SORTING DENGAN WAKTU LINIER

KULIAH ANALISIS ALGORITMA DAN KOMPLEKSITAS

ALGORITMA SORTING DENGAN WAKTU LINIER

- **Algortima-algoritma yang dapat melakukan sorting n bilangan dengan O(n lg n):**
  - Merge sort dan heapsort → untuk worst case
  - Quicksort → untuk average case
  - Untuk setiap algoritma ini, kita dapat menghasilkan sekuens *n bilangan* input yang dapat mengakibatkan algoritma dapat bekerja dalam waktu $\Theta(n \lg n)$.
- **Karakteristik: pengurutan berdasarkan perbandingan antara elemen-elemen input → comparison sorts.**
  - Semua algoritma sorting yang sudah dibahas merupakan comparison sort: insertion sort, selection sort, bubble sort, merge sort, quicksort, heapsort.
- **Algoritma sorting dengan waktu linier: Counting sort, Radix sort, Bucket sort**

→ Ada yang tidak menggunakan operasi perbandingan antara elemen-elemen input

COUNTING SORT: OVERVIEW

- **Asumsi pada counting sort: setiap elemen adalah bilangan bulat pada range {0.. k}, untuk suatu bilangan bulat k.**
- **Jika $k = O(n)$, maka running time dari algoritma adalah $\Theta(n)$.**
- **Ide dari counting sort:**
  - Untuk setiap input elemen x, tentukan jumlah elemen yang lebih kecil dari x.
  - Gunakan informasi tersebut untuk menentukan secara langsung tempat yang tepat untuk x.
  - Contoh: jika terdapat 17 elemen yang lebih kecil dari x, maka x berada di tempat ke-18.
- **Selain array input $A[1 \ldots n]$, diperlukan 2 array lagi yaitu $B[1 \ldots n]$ yang menyimpan output yang sudah terurut dan array C[0..k] sebagai penyimpanan sementara.**

