Metody řazení s lineární časovou složitostí

Counting sort
Radix sort
Bucket sort

Úvod

- Porovnávací řazení: pořadí prvků v řazené posloupnosti je určováno pouze na základě porovnávání vstupních prvků
  - Libovolné porovnání must musí mít v nejhorším případě složitost $\Omega(n \lg n)$
  - Merge sort a heapsort jsou asymptoticky optimální
- Řazení s lineární časovou složitostí
  - K uspořádání prvků používá jiných operací než je porovnávání
  - Counting sort, radix sort, a bucket sort

Dolní hranice pro Comparison Sort

Decision-Tree Model

- Decision tree (Rozhodovací strom)
  - Úplný binární rozhodovací strom, který reprezentuje porovnání mezi prvky vstupního pole dané velikosti.
    - i:j → porovnáme a[i] a a[j]
    - Každý list je označen permutací $<\pi(1), \pi(2), \ldots, \pi(n)>$
      - n! je celkový počet permutací
  - Vykonávání kroků řadícího algoritmu odpovídá průchodu stromem od kořene k listu
  - Nutnou podmínkou toho, aby porovnávací řazení správně fungovalo je, že každá z n! permutací provedenou na vstupních prvcích se musí vyskytnout v jednom z listů rozhodovacího stromu a každý z těchto listů musí být dosažitelný z kořene.

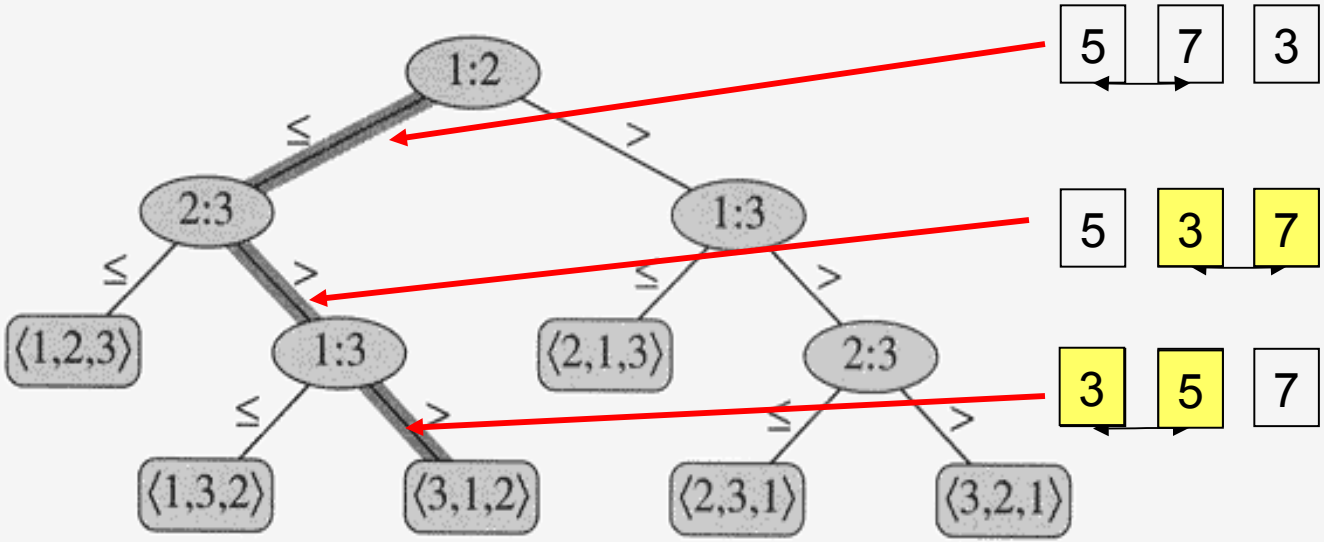


Figure 8.1 The decision tree for insertion sort operating on three elements. An internal node annotated by $i{:}j$ indicates a comparison between a_i and a_j. A leaf annotated by the permutation $\langle \pi(1), \pi(2), \ldots, \pi(n) \rangle$ indicates the ordering $a_{\pi(1)} \leq a_{\pi(2)} \leq \cdots \leq a_{\pi(n)}$. The shaded path indicates the decisions made when sorting the input sequence $\langle a_1 = 6, a_2 = 8, a_3 = 5 \rangle$; the permutation $\langle 3, 1, 2 \rangle$ at the leaf indicates that the sorted ordering is $a_3 = 5 \leq a_1 = 6 \leq a_2 = 8$. There are $3! = 6$ possible permutations of the input elements, so the decision tree must have at least 6 leaves.

