 1, j)                               ▷ Sort the sub-array A[k + 1..j]
8:     MERGE(A, i, k, j)                                    ▷ Merge sorted A[i..k] and A[k + 1..j] into the sorted A[i..j]
9: end procedure
```

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))          ▷ A[i..k] and A[k+1..j] are sorted (ascending)   ▷ output: Array A[i..j] sorted (ascending)
2:      declaration
3:          B : temporary array to store the merged values
4:      end declaration
5:      p ← i; q ← k + 1; r ← i
6:      while p ≤ k and q ≤ j do                                   ▷ while the left-array and the right-array are not finished
7:          if A[p] ≤ A[q] then
8:              B[r] ← A[p]                                        ▷ B is a temporary array to store the merged array; assign A[p] (of left array) to B
9:              p ← p + 1
10:         else
11:             B[r] ← A[q]                                        ▷ Assign A[q] (of right array) to B
12:             q ← q + 1
13:         end if
14:         r ← r + 1
15:     end while                                                  ▷ At this point, p > k or q > j
16:     while p ≤ k do                                             ▷ If the left-array is not finished, copy the rest of left-array A to B (if any)
17:         B[r] ← A[p]
18:         p ← p + 1
19:         r ← r + 1
20:     end while
21:     while q ≤ j do                                             ▷ If the right-array is not finished, copy the rest of right-array A to B (if any)
22:         B[r] ← A[q]
23:         q ← q + 1
24:         r ← r + 1
25:     end while
26:     for r ← i to j do                                          ▷ Assign back all elements of B to A
27:         A[r] ← B[r]
28:     end for
29:     return A                                                   ▷ A is in ascending order
30: end function
```

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.

   (a) Operasi apakah yang harus kita hitung pada algoritma ini? Lambangkan dengan $T(n)$: banyaknya operasi yang terjadi ketika input berukuran n.

   (b) Pertama-tama tentukan fungsi rekursif dari $T(n)$

   (c) Hitunglah fungsi eksplisit dari $T(n)$ dengan metode substitusi berulang. Untuk memudahkan kalkulasi, asumsikan bahwa $n = 2^k$ untuk suatu bilangan bulat positif k.

   (d) Nyatakan kompleksitas waktunya dalam notasi $\mathcal{O}$.

   (e) 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:*

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

Catatan: Hasil yang serupa juga berlaku untuk notasi Ω dan Θ.

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 sebelummnya (nomor 1). Apakah hasil yang Anda dapatkan sama dengan hasil yang penghitungan pada kalkulasi sebelumnya?