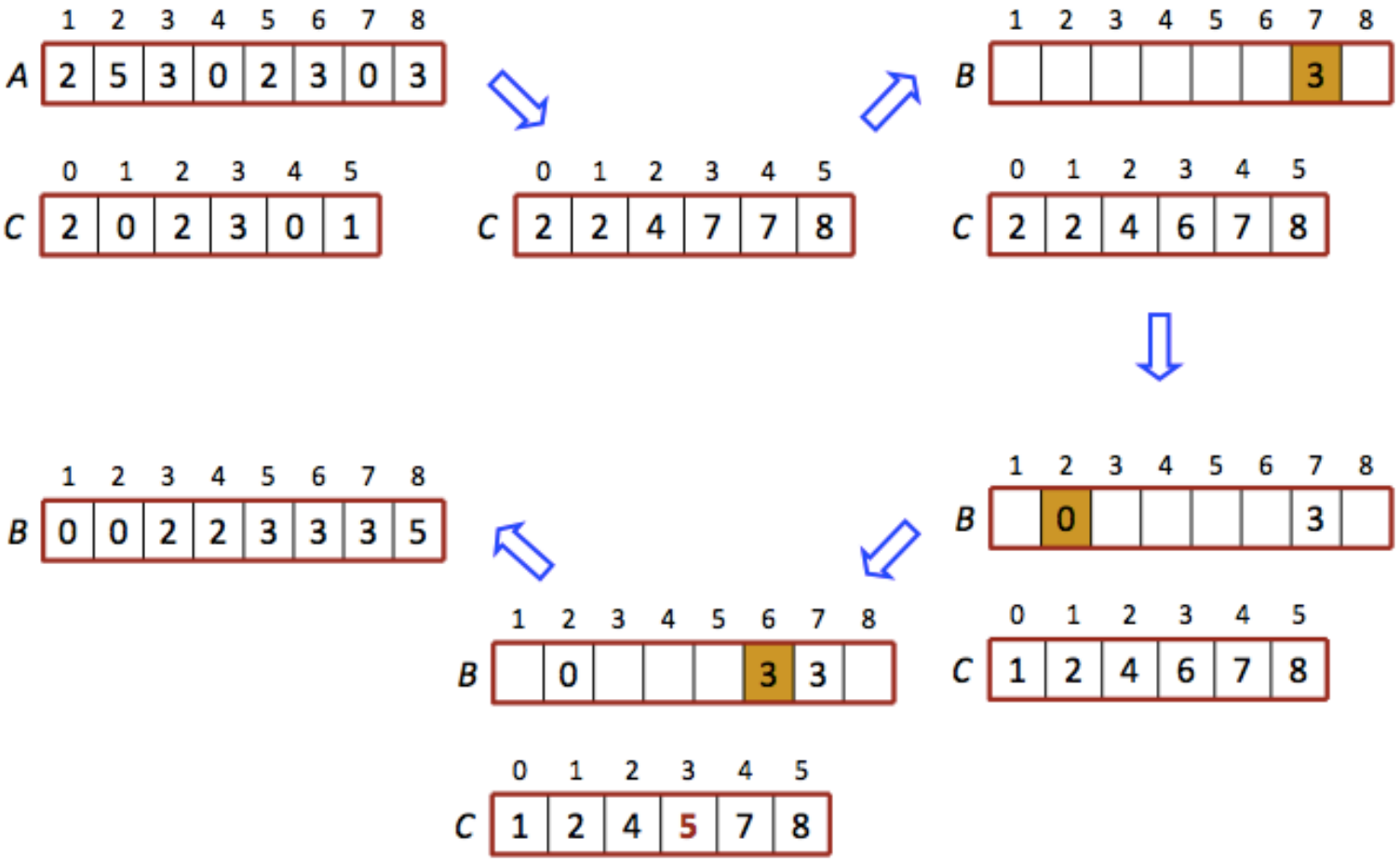
ALGORITMA COUNTING SORT

COUNTING-SORT(A, B, k)

1. for $i \leftarrow 0$ to k // $\Theta(k)$
2. do $C[i] \leftarrow 0$ // $\Theta(k)$
3. for $j \leftarrow 1$ to $length[A]$ // $\Theta(n)$
4. do $C[A[j]] \leftarrow C[A[j]] + 1$ // $\Theta(n)$
5. /* $C[i]$ memuat jumlah elemen pada i. */
6. For $i \leftarrow 1$ to k // $\Theta(k)$
7. do $C[i] \leftarrow C[i] + C[i - 1]$ // $\Theta(k)$
8. /* $C[i]$ memuat jumlah elemen kurang dari atau sama dengan i. */
9. for $j \leftarrow length[A]$ downto 1 // $\Theta(n)$
10. do $B[C[A[j]]] \leftarrow A[j]$ // $\Theta(n)$
11. $C[A[j]] \leftarrow C[A[j]] - 1$ // $\Theta(n)$

- **Running time: $\Theta(n+k)$**

COUNTING SORT: ILUSTRASI



- **Analisis algoritma:**
  - Loop pada baris 1-2 memerlukan waktu $\Theta(k)$
  - Loop pada baris 3-4 memerlukan waktu $\Theta(n)$
  - Loop pada baris 6-7 memerlukan waktu $\Theta(k)$
  - Loop pada baris 9-11 memerlukan waktu $\Theta(n)$
  - Maka total waktu yang diperlukan adalah: $\Theta(k) + \Theta(n) + \Theta(k) + \Theta(n) = \Theta(k + n)$
- **Pada aplikasi, counting sort biasanya dipakai jika $k = O(n)$, sehingga running time adalah $\Theta(n)$.**

PROPERTI COUNTING SORT

- **Counting sort merupakan stable sort → beberapa elemen dengan nilai yang sama akan ditempatkan pada array terurut dengan urutan yang sama pada input array.**
- **Counting sort lebih efisien daripada batas bawah pada comparison sort $\Omega(n \lg n)$ karena bukan merupakan comparison sort.**
  - Tidak ada perbandingan antar elemen.
  - Counting sort menggunakan nilai sebenarnya dari elemen untuk mengindeksnya ke array.
- **Counting sort dapat digunakan pada radix sort.**

