04-05 - Algoritma Rekursif

[KOMS124404]

Desain dan Analisis Algoritma (2024/2025)

Dewi Sintiari

Prodi S1 Ilmu Komputer
Universitas Pendidikan Ganesha

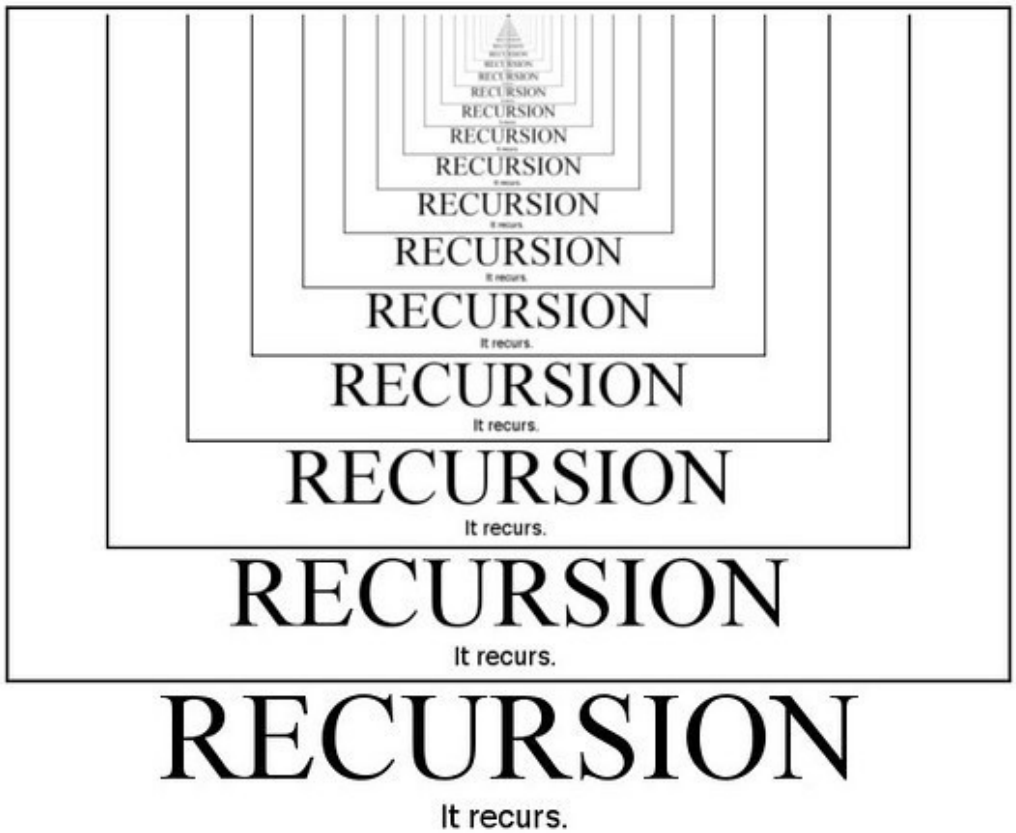
Week 4 (Maret 2025)

Daftar isi

Tujuan pembelajaran

Anda diharapkan mampu untuk:

1. Menjelaskan prinsip algoritma rekursif
2. Menerapkan algoritma rekursif untuk penyelesaian masalah algoritmik
3. Menganalisis kelebihan dan kekurangan algoritma rekursif
4. Memutuskan kapan menerapkan algoritma rekursif untuk menyelesaikan masalah algoritmik



Apa itu rekursi atau algoritma rekursif?

Bagian 1. Prinsip algoritma rekursif

Prinsip algoritma rekursif

Algoritma rekursif adalah sebuah algoritma yang 'memanggil' dirinya sendiri dengan nilai input "lebih kecil (atau lebih sederhana)", dimana output (untuk input yang diberikan) diperoleh berdasarkan output dari input yang lebih kecil (atau lebih sederhana) tersebut.

Karakteristik algoritma rekursif:

1. Algoritma memanggil dirinya secara rekursif
2. Algoritma memiliki kasus dasar (*base case*)
3. Algoritma mengubah *state*-nya dan bergerak menuju *base-case*.

Base case adalah kondisi yang memungkinkan algoritma berhenti berulang: kasus dasar biasanya merupakan masalah yang cukup kecil untuk diselesaikan secara langsung.

Perubahan *state* berarti bahwa beberapa data yang digunakan algoritma diubah. Biasanya data yang mewakili masalah kita menjadi lebih kecil.

