SORTING (1)

KULIAH ANALISIS ALGORITMA DAN KOMPLEKSITAS

MASALAH SORTING (PENGURUTAN)

- **Algoritma sorting: algoritma yang meletakkan elemen-elemen yang ada pada suatu list dengan urutan tertentu.**
- **Definisi algoritma sorting:**
  - Input: sederet n angka $<a_1, a_2, \ldots, a_n>$
  - Output: permutasi (reordering) $<a_1', a_2', \ldots, a_n'>$ dari deretan input sedemikian hingga $a_1' \leq a_2' \leq \ldots \leq a_n'$.
- **Input biasanya berupa array/larik dengan n elemen, tetapi bisa juga direpresentasikan sebagai linked list.**
- **Syarat output:**
  - Non-decreasing order.
  - Permutasi dari elemen-elemen input.

MENGAPA SORTING?

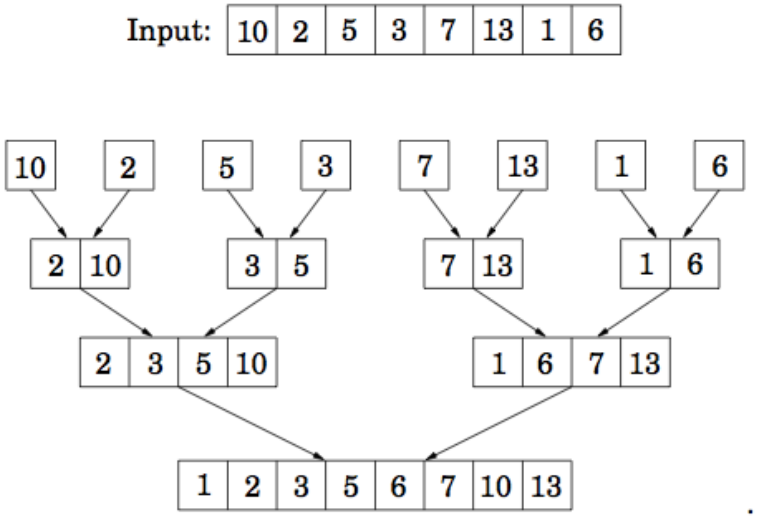
- **Sorting sering dianggap sebagai masalah fundamental dalam studi tentang algoritma.**
  - Aplikasi memerlukan pengurutan informasi.
  - Algoritma sering menggunakan sorting sebagai sub-routine.
  - Terdapat banyak algoritma sorting.
  - Implementasi sorting tergantung pada banyak faktor.

ALGORITMA SORTING

- **Order kuadratik:**
  - Insertion sort
  - Bubble sort
  - Selection sort
- **Order linieritmik**
  - Merge sort
  - Quick sort
  - Heap sort

MERGE SORT

- **Ide:**
  - Bagi n elemen menjadi 2 bagian, masing-masing berisi n/2 elemen.
  - Urutkan masing-masing bagian.
  - Gabungkan 2 bagian sehingga menjadi larik yang terurut.



- **Algoritma:**

  Merge(A, p, q, r)

```
n = q – p + 1 // panjang A[p,..q]
m = r – q      // panjang A[q+1..r]
L[1 .. n+1] dan R[1..m+1] adalah array yang baru.
for i = 1 to n
        L[i] = A[p+i – 1] // copy array A[p..q] ke  L[1 .. n]
for j = 1 to m
        R[j] = A[q + j] // copy array A[q+1..r] ke  R[1 .. m]
L[n+1] = ∞
R[m+1] = ∞
for k = p to r
        if L[i] ≤ R[j]
                A[k] = L[i]
                i = i + 1
        else
                A[k] = R[j]
                j = j + 1
```

- **Algoritma dengan rekursi:**

```
Merge-sort(A, p, r)
 If p < r
        q = ⌊(p+r)/2⌋
        Merge-sort(A, p, q)
        Merge-sort(A, q + 1, r)
        Merge(A, p, q, r)
```

- **Best case, worst case dan average case: O(n log n)**

QUICKSORT

- **Versi dasar ditemukan oleh C. A. R. Hoare pada 1960**
- **Quick sort secara formal diperkenalkan pada 1962.**
- **Kelebihan:**
  - Tidak memakan memory banyak karena hanya menggunakan stack kecil.
  - Memerlukan waktu rata-rata (average running time) $n \lg(n)$ untuk n item.
  - Tidak memerlukan array tambahan.
  - Inner loop yang pendek.
- **Keburukan:**
  - Rekursif
  - Worst case: n^2
  - Fragile → kesalahan sedikit pada implementasi menyebabkan jeleknya performance.

