2IL50 Data Structures

2023-24 Q3

Lecture 4: Sorting in linear time

TU/e EINDHOVEN UNIVERSITY OF TECHNOLOGY

Heaps

One more time …

Building a heap

Build-Max-Heap(A)

$A.\text{heap-size} = A.\text{length}$
for $i = A.\text{length}$ **downto** 1
    Max-Heapify(A, i)

Building a heap

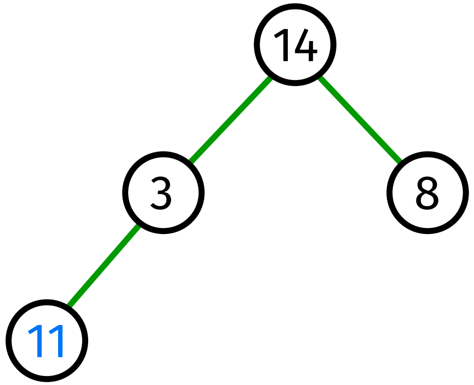
Build-Max-Heap2(A)

$A.\text{heap-size} = 1$
for $i = 2$ **to** $A.\text{length}$: Insert(A, $A[i]$)

14	3	8	11	2	24	35	28	16	5	20	21

Insert(A, key)

$A.\text{heap-size} = A.\text{heap-size} + 1$
$A[A.\text{heap-size}] = -\infty$
Increase-Key(A, $A.\text{heap-size}$, key)



Lower bound worst case running time?

$A[i]$ moves up until it reaches the correct position

Building a heap

Build-Max-Heap2(A)

$A.\text{heap-size} = 1$

for $i = 2$ **to** $A.\text{length}$: Insert(A, $A[i]$)

Running time: $\Theta(1) + \sum_{2 \le i \le n}$(time for Insert$(A, A[i])$)

Insert(A, $A[i]$) takes $O(\log i) = O(\log n)$ time ➡ worst case $O(n \log n)$

If A is sorted in increasing order, then $A[i]$ is always the largest element when Insert(A, $A[i]$) is called and must move all the way up the tree

➡ Insert(A, $A[i]$) takes $\Omega(\log i)$ time.

Worst case running time: $\Theta(1) + \sum_{2 \le i \le n} \Omega(\log i) = \Omega(1 + \sum_{2 \le i \le n} \log i) = \Omega\,(n \log n)$

since $\sum_{2 \le i \le n} \log i \ge \sum_{n/2 \le i \le n} \log(n/2) = n/2 \log(n/2)$

Quiz

1. $\log^2 n = \Theta(\log n^2)$? no

2. $\sqrt{n} = \Omega(\log^4 n)$? yes

3. $2^{\log n} = \Omega(n^2)$? no

4. $2^n = \Omega(n^2)$? yes

5. $\log(\sqrt{n}) = \Theta(\log n)$? yes

Sorting in linear time

The sorting problem

Input: a sequence of n numbers $A = \langle a_1, a_2, \ldots, a_n \rangle$
Output: a permutation of the input such that $\langle a_{i1} \le a_{i2} \le \cdots \le a_{in} \rangle$

Why do we care so much about sorting?

- sorting is used by many applications
- (first) step of many algorithms
- many techniques can be illustrated by studying sorting

Can we sort faster than $\Theta(n \log n)$??

Worst case running time of sorting algorithms:

InsertionSort: $\Theta(n^2)$

MergeSort: $\Theta(n \log n)$

HeapSort: $\Theta(n \log n)$

Can we do this faster? $\Theta(n \log \log n)$? $\Theta(n)$?

Upper and lower bounds

Upper bound

How do you show that a problem (for example sorting) can be solved in $\Theta(f(n))$ time?

➡ give an algorithm that solves the problem in $\Theta(f(n))$ time.

Lower bound

How do you show that a problem (for example sorting) cannot be solved faster than in $\Theta(f(n))$ time?

➡ prove that every possible algorithm that solves the problem needs $\Omega(f(n))$ time.

Lower bounds

Lower bound

How do you show that a problem (for example sorting) cannot be solved faster than in $\Theta(f(n))$ time?

➡ prove that every possible algorithm that solves the problem needs $\Omega(f(n))$ time.

Model of computation: which operations is the algorithm allowed to use?

Bit-manipulations?
Random-access (array indexing) vs. pointer-machines?