Dolní hranice pro nejhorší případ

- Délka nejdelší cesty z kořene do jeho dosažitelný listů reprezentuje nejhorší případ počtu porovnávání, které odpovídá jednotlivým krokům algoritmu
  - Výška rozhodovacího stromu
- ***Věta 8.1.*** Libovolný algoritmus porovnávacího řazení vyžaduje v nejhorším případě $\Omega(n \lg n)$.
- Uvažujme rozhodovací strom výšky h s l dosažitelnými listy
  - $n! \leq l \leq 2^h$
    - $h \geq \lg(n!)$ (protože lg funkce je monotónně rostoucí)
      $= \Omega(n \lg n)$

Counting Sort

Counting Sort Overview

- Předpoklad: n vstupních celočíselných prvků z rozsahu 0 to k (integer).
  - $\Theta(n+k)$ ➔ Když k=O(n), counting sort: $\Theta(n)$
- Základní myšlenka
  - Pro každý vstupní prvek x, určíme počet prvků menších nebo rovných x
  - Pro každé integer číslo i ($0 \leq i \leq k$), určíme kolik prvků má hodnotu i
    - Pak také víme, kolik prvků je menších nebo rovných i
- Algoritmus využívá následující proměnné
  - A[1..n]: vstupní prvky
  - B[1..n]: seřazené pole (výstup)
  - C[0..k]: pole ve kterém se uchovává počet prvků menších nebo rovných i

Counting Sort příklad

(Rozsah prvků od 0 do 5)