CARA KERJA

- **Berdasarkan pada 3 langkah strategi divide-and conquer**
- **Untuk mengurutkan subarray A[p..r]:**
  - Divide: Larik A[*p* . . *r*] dipartisi menjadi 2 non-empty array *A*[*p* . . *q*] and *A*[*q*+1 . . *r*] sedemikian hingga setiap elemen pada *A*[*p* . . *q*] kurang dari atau sama dengan elemen-elemen pada *A*[*q*+1 . . *r*].
  - Conquer: dua sub-array diurutkan dengan pemanggilan rekursif pada Quick sort.
  - Combine: tidak perlu menggabungkan hasil sebelumnya karena array sudah terurut.
- **Posisi dari partisi bergantung pada elemen terakhir larik yang dipakai sebagai pivot.**
- **Indeks q dihitung sebagai bagian dari prosedur partisi.**

PROSEDUR

QUICKSORT(*A*, *p*, *r*)

1. if *p* < *r*
2. then *q* ←PARTITION(*A*, *p*, *r*)
3. QUICKSORT(*A*, *p*, *q*-1)
4. QUICKSORT(*A*, *q*+1, *r*)

- **Untuk mengurutkan seluruh larik, pemanggilan adalah (A, 1, A.length)**
- **Langkah divide dilakukan dengan prosedur ini, yang akan menghasilkan indeks *q*.**

PROSEDUR UNTUK MEMPARTISI

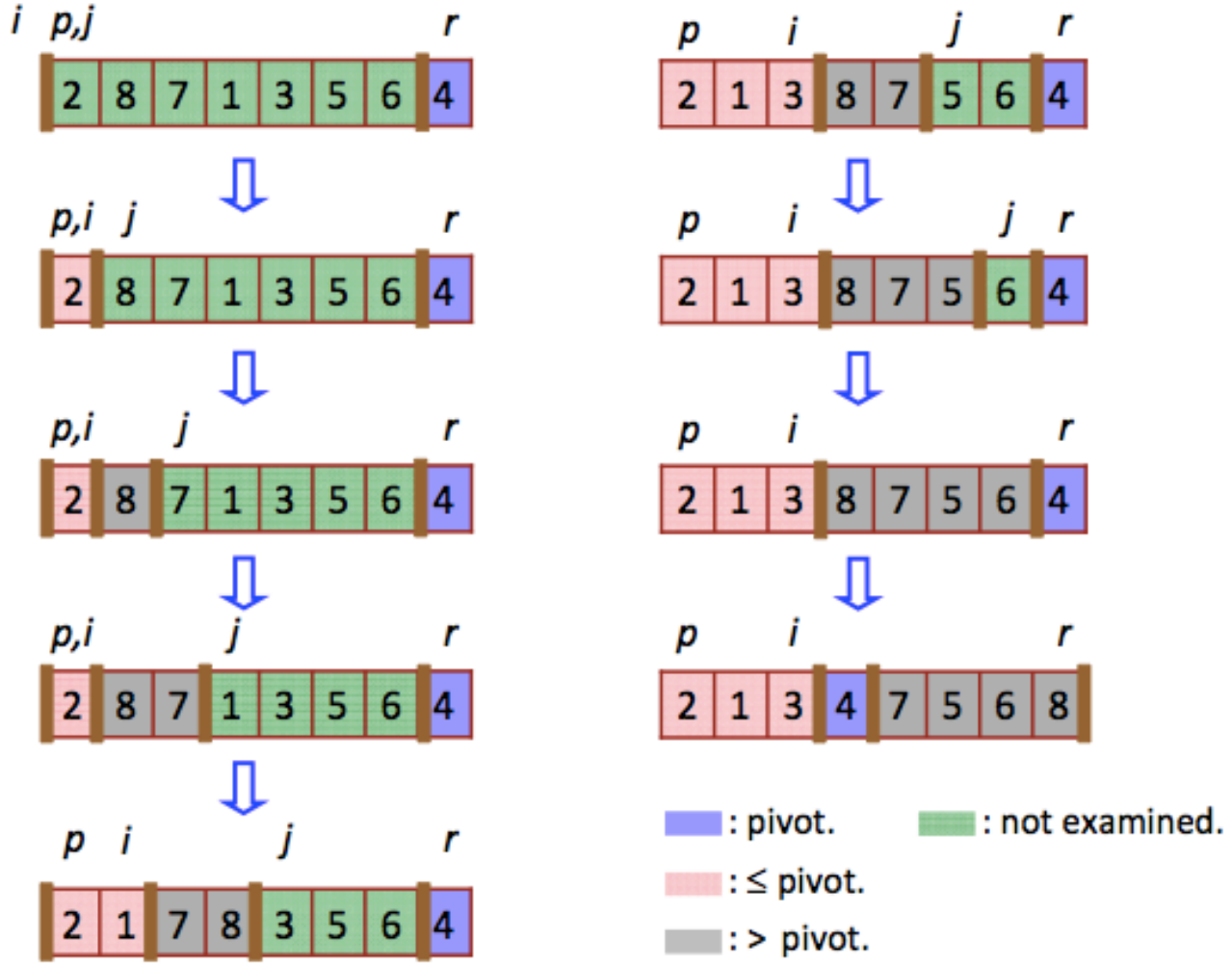
- **Mempartisi subarray A[p..r] menjadi dua A[p..q-1] dan A[q +1..r]**