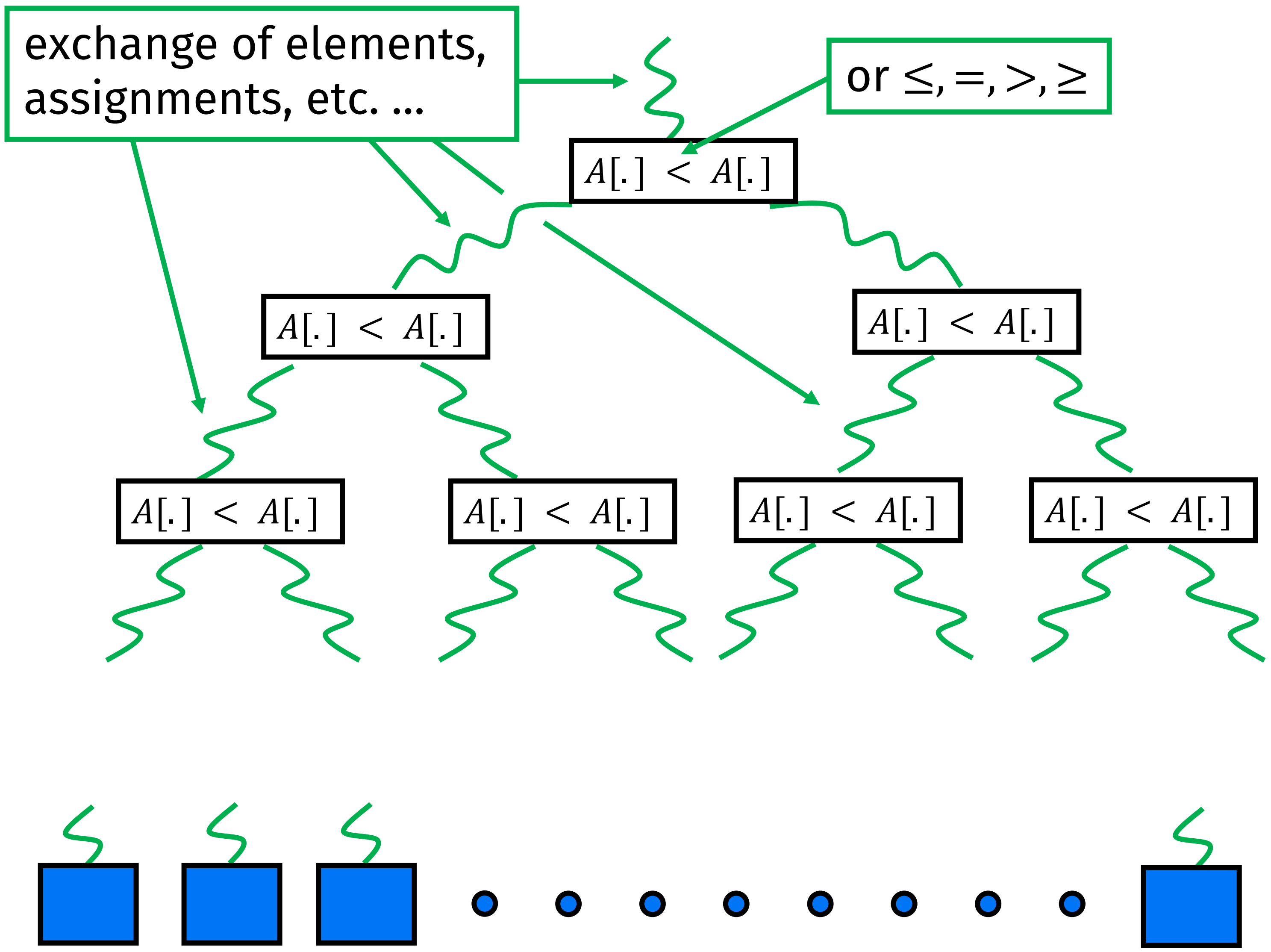
Comparison-based sorting

InsertionSort(A)

initialize: sort $A[1]$
for $j = 2$ **to** $A.\text{length}$
    $\text{key} = A[j]$
    $i = j - 1$
    while $i > 0$ and $A[i] > \text{key}$
        $A[i+1] = A[i]$
        $i = i - 1$
    $A[i+1] = \text{key}$

Which steps precisely the algorithm executes,
and hence, which element ends up where,
only depends on the result of comparisons between the input elements.

Decision tree for comparison-based sorting



Proving comparison-based lower bound

Proving lower bound of $f(n)$ comparisons

height of decision tree

- Proof by contradiction
- Assume algorithm with worst case $f(n) - 1$ comparisons
- Show two different inputs with same comparison results

➡ Both inputs follow same path in decision tree

➡ Algorithm cannot be correct

Easy approach

- Count number of different inputs (requiring different outputs)
- Every different input must correspond to a distinct leaf

Hard approach

- Maintain set of possible inputs corresponding to comparisons
- Show that at least two inputs remain after $f(n) - 1$ comparisons
- Cannot choose comparisons, can choose results