Perbedaan Rekursi dan Iterasi

Iterasi: Suatu fungsi yang mengulangi proses yang ditentukan sampai terdapat *stopping condition*. Ini biasanya dilakukan melalui perulangan, seperti perulangan *for* atau *while* dengan *counter* dan pernyataan komparatif yang membentuk kondisi yang akan gagal. Perulangan iterasi secara *infinite* (tak terbatas) terjadi ketika *stopping condition* tidak pernah terpenuhi.

Rekursi: Alih-alih menjalankan proses tertentu di dalam fungsi, fungsi tersebut memanggil dirinya berulang kali hingga kondisi tertentu terpenuhi (kondisi ini menjadi kasus dasar). Kasus dasar secara eksplisit dinyatakan untuk mengembalikan nilai tertentu ketika kondisi tertentu terpenuhi. Loop rekursif yang berulang secara *infinite* terjadi ketika fungsi tidak mengurangi inputnya, sehingga tidak menuju kasus dasarnya (*base case*).

Bagian 2. Contoh sederhana algoritma rekursif

2.1. Komputasi faktorial

2.1 - Komputasi faktorial (1): Pernyataan masalah

Diberikan formula faktorial:

$$n! = n \times (n-1) \times (n-2) \times \cdots \times 2 \times 1$$

Rumusnya dapat dinyatakan secara rekursif:

$$n! = \begin{cases} n \times (n-1)!, & \text{if } n > 1 \\ 1, & n = 1 \end{cases}$$

2.1 - Komputasi faktorial (2): Pseudocode

Algorithm 1 Factorial of a number

```
1: procedure FACTORIAL(n)
2:     if n = 1 then
3:         return 1
4:     else
5:         temp = FACTORIAL(n − 1)
6:         return n ∗ temp
7:     end if
8: end procedure
```

- Apakah *base case*-nya?
- Deskripsikan *change-of-state*-nya!
- Berapakah kompleksitasnya?

2.1 - Komputasi faktorial (2): Pseudocode

Algorithm 2 Factorial of a number

```
1: procedure FACTORIAL(n)
2:     if n = 1 then
3:         return 1
4:     else
5:         temp = FACTORIAL(n − 1)
6:         return n ∗ temp
7:     end if
8: end procedure
```

- Apakah *base case*-nya? $n = 1$
- Deskripsikan *change-of-state*-nya! nilai n menurun
- Berapakah kompleksitasnya? $\mathcal{O}(n)$