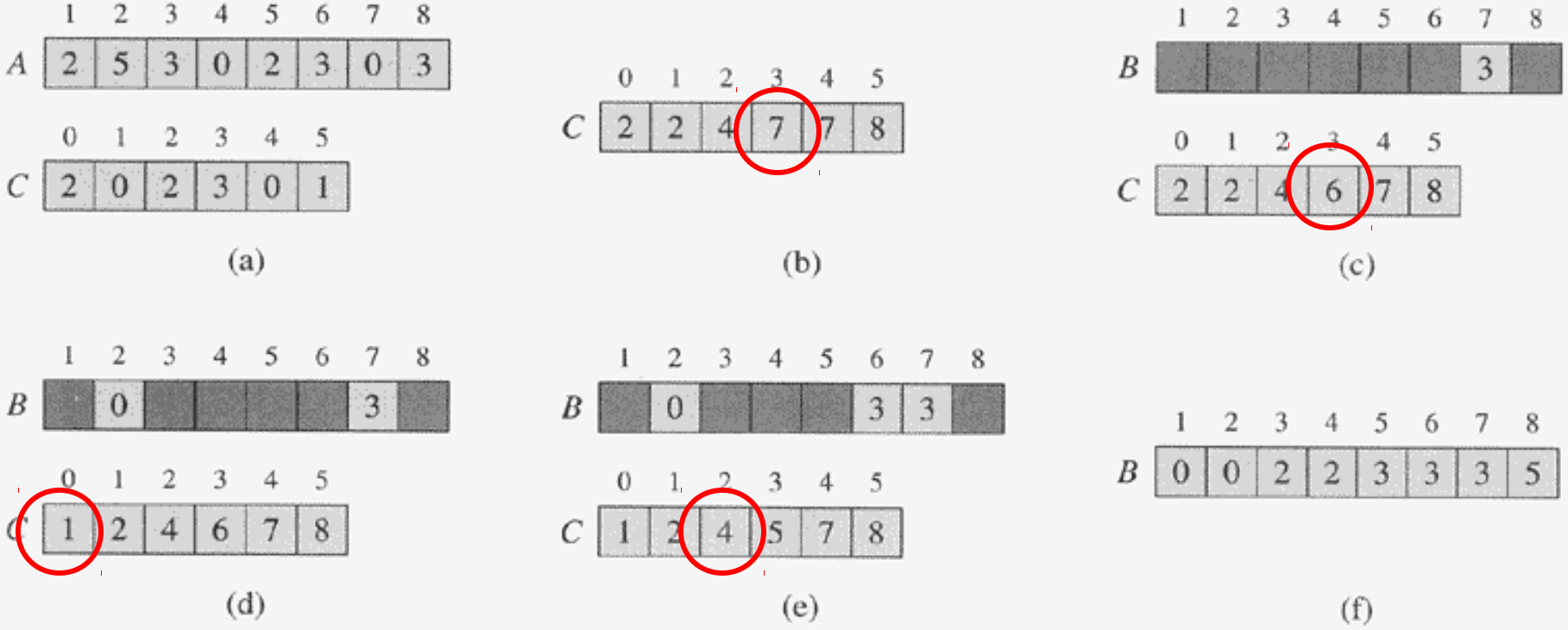
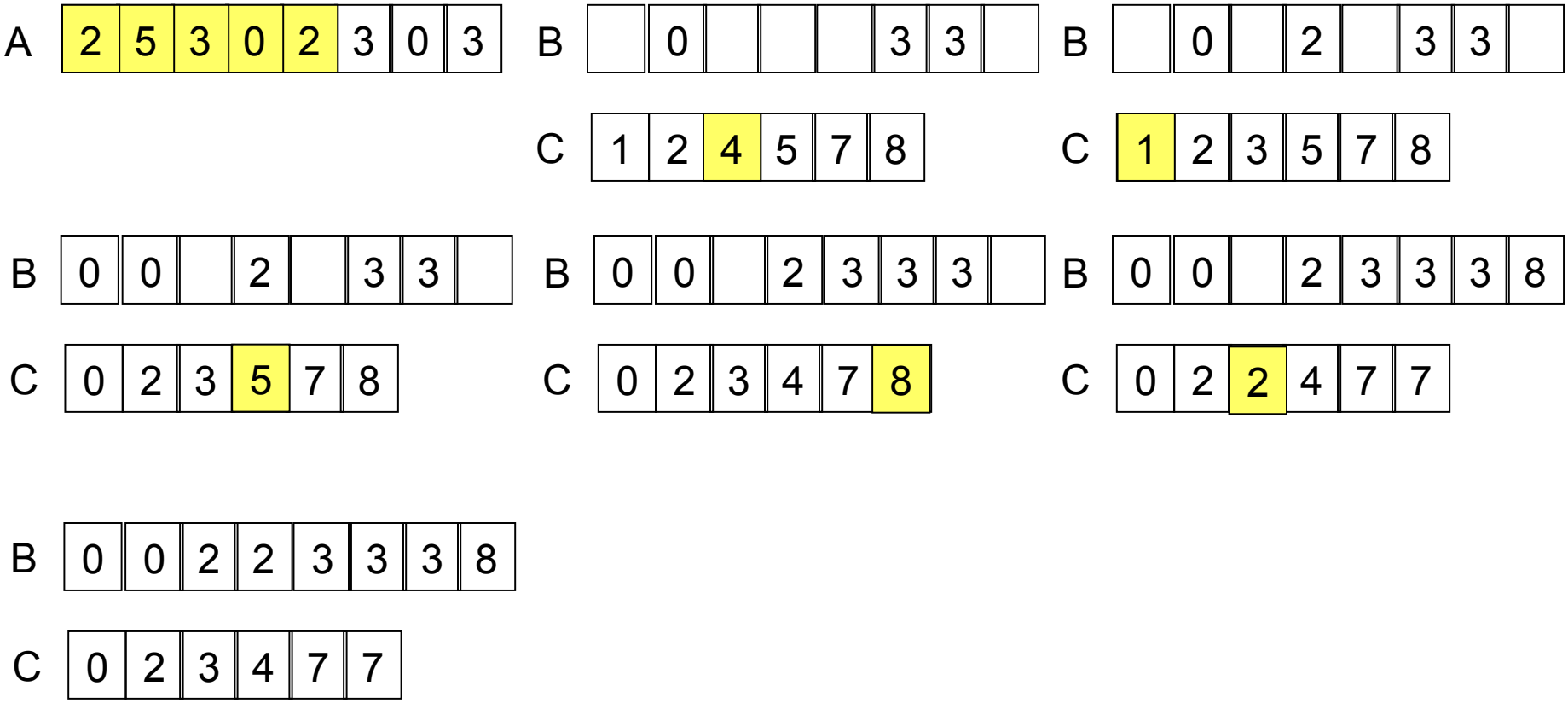


Figure 8.2 The operation of COUNTING-SORT on an input array $A[1..8]$, where each element of A is a nonnegative integer no larger than $k = 5$. **(a)** The array A and the auxiliary array C after line 4. **(b)** The array C after line 7. **(c)–(e)** The output array B and the auxiliary array C after one, two, and three iterations of the loop in lines 9–11, respectively. Only the lightly shaded elements of array B have been filled in. **(f)** The final sorted output array B.