Comparison-based sorting

every permutation of the input follows a different path in the decision tree

➡ the decision tree has at least $n!$ leaves

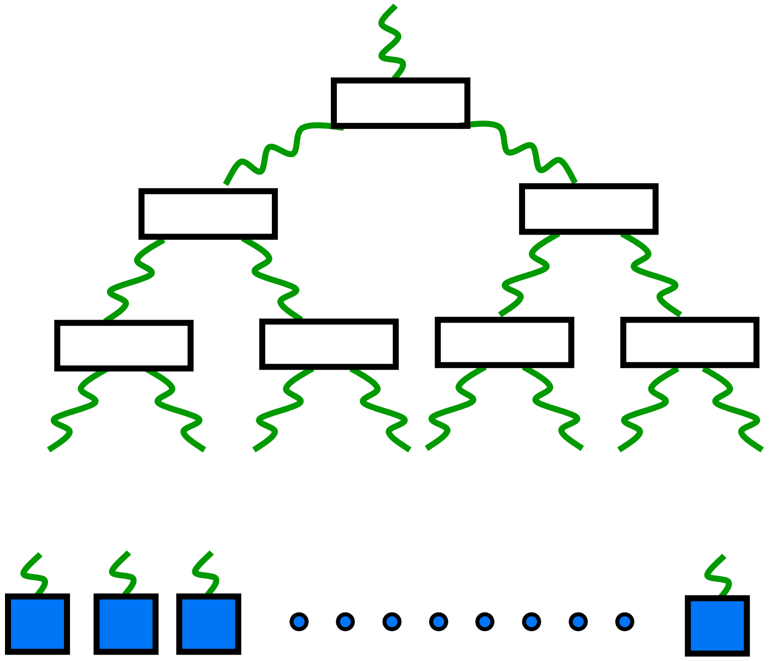
the height of a binary tree with $n!$ leaves is at least $\log(n!)$

worst case running time

$\geq$ longest path from root to leaf

$=$ the height of the tree

$\geq \log(n!) \quad = \quad \Omega(n \log n)$



Lower bound for comparison-based sorting

Theorem

Any comparison-based sorting algorithm requires $\Omega(n \log n)$ comparisons in the worst case.

➡ The worst-case running time of MergeSort and HeapSort is optimal.

Sorting in linear time …

Three algorithms which are faster:

1. CountingSort

2. RadixSort

3. BucketSort

not comparison-based, make assumptions on the input

CountingSort

Input: array $A[1{:}n]$ of numbers
Assumption: the input elements are integers in the range 0 to k, for some k

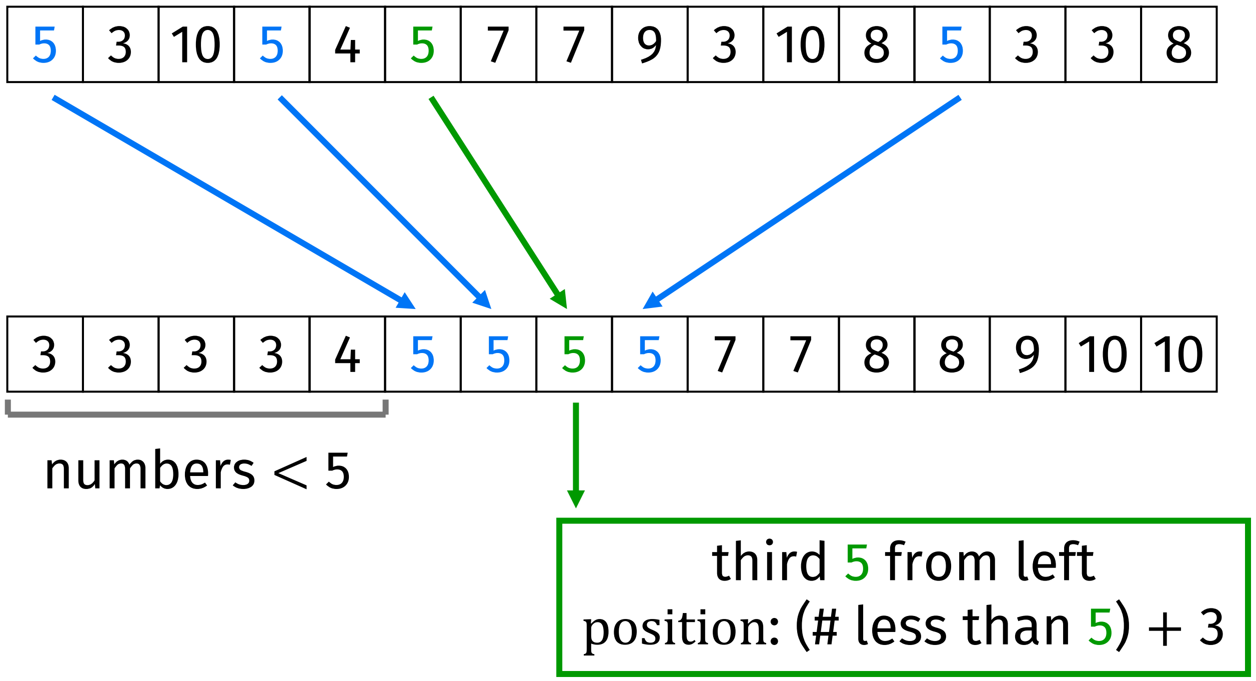
Main idea: count for every $A[i]$ the number of elements less than $A[i]$
➡ position of $A[i]$ in the output array