PARTITION(A, p, r)

1. $x \leftarrow A[r]$ //→ $\Theta(1)$
2. $i \leftarrow p-1$ //→ $\Theta(1)$
3. For $j \leftarrow p$ to $r-1$ //→ $(n-1) \cdot \Theta(1)$
4. do if $A[j] \le x$ //→ $(n-1) \cdot \Theta(1)$
5. then $i \leftarrow i + 1$ //→ $(n-1) \cdot \Theta(1)$
6. exchange $A[i] \leftrightarrow A[j]$ //→ $(n-1) \cdot \Theta(1)$
7. exchange $A[i + 1] \leftrightarrow A[r]$ //→ $\Theta(1)$
8. Return $i+1$ // → $\Theta(1)$

- **PARTITION selalu memilih elemen terakhir *A[r]* di subarray *A[p...r]* sebagai pivot.**
- **Waktu: *O(n)*.**

ILUSTRASI PROSEDUR PARTITION



PERFORMANCE

- **Running time pada quicksort tergantung pada partisi subarray:**
  - Jika sub-array seimbang, quicksort dapat secepat mergesort.
  - Jika tidak seimbang, quicksort berjalan selambat insertion sort.
- **Best case:**
  - Setiap partitioning membagi menjadi 2 bagian yang seimbang.
  - Running time: Θ (n lg n)
- **Worst case:**
  - *A*[1 . . *n*] sudah diurutkan.
  - Running time: Θ (n^2)
- **Average case:** Θ (n lg n)