2.1 - Komputasi faktorial (3): Diagram

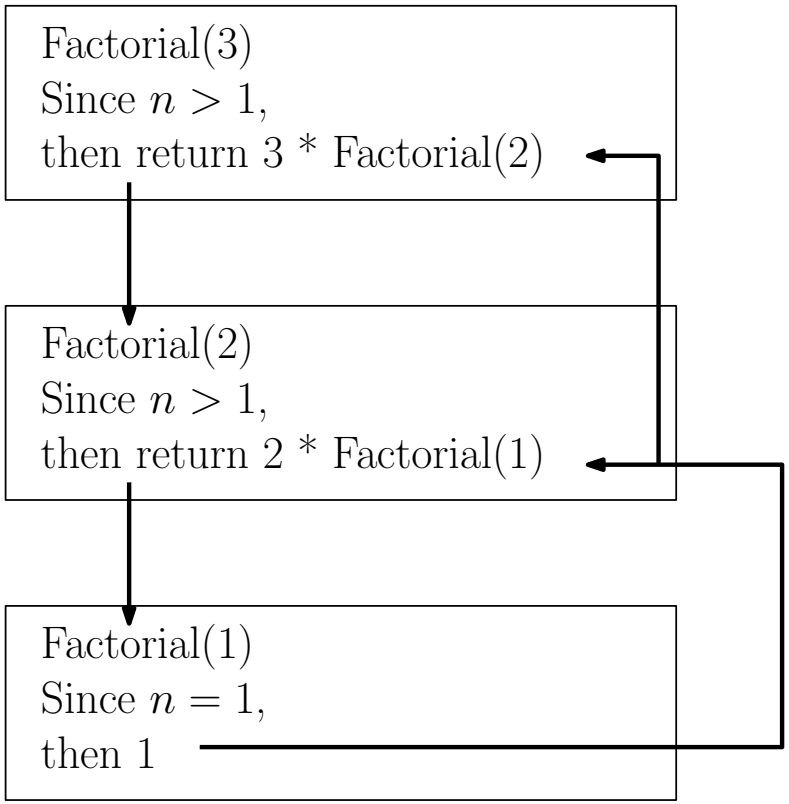


Figure: Ilustrasi algoritma rekursif FACTORIAL dimana $n = 3$

2.1 - Komputasi faktorial (4): Pembuktian kebenaran dengan induksi

- Basis induksi: dari baris ke-1, kita melihat bahwa fungsi bernilai benar untuk $n = 1$.
- Hipotesis: misalkan fungsi bernilai benar untuk suatu input berukuran $n = m$, untuk suatu integer $m \geq 1$.
- Tahap induksi: Kita buktikan bahwa fungsi tersebut bernilai benar ketika dipanggil dengan input $n = m + 1$. Berdasarkan hipotesis, kita tahu bahwa panggilan rekursif bekerja dengan benar untuk $n = m$ dan memberikan hasil $m!$.

  Selanjutnya, jika dikalikan dengan $n = m + 1$, maka menghasilkan $(m + 1)!$. Dan ini adalah nilai yang menjadi output program, dan merupakan nilai yang benar.

2.2. Menemukan elemen maksimum dari array

2.2 - Menemukan elemen maksimum dari array (1)

Untuk menghitung nilai maksimum array dengan n elemen untuk $n > 1$ secara rekursif:

- Hitung maks dari $n - 1$ elemen
- Bandingkan dengan elemen terakhir untuk menemukan nilai maksimum dari keseluruhan array

2.2 - Menemukan elemen maksimum dari array (1)

Untuk menghitung nilai maksimum array dengan n elemen untuk $n > 1$ secara rekursif:

- Hitung maks dari $n - 1$ elemen
- Bandingkan dengan elemen terakhir untuk menemukan nilai maksimum dari keseluruhan array

Algorithm 4 Finding maximum of an array

```
1:  procedure MAX(A[0..n − 1], int n)
2:      if n = 1 then return A[0]
3:      else
4:          T = MAX(A, n − 1)
5:          if T < A[n − 1] then
6:              return A[n − 1]
7:          else
8:              return T
9:          end if
10:     end if
11: end procedure
```

2.2 - Menemukan elemen maksimum dari array (2)

Tugas:

- Hitunglah kompleksitas algoritma di atas!
- Periksalah kebenaran algoritma di atas!

2.3. Menghitung jumlah elemen pada array

2.3 - Menghitung jumlah elemen pada array (1)

Permasalahan: Diberikan sebuah array dari n elemen $A[0..n-1]$. Kita ingin menghitung nilai dari: $S = \sum_{i=0}^{n-1} A[i]$

Algorithm 5 Sum of an array

```
1:  procedure SUM(A[0..n − 1], int n)
2:      if n = 1 then return A[0]
3:      else
4:          S = SUM(A, n − 1)
5:          S = S + A[n − 1]
6:          if T < A[n − 1] then
7:              return S
8:          end if
9:      end if
10: end procedure
```

2.3 - Menghitung jumlah elemen pada array (2)

Tugas:

- Hitunglah kompleksitas algoritma di atas!
- Periksalah kebenaran algoritma di atas!

2.4. Recursive MAX

2.4. Recursive MAX, metode kedua (1)

Permasalahan: Diberikan array A dari n elemen, kita bertujuan untuk menemukan elemen dengan nilai maksimum array.

metode:

1. Bagilah array menjadi dua bagian sub-array, yaitu sub-array Left dan sub-array Right.
2. Temukan maks dari setiap sub-array.
3. Bandingkan nilai maksimum array kiri dan array kanan.
4. Mengembalikan maksimum dari dua nilai.