Beware of elements that have the same value!

$$\begin{aligned}\text{position}(i) \;&=\; \text{number of elements less than } A[i] \text{ in } A[1{:}n] \\ &+\; \text{number of elements equal to } A[i] \text{ in } A[1{:}i]\end{aligned}$$

CountingSort

$$\text{position}(i) = \text{number of elements less than } A[i] \text{ in } A[1{:}n]$$
$$+ \text{ number of elements equal to } A[i] \text{ in } A[1{:}i]$$



CountingSort

$$\begin{aligned}\mathrm{position}(i) \;=\;& \text{number of elements less than } A[i] \text{ in } A[1{:}\,n] \\ +\;& \text{number of elements equal to } A[i] \text{ in } A[1{:}\,i]\end{aligned}$$

Lemma

If every element $A[i]$ is placed on $\mathrm{position}(i)$,
then the array is sorted and the sorted order is stable.

Numbers with the same value appear in the same order in the output array as they do in the input array.

CountingSort

$C[i]$ will contain the number of elements $\leq\ i$

CountingSort(A, k)

// Input: array $A[1{:}n]$ of integers in the range $0 \ldots k$

// Output: array $B[1{:}n]$ which contains the elements of A, sorted

for $i = 0$ **to** k: $\boxed{C[i]} = 0$

for $j = 1$ **to** $A.\mathrm{length}$: $C[A[j]] = C[A[j]] + 1$

// $C[i]$ now contains the number of elements equal to i

for $i = 1$ **to** k: $C[i] = C[i] + C[i-1]$

// $C[i]$ now contains the number of elements less than or equal to i

for $j = A.\mathrm{length}$ **downto** 1

    $B[C[A[j]]] = A[j]$; $C[A[j]] = C[A[j]] - 1$

CountingSort

```
CountingSort(A, k)
    // Input: array A[1:n] of integers in the range 0 … k
    // Output: array B[1:n] which contains the elements of A, sorted
    for i = 0 to k:  C[i] = 0
    for j = 1 to A.length:  C[A[j]] = C[A[j]] + 1
    // C[i] now contains the number of elements equal to i
    for i = 1 to k:  C[i] = C[i] + C[i − 1]
    // C[i] now contains the number of elements less than or equal to i
    for j = A.length downto 1
        B[C[A[j]]] = A[j];  C[A[j]] = C[A[j]] − 1
```

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
A	5	3	10	5	4	5	7	7	9	3	10	8	5	3	3	8

CountingSort

```
CountingSort(A, k)
    // Input: array A[1:n] of integers in the range 0 ... k
    // Output: array B[1:n] which contains the elements of A, sorted
→   for i = 0 to k:  C[i] = 0
    for j = 1 to A.length:  C[A[j]] = C[A[j]] + 1
    // C[i] now contains the number of elements equal to i
    for i = 1 to k:  C[i] = C[i] + C[i − 1]
    // C[i] now contains the number of elements less than or equal to i
    for j = A.length downto 1
        B[C[A[j]]] = A[j];  C[A[j]] = C[A[j]] − 1
```

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
A	5	3	10	5	4	5	7	7	9	3	10	8	5	3	3	8

	0	1	2	3	4	5	6	7	8	9	10
C	0	0	0	0	0	0	0	0	0	0	0

CountingSort

```
CountingSort(A, k)
    // Input: array A[1:n] of integers in the range 0 ... k
    // Output: array B[1:n] which contains the elements of A, sorted
for i = 0 to k:  C[i] = 0
for j = 1 to A.length:  C[A[j]] = C[A[j]] + 1
// C[i] now contains the number of elements equal to i
for i = 1 to k:  C[i] = C[i] + C[i − 1]
// C[i] now contains the number of elements less than or equal to i
for j = A.length downto 1
    B[C[A[j]]] = A[j];  C[A[j]] = C[A[j]] − 1
```

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
A	5	3	10	5	4	5	7	7	9	3	10	8	5	3	3	8

	0	1	2	3	4	5	6	7	8	9	10
C	0	0	0	0	0	1	0	0	0	0	0

CountingSort

CountingSort(A, k)