Counting Sort příklad (pokračování)



Counting Sort Algoritmus

Θ(n+k)

```
COUNTING-SORT(A, B, k)
 1  for i ← 0 to k                                   Θ(k)
 2       do C[i] ← 0
 3  for j ← 1 to length[A]                           Θ(n)
 4       do C[A[j]] ← C[A[j]] + 1
 5  ▷ C[i] now contains the number of elements equal to i.
 6  for i ← 1 to k                                   Θ(k)
 7       do C[i] ← C[i] + C[i − 1]
 8  ▷ C[i] now contains the number of elements less than or equal to i.
 9  for j ← length[A] downto 1                       Θ(n)
10       do B[C[A[j]]] ← A[j]
11          C[A[j]] ← C[A[j]] − 1
```

Counting Sort je stabilní

- Algoritmus řazení je stabilní pokud
  - Čísla se stejnou hodnotou se ve výstupním (seřazeném) poli vyskytne ve stejné pozici, jako ve vstupním poli

- Řádek 9 algoritmu counting sort: for j←length[A] down to 1 je základem toho, aby algoritmus byl stabilní

Radix Sort

Radix Sort Příklad

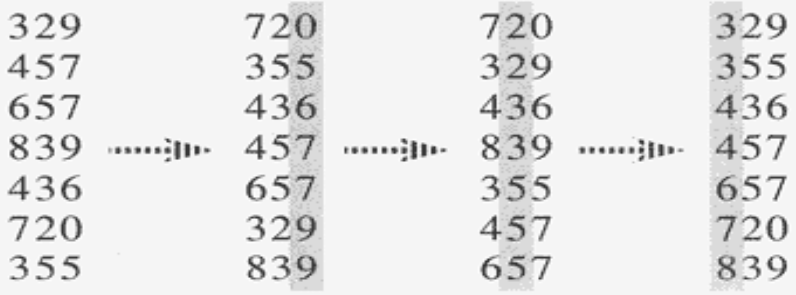


Figure 8.3 The operation of radix sort on a list of seven 3-digit numbers. The leftmost column is the input. The remaining columns show the list after successive sorts on increasingly significant digit positions. Shading indicates the digit position sorted on to produce each list from the previous one.

1. Nejprve se začíná řadit podle nejméně významné číslice
2. To je základem toho, že algoritmus řazení čísel je stabilní

Radix Sort Algorithm

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

d je celkový počet čísel

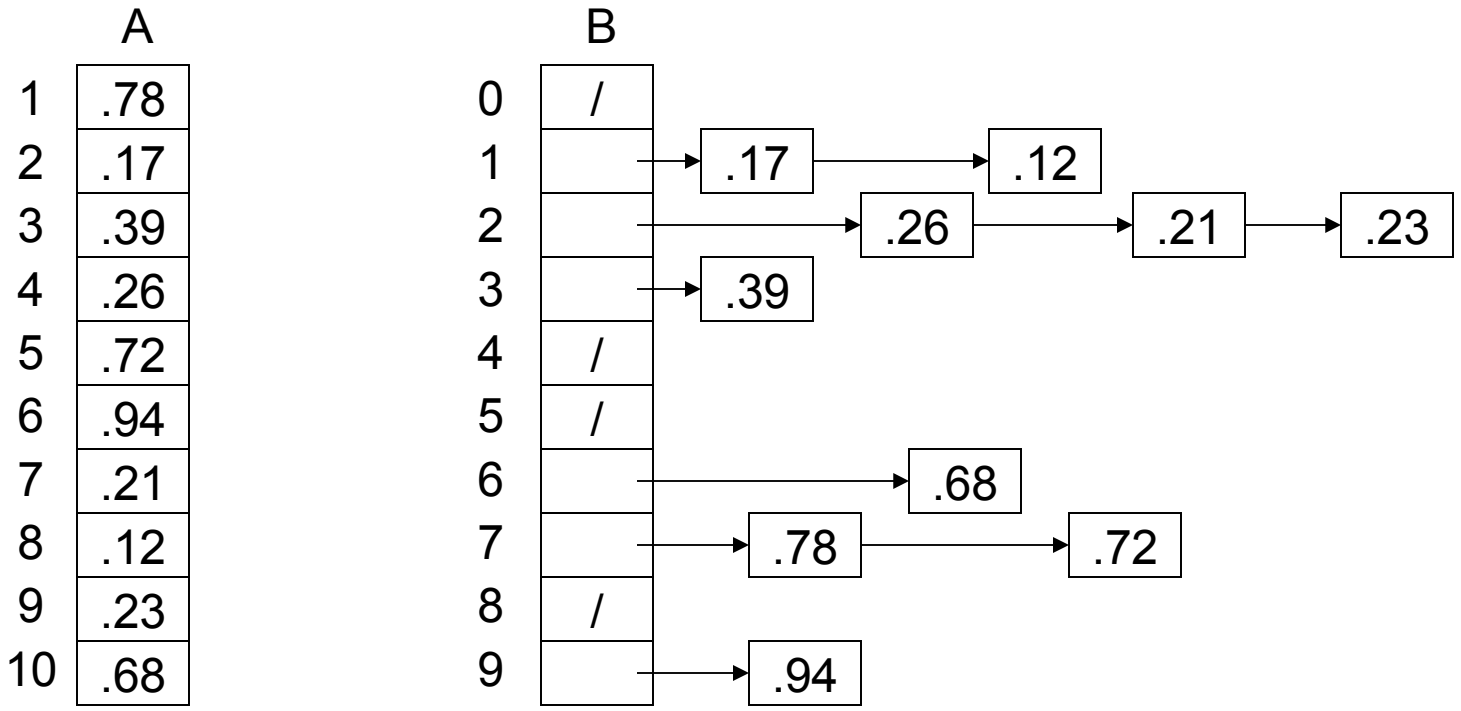
Lemma 8.3 Pro n d-místných čísel, u kterých každá číslice nabývá jedné z k možných hodnot, provádí RADIX-SORT řazení těchto čísel v čase $\Theta(d(n+k))$