2.4. Recursive MAX, metode kedua (2)

Algorithm 6 Finding max of an array

```
1:  procedure FindMax(A[i..j], n)        ▷ i, j are respectively the index of start, end of A
2:      if n = 1 then  return A[0]
3:      end if
4:      m = ⌊(i+j)/2⌋
5:      T1 = FindMax(A[i..m], ⌊n/2⌋)     ▷ Recursive call the left sub-array
6:      T2 = FindMax(A[(m+1)..j], n − ⌊n/2⌋)   ▷ Rec. call right sub-array
7:      if T1 ≥ T2 then return T1        ▷ Compare the two max elements
8:      else return T2
9:      end if
10: end procedure
```

Catatan. Fungsi *floor* $\lfloor x \rfloor$ berarti bilangan bulat terbesar yang $\leq x$; contoh: $\lfloor 3.5 \rfloor = 3$

2.4. Recursive MAX, metode kedua (3)

Analisis kompleksitas: Kasus khusus ketika $n = 2^k$

Misalkan $f(n)$: jumlah perbandingan kunci untuk menemukan maks dari n-array, dengan $n = 2^k$ untuk beberapa bilangan bulat positif k. Sehingga:

$$f(n) = \begin{cases} 0, & n = 1 \\ 1 + 2f(n/2), & n \geq 2 \end{cases}$$

Dengan substitusi berulang:

$$\begin{aligned} f(n) &= 1 + 2f(n/2) \\ &= 1 + 2[1 + 2f(n/4)] = 1 + 2 + 2f(n/4) \\ &= 1 + 2 + 4 + 8f(n/4) \\ &\vdots \\ &= 1 + 2 + 4 + \cdots + 2^{k-1} + 2^k f(n/2^k) \\ &= 1 + 2 + 4 + \cdots + 2^{k-1} \\ &= 2^k - 1/(2-1) = 2^k - 1 \\ &= n - 1 \end{aligned}$$

2.4. Recursive MAX, metode kedua (4)

Misal $f(n)$: banyaknya perbandingan kunci untuk menemukan maksimum n-array, dengan $n = 2^k$ untuk beberapa $k \in \mathbb{Z}^+$.

Analisis kompleksitas: Untuk bilangan bulat n

$$f(n) = \begin{cases} 0, & n = 1 \\ f(\lfloor \frac{n}{2} \rfloor) + f(n - \lfloor \frac{n}{2} \rfloor) + 1, & n \geq 2 \end{cases}$$

Buktikan bahwa:

Dengan induksi, diperoleh $f(n) = n - 1$. Coba Anda jelaskan bagaimana hasil ini diperoleh dengan menggunakan induksi?

Contoh lanjut: Recursive powering

Recursive powering (1): Deskripsi masalah

Permasalahan: Diberikan X dan bilangan bulat n. Kita ingin menghitung X^n.

Algorithm 7 Recursive powering (*brute force*)

```
1: procedure POWER1(X, n)
2:     T = X
3:     for i = 2 to n do
4:         T = T * X
5:     end for
6: end procedure
```

Kompleksitas $\mathcal{O}(n)$. Selidikilah mengapa?

Recursive powering (2): Penyelesaian

Ide: tuliskan $X^{16} = ((((X^2)^2)^2)^2)^2$

Diberikan $n = 2^k$, kita dapat mencari kuadrat-nya secara berulang.

Algorithm 8 Improvement brute force

```
1: procedure POWER2(X, n = 2^k)
2:     T = X
3:     for i = 2 to k do
4:         T = T * T
5:     end for
6: end procedure
```

Kompleksitas: $\mathcal{O}(\log n)$. Dapatkah Anda jelaskan mengapa?

Recursive powering (3): Penyelesaian

Generalisasi untuk sebarang nilai n: Hitunglah X^n untuk $n \in \mathbb{Z}^+$

- Hitung $X^2 = X * X$
- Hitung $X^3 = X^2 * X$
- Hitung $X^6 = X^3 * X^3$
- Hitung $X^{12} = X^6 * X^6$
- Hitung $X^{13} = X^{12} * X$

Recursive powering (4): Penyelesaian

Ide dasar: Bagi n dengan 2, $n = n/2 + n/2$. Jadi

$$X^n = X^{(n/2+n/2)} = X^{n/2} \cdot X^{n/2}$$

Masalahnya adalah $n/2$ tidak selalu bilangan bulat. Jadi kita harus menerapkan sedikit modifikasi:

- Untuk $n = 0$, lalu $X^n = 1$
- Untuk $n > 0$, maka:
  - Jika n *genap*, maka $X^n = X^{n/2} \cdot X^{n/2}$
  - Jika n *ganjil*, maka $X^n = X^{\lfloor n/2 \rfloor} \cdot X^{\lfloor n/2 \rfloor} \cdot X$

Recursive powering (5): Pseudocode

Algorithm 9 Recursive powering

```
1:  procedure POWER3(X, n)
2:      if n = 1 then
3:          return X
4:      end if
5:      T = POWER3(X, ⌊n/2⌋)
6:      T = T * T                          ▷ T = T^⌊n/2⌋ * T^⌈n/2⌉
7:      if n mod 2 = 1 then                ▷ skipped
8:          T = T * X
9:          return T
10:     end if
11: end procedure
```

Berapakah kompleksitas waktunya?