  // Input: array $A[1{:}n]$ of integers in the range $0 \dots k$
  // Output: array $B[1{:}n]$ which contains the elements of A, sorted

for $i = 0$ **to** k: $C[i] = 0$
for $j = 1$ **to** $A.\text{length}$: $C[A[j]] = C[A[j]] + 1$
// $C[i]$ now contains the number of elements equal to i
for $i = 1$ **to** k: $C[i] = C[i] + C[i-1]$
// $C[i]$ now contains the number of elements less than or equal to i
for $j = A.\text{length}$ **downto** 1
  $B[C[A[j]]] = A[j]$; $C[A[j]] = C[A[j]] - 1$

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
A	5	3	10	5	4	5	7	7	9	3	10	8	5	3	3	8

	0	1	2	3	4	5	6	7	8	9	10
C	0	0	0	1	0	1	0	0	0	0	0

CountingSort

CountingSort(A, k)

// Input: array $A[1{:}n]$ of integers in the range $0 \dots k$

// Output: array $B[1{:}n]$ which contains the elements of A, sorted

for $i = 0$ **to** k: $C[i] = 0$

for $j = 1$ **to** $A.\text{length}$: $C[A[j]] = C[A[j]] + 1$

// $C[i]$ now contains the number of elements equal to i

for $i = 1$ **to** k: $C[i] = C[i] + C[i-1]$

// $C[i]$ now contains the number of elements less than or equal to i

for $j = A.\text{length}$ **downto** 1

    $B[C[A[j]]] = A[j]$; $C[A[j]] = C[A[j]] - 1$

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
A	5	3	10	5	4	5	7	7	9	3	10	8	5	3	3	8

	0	1	2	3	4	5	6	7	8	9	10
C	0	0	0	1	0	1	0	0	0	0	1

CountingSort

```
CountingSort(A, k)
    // Input: array A[1:n] of integers in the range 0 ... k
    // Output: array B[1:n] which contains the elements of A, sorted
    for i = 0 to k:  C[i] = 0
    for j = 1 to A.length:  C[A[j]] = C[A[j]] + 1
    // C[i] now contains the number of elements equal to i
    for i = 1 to k:  C[i] = C[i] + C[i − 1]
    // C[i] now contains the number of elements less than or equal to i
    for j = A.length downto 1
        B[C[A[j]]] = A[j];  C[A[j]] = C[A[j]] − 1
```

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
A	5	3	10	5	4	5	7	7	9	3	10	8	5	3	3	8

	0	1	2	3	4	5	6	7	8	9	10
C	0	0	0	1	0	2	0	0	0	0	1

CountingSort

CountingSort(A, k)

// Input: array $A[1{:}n]$ of integers in the range $0 \dots k$
// Output: array $B[1{:}n]$ which contains the elements of A, sorted
for $i = 0$ **to** k: $C[i] = 0$
for $j = 1$ **to** $A.\mathrm{length}$: $C[A[j]] = C[A[j]] + 1$
// $C[i]$ now contains the number of elements equal to i
for $i = 1$ **to** k: $C[i] = C[i] + C[i-1]$
// $C[i]$ now contains the number of elements less than or equal to i
for $j = A.\mathrm{length}$ **downto** 1
    $B[C[A[j]]] = A[j]$; $C[A[j]] = C[A[j]] - 1$

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
A	5	3	10	5	4	5	7	7	9	3	10	8	5	3	3	8

	0	1	2	3	4	5	6	7	8	9	10
C	0	0	0	1	1	2	0	0	0	0	1

CountingSort

CountingSort(A, k)
    // *Input: array $A[1{:}n]$ of integers in the range $0 \ldots k$*
    // *Output: array $B[1{:}n]$ which contains the elements of A, sorted*
for $i = 0$ **to** k: $C[i] = 0$
→ **for** $j = 1$ **to** $A.\text{length}$: $C[A[j]] = C[A[j]] + 1$
// $C[i]$ now contains the number of elements equal to i
for $i = 1$ **to** k: $C[i] = C[i] + C[i-1]$
// $C[i]$ now contains the number of elements less than or equal to i
for $j = A.\text{length}$ **downto** 1
    $B[C[A[j]]] = A[j]$; $C[A[j]] = C[A[j]] - 1$

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
A	5	3	10	5	4	5	7	7	9	3	10	8	5	3	3	8

	0	1	2	3	4	5	6	7	8	9	10
C	0	0	0	1	1	3	0	0	0	0	1

CountingSort

CountingSort(A, k)