Bucket Sort

Bucket Sort

- Bucket sort běží v lineárním čase, pokud mají vstupní prvky rovnoměrné rozložení v interval [0, 1)
- Základní myšlenka
  - Rozdělíme interval [0,1) na n rovnoměrně rozložených subintervals (buckets)
  - Rozdělíme n čísle do těchto subintervalů
  - Seřadíme čísla v každém bucketu
  - Procházíme buckety v daném pořadí, vypisujeme prvky vkaždém bucketu

Bucket Sort příklad



Bucket Sort příklad (pokračování)

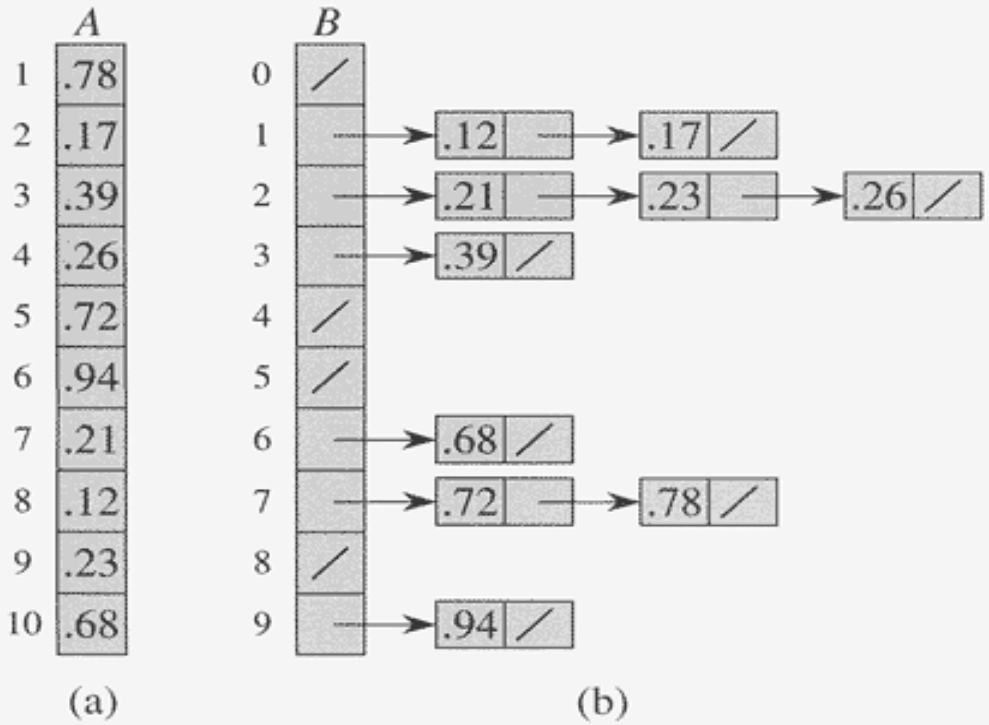


Figure 8.4 The operation of BUCKET-SORT. **(a)** The input array $A[1..10]$. **(b)** The array $B[0..9]$ of sorted lists (buckets) after line 5 of the algorithm. Bucket i holds values in the half-open interval $[i/10, (i+1)/10)$. The sorted output consists of a concatenation in order of the lists $B[0], B[1], \ldots, B[9]$.

Bucket Sort Algoritmus

```
BUCKET-SORT(A)
1  n ← length[A]
2  for i ← 1 to n
3        do insert A[i] into list B[⌊n A[i]⌋]
4  for i ← 0 to n − 1
5        do sort list B[i] with insertion sort
6  concatenate the lists B[0], B[1], . . . , B[n − 1] together in order
```