Recursive powering (6): Contoh penerapan

Contoh: Hitung 3^{16}

$$
\begin{aligned}
3^{16} = 3^8 \cdot 3^8 &= (3^8)^2 \\
&= ((3^4)^2)^2 \\
&= (((3^2)^2)^2)^2 \\
&= ((((3^1)^2)^2)^2)^2 \\
&= ((((3^0) \cdot 3)^2)^2)^2)^2 \\
&= (((1 \cdot 3)^2)^2)^2)^2 \\
&= (((3)^2)^2)^2)^2 \\
&= (((9)^2)^2)^2 \\
&= ((81)^2)^2 \\
&= (6561)^2 \\
&= 43,046,721
\end{aligned}
$$

Recursive powering (7): Kebenaran algoritma

Algorithm 14 Power by multiplications

```
1:  procedure POWER3(X, n)
2:      if n = 1 then
3:          return X
4:      end if
5:      T = POWER(X, ⌊n/2⌋)
6:      T = T * T
7:      if n mod 2 = 1 then
8:          T = T * X
9:          return T
10:     end if
11: end procedure
```

Misalkan $n = 2m + r$, dimana $r \in \{0, 1\}$.

- Algoritma melakukan panggilan rekursif untuk menghitung $T = X^m$.
- Kuadratkan T untuk mendapatkan $T = X^{2m}$. Jika $r = 0$, maka *return*.
- Jika tidak, ketika $r = 1$, algoritma mengalikan T dengan X, untuk menghasilkan $T = X^{2m+1}$.

Recursive powering (8): Analisis kompleksitas waktu

Misalkan $f(n)$: jumlah kasus terburuk dari banyaknya perkalian untuk menghitung X^n.

Untuk menghitung $f(n)$, perhatikan bahwa operasi yang dilakukan adalah:

- Panggilan rekursif untuk menghitung $f(\lfloor \frac{n}{2} \rfloor)$.
- Kemudian diikuti dengan satu perkalian lagi. Dalam kasus terburuk, ketika n ganjil, satu perkalian tambahan dibutuhkan.

Jadi, fungsi rekurens-nya adalah:

$$f(n) = \begin{cases} 0, & \text{if } n = 1 \\ f(\lfloor \frac{n}{2} \rfloor) + 2, & \text{if } n \geq 2, n \text{ ganjil} \\ f(\lfloor \frac{n}{2} \rfloor) + 1, & \text{if } n \geq 2, n \text{ genap} \end{cases}$$

Tunjukkan bahwa $f(n) = 2\lfloor \log n \rfloor$ (coba gunakan induksi).

Recursive powering (8): Analisis kompleksitas waktu

$$f(n) = \begin{cases} 0, & \text{if } n = 1 \\ f(\lfloor \frac{n}{2} \rfloor) + 2, & \text{if } n \geq 2,\ n \text{ ganjil} \\ f(\lfloor \frac{n}{2} \rfloor) + 1, & \text{if } n \geq 2,\ n \text{ genap} \end{cases}$$

Dua kasus terakhir memiliki perbedaan kecil. Jadi kita dapat mengaproksimasi fungsi di atas dengan fungsi berikut untuk menyederhanakan perhitungan:

$$f(n) = \begin{cases} 0, & \text{if } n = 1 \\ f(\lfloor \frac{n}{2} \rfloor) + 2, & \text{if } n \geq 2 \end{cases}$$

Recursive powering (9): Pembuktian induktif

Tujuan: untuk menunjukkan bahwa $f(n) = 2\lfloor \log n \rfloor$.

- Basis induksi ($n = 1$): Dari pengulangan diperoleh $f(1) = 0$, dan dari rumus diperoleh $f(1) = 2\lfloor \log 1 \rfloor = 0$. (Jadi basis induksi bernilai benar.)
- Pembuktian induktif: Misalkan rumusnya benar untuk semua nilai yang lebih kecil.

$$f(m) = 2\lfloor \log m \rfloor, \quad \forall m < n$$

Setiap bilangan bulat n dapat dinyatakan sebagai:

$$2^k \leq n < 2^{k+1} \quad \text{untuk suatu bilangan bulat } k$$

Jadi, $\lfloor \log n \rfloor = k$, and $\lfloor \frac{\log n}{2} \rfloor = k - 1$. Dengan fungsi rekursif:

$$f(n) = f(\lfloor \frac{n}{2} \rfloor) + 2 = 2(k-1) + 2 = 2k = 2\lfloor \log n \rfloor$$

Catatan. Metode ini memberikan kompleksitas yang lebih baik dibandingkan *brute-force* ($\mathcal{O}(n)$).