// Input: array $A[1{:}n]$ of integers in the range $0 \ldots k$

// Output: array $B[1{:}n]$ which contains the elements of A, sorted

for $i = 0$ **to** k: $C[i] = 0$

for $j = 1$ **to** $A.\text{length}$: $C[A[j]] = C[A[j]] + 1$

// $C[i]$ now contains the number of elements equal to i

for $i = 1$ **to** k: $C[i] = C[i] + C[i-1]$

// $C[i]$ now contains the number of elements less than or equal to i

for $j = A.\text{length}$ **downto** 1

  $B[C[A[j]]] = A[j]$; $C[A[j]] = C[A[j]] - 1$

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
A	5	3	10	5	4	5	7	7	9	3	10	8	5	3	3	8

	0	1	2	3	4	5	6	7	8	9	10
C	0	0	0	1	1	3	0	1	0	0	1

CountingSort

```
CountingSort(A, k)
    // Input: array A[1:n] of integers in the range 0 ... k
    // Output: array B[1:n] which contains the elements of A, sorted
    for i = 0 to k:  C[i] = 0
    for j = 1 to A.length:  C[A[j]] = C[A[j]] + 1
    // C[i] now contains the number of elements equal to i
    for i = 1 to k:  C[i] = C[i] + C[i − 1]
    // C[i] now contains the number of elements less than or equal to i
    for j = A.length downto 1
        B[C[A[j]]] = A[j];  C[A[j]] = C[A[j]] − 1
```

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
A	5	3	10	5	4	5	7	7	9	3	10	8	5	3	3	8

	0	1	2	3	4	5	6	7	8	9	10
C	0	0	0	4	1	4	0	2	2	1	2

CountingSort

CountingSort(A, k)

// Input: array $A[1{:}n]$ of integers in the range $0 \ldots k$

// Output: array $B[1{:}n]$ which contains the elements of A, sorted

for $i = 0$ **to** k: $C[i] = 0$

for $j = 1$ **to** $A.\text{length}$: $C[A[j]] = C[A[j]] + 1$

// $C[i]$ now contains the number of elements equal to i

for $i = 1$ **to** k: $C[i] = C[i] + C[i-1]$

// $C[i]$ now contains the number of elements less than or equal to i

for $j = A.\text{length}$ **downto** 1

  $B[C[A[j]]] = A[j]$; $C[A[j]] = C[A[j]] - 1$

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
A	5	3	10	5	4	5	7	7	9	3	10	8	5	3	3	8

	0	1	2	3	4	5	6	7	8	9	10
C	0	0	0	4	1	4	0	2	2	1	2

CountingSort

```
CountingSort(A, k)
    // Input: array A[1:n] of integers in the range 0 ... k
    // Output: array B[1:n] which contains the elements of A, sorted
for i = 0 to k:  C[i] = 0
for j = 1 to A.length:  C[A[j]] = C[A[j]] + 1
// C[i] now contains the number of elements equal to i
for i = 1 to k:  C[i] = C[i] + C[i − 1]
// C[i] now contains the number of elements less than or equal to i
for j = A.length downto 1
        B[C[A[j]]] = A[j];  C[A[j]] = C[A[j]] − 1
```

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
A	5	3	10	5	4	5	7	7	9	3	10	8	5	3	3	8

	0	1	2	3	4	5	6	7	8	9	10
C	0	0	0	4	1	4	0	2	2	1	2

CountingSort

CountingSort(A, k)

  // *Input: array $A[1{:}n]$ of integers in the range $0 \dots k$*

  // *Output: array $B[1{:}n]$ which contains the elements of A, sorted*

for $i = 0$ **to** k: $C[i] = 0$

for $j = 1$ **to** $A.\text{length}$: $C[A[j]] = C[A[j]] + 1$

// $C[i]$ now contains the number of elements equal to i

for $i = 1$ **to** k: $C[i] = C[i] + C[i-1]$

// $C[i]$ now contains the number of elements less than or equal to i

for $j = A.\text{length}$ **downto** 1

  $B[C[A[j]]] = A[j]$; $C[A[j]] = C[A[j]] - 1$

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
A	5	3	10	5	4	5	7	7	9	3	10	8	5	3	3	8

	0	1	2	3	4	5	6	7	8	9	10
C	0	0	0	4	1	4	0	2	2	1	2

CountingSort

CountingSort(A, k)

// *Input: array* $A[1{:}n]$ *of integers in the range* $0 \ldots k$

// *Output: array* $B[1{:}n]$ *which contains the elements of* A*, sorted*

for $i = 0$ **to** k: $C[i] = 0$

for $j = 1$ **to** $A.\text{length}$: $C[A[j]] = C[A[j]] + 1$

// $C[i]$ now contains the number of elements equal to i

for $i = 1$ **to** k: $C[i] = C[i] + C[i-1]$

// $C[i]$ now contains the number of elements less than or equal to i

for $j = A.\text{length}$ **downto** 1

$\quad B[C[A[j]]] = A[j]$; $C[A[j]] = C[A[j]] - 1$

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
A	5	3	10	5	4	5	7	7	9	3	10	8	5	3	3	8