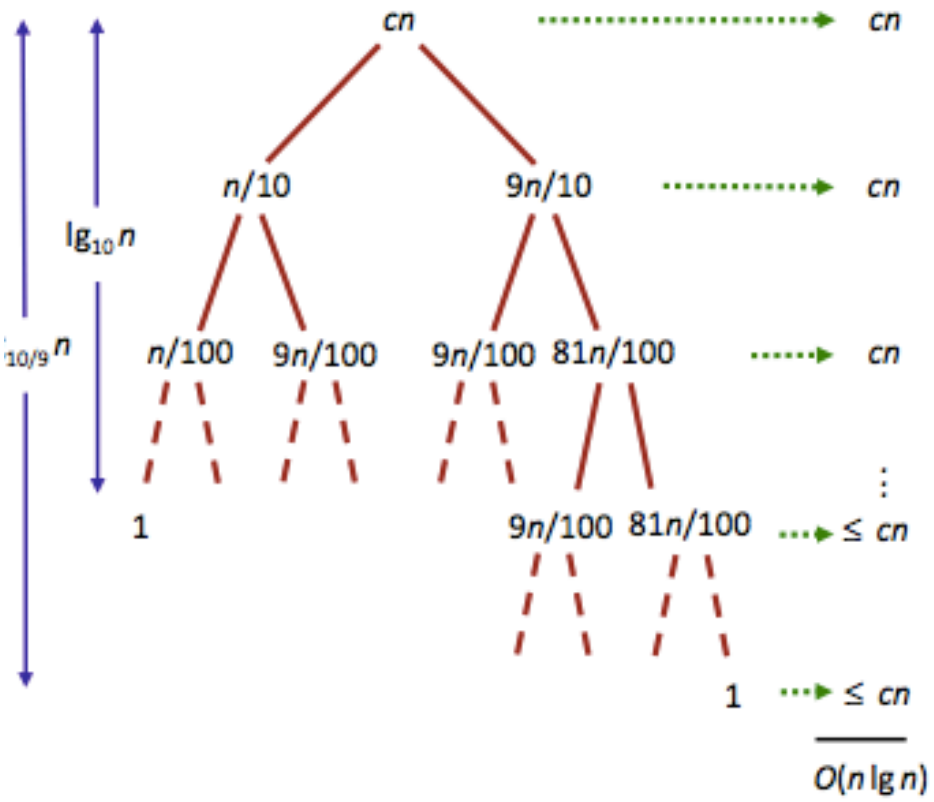
ANALISIS PERFORMANCE QUICKSORT

- **Worst-case partitioning:**
  - Ada 0 elemen di satu sub-array dan n-1 elemen di sub-array yang lain.
  - Rekurensi $T(n) = T(n\text{-}1) + T(0) + \Theta(n)$
    $= T(n\text{-}1) + \Theta(n)$
    $= \Theta(n^2)$.
  - Terjadi saat array input sudah terurut.
- **Best-case partitioning:**
  - Terjadi saat sub-array selalu seimbang.
  - Setiap subarray memiliki $\leq n/2$ elemen.
  - Bentuk rekurensi: $T(n) \leq 2T(n/2) + \Theta(n) = \Theta(n \lg n)$.

PARTISI YANG SEIMBANG

- **Avarage time pada quicksort lebih dekat pada best case daripada pada worst case.**
- **Misalkan bahwa partisi selalu menghasilkan subarray yang terdiri dari 9 elemen dan 1 elemen, maka running time-nya**

  $T(n) \le T(9n/10) + T(n/10) + cn = \Theta(n \lg n)$.



VERSI RANDOMIZED DARI QUICKSORT

- **Pada average-case, kita asumsikan bahwa semua permutasi dari input memiliki probabilitas yang sama untuk muncul → tidak selalu benar**
- **Cara merandomisasi:**
  - Jangan selalu gunakan *A*[*r*] sebagai pivot.
  - Pilih elemen secara random dari array yang sedang diurutkan sebagai pivot.

- **Algoritma randomized untuk quick sort**

  RANDOMIZED-PARTITION(*A*, *p*, *r*)

  1. *i* ←RANDOM(*p*, *r*)
  2. exchange *A*[*r*] ↔ *A*[*i*]
  3. return PARTITION(*A*, *p*, *r*)

- **Dengan memilih elemen pivot secara random, pembagian elemen pada sub-array akan seimbang secara rata-rata.**

RANDOMIZED-QUICKSORT(A, p, r)

1. if $p < r$
2. then $q \leftarrow$ RANDOMIZED-PARTITION(A, p, r)
3. RANDOMIZED-QUICKSORT(A, p, q-1)
4. RANDOMIZED-QUICKSORT(A, q + 1, r)

- **Worst-case dari randomized quick sort: $O(n^2)$.**
- **Expected running time quicksort, dengan menggunakan RANDOMIZED- PARTITION, adalah $O(n \lg n)$.**

LATIHAN

- **Kapankah best case dan worst case terjadi untuk merge sort?**
- **Lakukan quicksort untuk larik: [8, 4, 1, 6, 20, 9, 14, 17]**