RADIX SORT

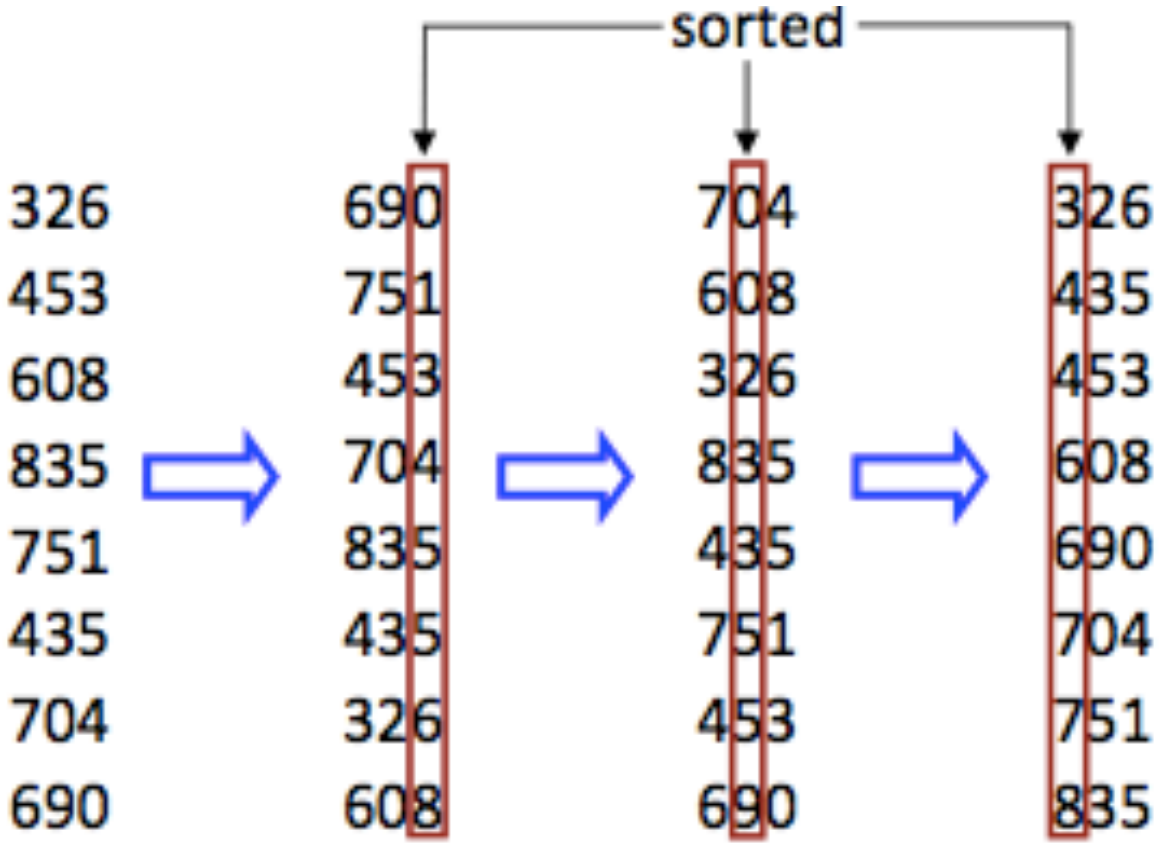
- **Asumsi**:
  - Setiap elemen pada array A terdiri dari d digit, dengan digit 1 adalah digit dengan order paling rendah, sedangkan digit d adalah digit dengan order tertinggi.
- **Secara intuitive, radix sort digunakan ketika orang mengurutkan nama**
  - Di sini, radix adalah 26 huruf alfabet.
  - Daftar nama diurutkan menurut huruf pertama pada setiap nama → nama dibagi menjadi 26 kelas.
- **Radix sort mengurutkan bilangan dari digit yang paling tidak signifikan.**
  - Pada putaran pertama, angka diurutkan berdasarkan digit paling tidak signifikan dan mengkombinasikannya pada array.
  - Pada urutan kedua, angka diurutkan berdasarkan digit kedua yang paling tidak signifikan, dst.