	0	1	2	3	4	5	6	7	8	9	10
C	0	0	0	4	1	4	0	2	2	1	2

CountingSort

CountingSort(A, k)

// Input: array $A[1\colon n]$ of integers in the range $0 \dots k$

// Output: array $B[1\colon n]$ which contains the elements of A, sorted

for $i = 0$ **to** k: $C[i] = 0$

for $j = 1$ **to** $A.\mathrm{length}$: $C[A[j]] = C[A[j]] + 1$

// $C[i]$ now contains the number of elements equal to i

for $i = 1$ **to** k: $C[i] = C[i] + C[i-1]$

// $C[i]$ now contains the number of elements less than or equal to i

for $j = A.\mathrm{length}$ **downto** 1

$\quad B[C[A[j]]] = A[j]$; $C[A[j]] = C[A[j]] - 1$

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
A	5	3	10	5	4	5	7	7	9	3	10	8	5	3	3	8

	0	1	2	3	4	5	6	7	8	9	10
C	0	0	0	4	5	4	0	2	2	1	2

CountingSort

```
CountingSort(A, k)
    // Input: array A[1:n] of integers in the range 0 ... k
    // Output: array B[1:n] which contains the elements of A, sorted
for i = 0 to k:  C[i] = 0
for j = 1 to A.length:  C[A[j]] = C[A[j]] + 1
// C[i] now contains the number of elements equal to i
for i = 1 to k:  C[i] = C[i] + C[i − 1]
// C[i] now contains the number of elements less than or equal to i
for j = A.length downto 1
    B[C[A[j]]] = A[j];  C[A[j]] = C[A[j]] − 1
```

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
A	5	3	10	5	4	5	7	7	9	3	10	8	5	3	3	8

	0	1	2	3	4	5	6	7	8	9	10
C	0	0	0	4	5	9	0	2	2	1	2

CountingSort

```
CountingSort(A, k)
    // Input: array A[1: n] of integers in the range 0 … k
    // Output: array B[1: n] which contains the elements of A, sorted
for i = 0 to k:  C[i] = 0
for j = 1 to A.length:  C[A[j]] = C[A[j]] + 1
// C[i] now contains the number of elements equal to i
for i = 1 to k:  C[i] = C[i] + C[i − 1]
// C[i] now contains the number of elements less than or equal to i
for j = A.length downto 1
        B[C[A[j]]] = A[j];  C[A[j]] = C[A[j]] − 1
```

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
A	5	3	10	5	4	5	7	7	9	3	10	8	5	3	3	8

	0	1	2	3	4	5	6	7	8	9	10
C	0	0	0	4	5	9	9	2	2	1	2

CountingSort

CountingSort(A, k)

// Input: array $A[1{:}n]$ of integers in the range $0 \dots k$

// Output: array $B[1{:}n]$ which contains the elements of A, sorted

for $i = 0$ **to** k: $C[i] = 0$

for $j = 1$ **to** $A.\text{length}$: $C[A[j]] = C[A[j]] + 1$

// $C[i]$ now contains the number of elements equal to i

for $i = 1$ **to** k: $C[i] = C[i] + C[i-1]$

// $C[i]$ now contains the number of elements less than or equal to i

for $j = A.\text{length}$ **downto** 1

  $B[C[A[j]]] = A[j]$; $C[A[j]] = C[A[j]] - 1$

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
A	5	3	10	5	4	5	7	7	9	3	10	8	5	3	3	8

	0	1	2	3	4	5	6	7	8	9	10
C	0	0	0	4	5	9	9	11	2	1	2

CountingSort

CountingSort(A, k)

// Input: array $A[1{:}n]$ of integers in the range $0 \dots k$
// Output: array $B[1{:}n]$ which contains the elements of A, sorted
for $i = 0$ **to** k: $C[i] = 0$
for $j = 1$ **to** $A.\text{length}$: $C[A[j]] = C[A[j]] + 1$
// $C[i]$ now contains the number of elements equal to i
for $i = 1$ **to** k: $C[i] = C[i] + C[i-1]$
// $C[i]$ now contains the number of elements less than or equal to i
for $j = A.\text{length}$ **downto** 1
    $B[C[A[j]]] = A[j]$; $C[A[j]] = C[A[j]] - 1$

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
A	5	3	10	5	4	5	7	7	9	3	10	8	5	3	3	8

	0	1	2	3	4	5	6	7	8	9	10
C	0	0	0	4	5	9	9	11	13	1	2

CountingSort

CountingSort(A, k)