Bagian 5. *Redundancy* pada algoritma rekursif

Contoh 1: Recursive powering (1)

Algorithm 14 Power by multiplications

```
1:  procedure POWER3(X, n)
2:      if n = 1 then
3:          return X
4:      end if
5:      T = POWER(X, ⌊n/2⌋)
6:      T = T * T
7:      if n mod 2 = 1 then
8:          T = T * X
9:          return T
10:     end if
11: end procedure
```

Apakah perlu untuk menyimpan $\textsc{Power}(X, \lfloor \frac{n}{2} \rfloor)$ dalam beberapa variabel T?

Contoh 1: Recursive powering (2)

Misalkan bahwa $n = 2^k$ untuk beberapa nilai k.

Algorithm 10 Recursive powering

```
1: procedure POWER4(X, n)
2:     if n = 1 then
3:         return X
4:     end if
5:     return POWER(X, ⌊n/2⌋) * POWER(X, ⌊n/2⌋)
6: end procedure
```

- Apakah algoritmanya benar?
- Bagaimana kompleksitasnya?

Contoh 1: Recursive powering (3)

Algoritmanya benar.

Jumlah panggilan rekursif:

$$f(n) = \begin{cases} 0, & \text{if } n = 1 \\ f(\lfloor \frac{n}{2} \rfloor) + f(\lfloor \frac{n}{2} \rfloor) + 1, & \text{if } n \geq 2 \end{cases}$$

Dengan induksi, kita dapat membuktikan bahwa $f(n) = n - 1$ (lebih buruk secara asimtotik dari algoritma sebelumnya).

Apa yang dapat Anda simpulkan?

Contoh 1: Recursive powering (3)

Algoritmanya benar.

Jumlah panggilan rekursif:

$$f(n) = \begin{cases} 0, & \text{if } n = 1 \\ f(\lfloor \frac{n}{2} \rfloor) + f(\lfloor \frac{n}{2} \rfloor) + 1, & \text{if } n \geq 2 \end{cases}$$

Dengan induksi, kita dapat membuktikan bahwa $f(n) = n - 1$ (lebih buruk secara asimtotik dari algoritma sebelumnya).

Apa yang dapat Anda simpulkan?
POWER4 juga tidak efisien, karena kita melakukan dua pemanggilan rekursif untuk fungsi yang sama $f(\lfloor \frac{n}{2} \rfloor)$

Bagian 6. Kelebihan & kekurangan algoritma rekursif

Kelebihan & kekurangan algoritma rekursif (1)

Kelebihan

- Rekursi memberikan kejelasan dan mengurangi waktu yang dibutuhkan untuk menulis dan men-debug kode (karena mengurangi panjang kode).
- Bermanfaat pada penyelesaian masalah yang secara alami bersifat rekursif, misalnya Masalah Menara Hanoi.
- Rekursi dapat mengurangi kompleksitas waktu (*terkadang kontra-intuitif*).
- Mengurangi pemanggilan fungsi yang tidak perlu.

Kelebihan dan kekurangan algoritma rekursif (2)

Kekurangan

- Fungsi rekursif umumnya lebih lambat daripada fungsi non-rekursif.
- Mungkin memerlukan banyak ruang memori untuk menyimpan "hasil antara" pada proses rekursi.
- Cenderung sulit untuk menganalisis atau memahami kode.
- Tidak lebih efisien dari segi kompleksitas ruang dan waktu (bisa lambat).
- Komputer mungkin kehabisan memori jika panggilan rekursif tidak diperiksa dengan benar.

Rangkuman...

What have we learned today?

1. Peninjauan kembali algoritma brute force
2. Memahami konsep algoritma rekursif
3. Beberapa contoh algoritma rekursif
4. Persamaan perulangan untuk menganalisis kompleksitas waktu
5. Redundansi dalam rekursi $\rightarrow$ jadi, berhati-hatilah saat menuliskan kode

end of slide...