ALGORITMA RADIX SORT

- **Ide: urutkan dari least significant digit dahulu.**
- **Algoritma**:

RADIX-SORT(*A*, *d*)
1. for *i* ← 1 to *d*
2. do use a stable sort to sort array *A* on digit *i*

RADIX SORT: ILUSTRASI



ANALISIS KOMPLEKSITAS

- **Running time tergantung pada algoritma sorting yang digunakan.**
  - Jika setiap angka berada pada range 1 sampai k, dan k tidak terlalu besar, maka Counting_Sort adalah pilihan terbaik.
    - Setiap putaran memerlukan waktu $O(n + k)$.
    - Terdapat d putaran → total waktu $\Theta(dn+dk)$.
  - Jadi jika d konstanta dan $k = \Theta(n)$, maka Radix sort memiliki waktu linier.
- **Lemma: diberikan *n* *d*-digit bilangan di mana setiap digit bernilai antara 0 sampai k, RADIXSORT melakukan pengurutan pada larik ini dalam waktu $\Theta(d(n + k))$.**

BUCKET SORT

- **Bucket sort mengasumsikan bahwa input dibangkitkan oleh proses random yang mendistribusikan elemen secara uniform dan independen pada interval [0, 1).**
- **Prinsip:**
  - Membagi interval [0, 1) pada n sub-interval dengan ukuran yang sama, yang disebut sebagai bucket, kemudian mendistribusikan n input angka ke bucket-bucket tersebut.
  - Karena input didistribusikan secara uniform pada (0, 1), tidak akan ada banyak angka yang masuk pada setiap bucket.
  - Untuk menghasilkan output, urutkan angka pada setiap bucket, kemudian urutkan bucket dan daftar elemen pada setiap bucket.

ALGORITMA

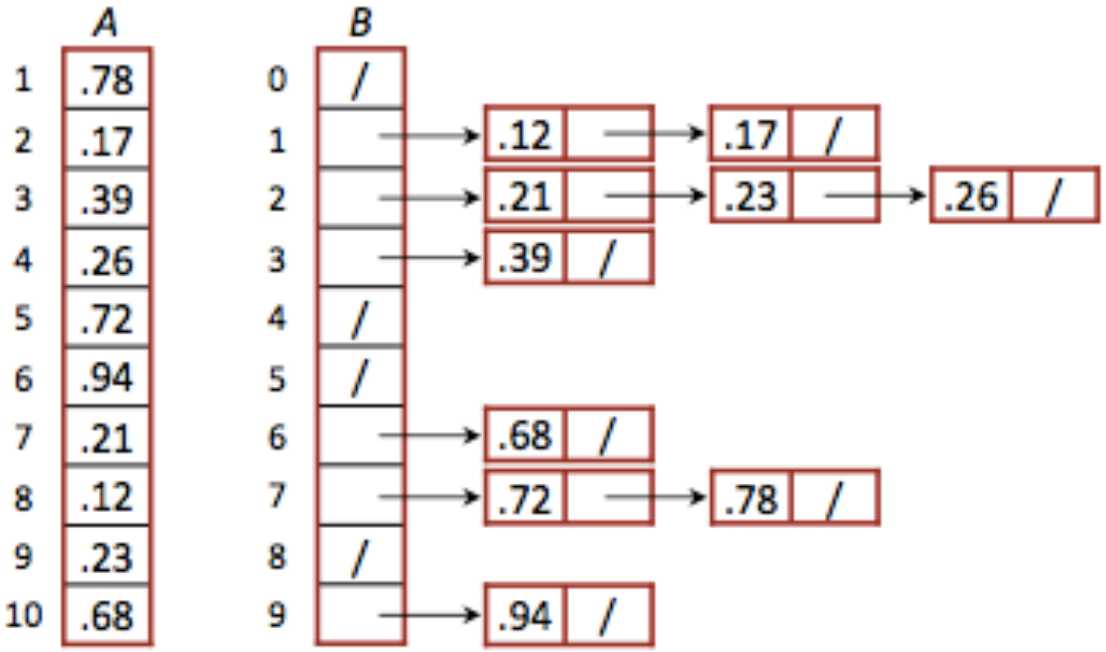
- **Input: n elemen array A dan setiap elemen memenuhi 0 ≤ *A*[*i*] < 1.**
- **Diperlukan juga array B[0 . . *n* -1] dari linked-lists (bucket) sebagai array list kosong.**

BUCKET-SORT(*A*, *n*)

1. for $i \leftarrow 1$ to n
2. do insert $A[i]$ into list $B[\lfloor n \cdot A[i] \rfloor]$
3. for $i \leftarrow 0$ to $n - 1$
4. do sort list $B[i]$ with insertion sort
5. concatenate lists $B[0], B[1], \ldots, B[n - 1]$ together in order
6. return the concatenated lists

BUCKET SORT: ILUSTRASI

- **Input array A[1..10].**
- **Array B[0..9] mengurutkan list (baris 5).**
- **Bucket i menyimpan nilai-nilai pada interval [*i*/10, (*i* +1)/ 10].**
- **Output memuat gabungan list pada B[0], B[1], … B[9].**



ANALISIS

- **Sangat bergantung pada requirement bahwa tidak ada bucket mendapatkan terlalu banyak nilai.**
- **Semua baris kecuali baris 4 memerlukan waktu O(*n*) untuk worst case.**
- **Secara intuitif, jika setiap bucket mendapatkan elemen berupa suatu angka konstan, diperlukan waktu *O*(1) untuk mengurutkan setiap bucket➔*O*(*n*) untuk semua bucket.**
- **Diharapkan bahwa setiap bucket memiliki sedikit elemen, karena terdapat rata-rata 1 elemen setiap bucket.**
- **Dengan bucket sort, jika input tidak dihasilkan dari distribusi uniform pada [0, 1), semua perhitungan performance meleset.**
  - Tetapi algoritma tetap benar.

RADIX SORT DENGAN MOST SIGNIFICANT DIGIT

- **Merupakan gabungan radix sort dan penggunaan bucket.**
- **Algoritma:**
  - Ambil most significant digit pada setiap input.
  - Kelompokkan elemen dengan most significant digit yang sama, masukkan pada bucket.
  - Secara rekursif, urutkan elemen pada setiap bucket.
  - Gabungkan semua bucket.

PERBANDINGAN ALGORITMA SORTING

Algoritma	Worst case	Average case	Best case	Extra space	Stable
Selection sort	$O(n^2)$	$O(n^2)$	$O(n^2)$	O(1)	Tidak
Insertion sort	$O(n^2)$	$O(n^2)$	O(n)	O(1)	Ya
Bubble sort	$O(n^2)$	$O(n^2)$	$O(n^2)$	O(1)	Ya
Merge sort	O(n lg n)	O(n lg n)	O(n lg n)	O(n)	Ya
Heap sort	O(n lg n)	O(n lg n)	O(n lg n)	O(1)	Tidak
Quick sort	$O(n^2)$	O(n lg n)	O(n lg n)	O(log n)	Tidak
Counting sort	O(n+k)	O(n+k)	O(n+k)	O(n+k)	Ya
Radix sort	O(n+k)	O(n+k)	O(n+k)	O(n+k)	Ya
Bucket sort	$O(n^2)$	O(n+k)	O(n)	$O(n^2)$	Ya

LATIHAN

- **Lakukan counting sort untuk data: A = [6, 0, 2, 0, 1, 3, 4, 6, 1, 3, 2].**
- **Lakukan radix sort untuk data: COW, DOG, SEA, RUG, ROW, MOB, BOX, TAB, TAB, BAR, EAR, TAR, DIG, BIG, TEA, NOW, FOX.**
- **Lakukan bucket sort untuk data: A = [0.79, 0.13, 0.16, 0.64, 0.39, 0.20, 0.89, 0.53, 0.71, 0.42]**