// Input: array $A[1:n]$ of integers in the range $0 \ldots k$

// Output: array $B[1:n]$ which contains the elements of A, sorted

for $i = 0$ **to** k: $C[i] = 0$

for $j = 1$ **to** $A.\text{length}$: $C[A[j]] = C[A[j]] + 1$

// $C[i]$ now contains the number of elements equal to i

for $i = 1$ **to** k: $C[i] = C[i] + C[i-1]$

// $C[i]$ now contains the number of elements less than or equal to i

for $j = A.\text{length}$ **downto** 1

$\quad B[C[A[j]]] = A[j]$; $C[A[j]] = C[A[j]] - 1$

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
A	5	3	10	5	4	5	7	7	9	3	10	8	5	3	3	8

	0	1	2	3	4	5	6	7	8	9	10
C	0	0	0	4	5	9	9	11	13	14	2

CountingSort

CountingSort(A, k)

// Input: array $A[1:n]$ of integers in the range $0 \ldots k$

// Output: array $B[1:n]$ which contains the elements of A, sorted

for $i = 0$ **to** k: $C[i] = 0$

for $j = 1$ **to** $A.\text{length}$: $C[A[j]] = C[A[j]] + 1$

// $C[i]$ now contains the number of elements equal to i

for $i = 1$ **to** k: $C[i] = C[i] + C[i-1]$

// $C[i]$ now contains the number of elements less than or equal to i

for $j = A.\text{length}$ **downto** 1

$\quad B[C[A[j]]] = A[j]$; $C[A[j]] = C[A[j]] - 1$

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
A	5	3	10	5	4	5	7	7	9	3	10	8	5	3	3	8

	0	1	2	3	4	5	6	7	8	9	10
C	0	0	0	4	5	9	9	11	13	14	16

CountingSort

CountingSort(A, k)

// Input: array $A[1{:}n]$ of integers in the range $0 \ldots k$

// Output: array $B[1{:}n]$ which contains the elements of A, sorted

for $i = 0$ **to** k: $C[i] = 0$

for $j = 1$ **to** $A.\text{length}$: $C[A[j]] = C[A[j]] + 1$

// $C[i]$ now contains the number of elements equal to i

for $i = 1$ **to** k: $C[i] = C[i] + C[i-1]$

// $C[i]$ now contains the number of elements less than or equal to i

→ **for** $j = A.\text{length}$ **downto** 1

  $B[C[A[j]]] = A[j]$; $C[A[j]] = C[A[j]] - 1$

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
A	5	3	10	5	4	5	7	7	9	3	10	8	5	3	3	8

	0	1	2	3	4	5	6	7	8	9	10
C	0	0	0	4	5	9	9	11	13	14	16

CountingSort

CountingSort(A, k)

// Input: array $A[1{:}n]$ of integers in the range $0 \dots k$

// Output: array $B[1{:}n]$ which contains the elements of A, sorted

for $i = 0$ **to** k: $C[i] = 0$

for $j = 1$ **to** $A.\text{length}$: $C[A[j]] = C[A[j]] + 1$

// $C[i]$ now contains the number of elements equal to i

for $i = 1$ **to** k: $C[i] = C[i] + C[i-1]$

// $C[i]$ now contains the number of elements less than or equal to i

→ **for** $j = A.\text{length}$ **downto** 1

$\quad B[C[A[j]]] = A[j]$; $C[A[j]] = C[A[j]] - 1$

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
A	5	3	10	5	4	5	7	7	9	3	10	8	5	3	3	8

	0	1	2	3	4	5	6	7	8	9	10
C	0	0	0	4	5	9	9	11	13	14	16

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
B																

CountingSort

CountingSort(A, k)
   // Input: array $A[1{:}n]$ of integers in the range $0 \dots k$
   // Output: array $B[1{:}n]$ which contains the elements of A, sorted
for $i = 0$ **to** k: $C[i] = 0$
for $j = 1$ **to** $A.\text{length}$: $C[A[j]] = C[A[j]] + 1$
// $C[i]$ now contains the number of elements equal to i
for $i = 1$ **to** k: $C[i] = C[i] + C[i-1]$
// $C[i]$ now contains the number of elements less than or equal to i
for $j = A.\text{length}$ **downto** 1
      $B[C[A[j]]] = A[j]$; $C[A[j]] = C[A[j]] - 1$

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
A	5	3	10	5	4	5	7	7	9	3	10	8	5	3	3	8

	0	1	2	3	4	5	6	7	8	9	10
C	0	0	0	4	5	9	9	11	13	14	16

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
B																

CountingSort

CountingSort(A, k)
    // *Input: array $A[1{:}n]$ of integers in the range $0 \ldots k$*
    // *Output: array $B[1{:}n]$ which contains the elements of A, sorted*
for $i = 0$ **to** k: $C[i] = 0$
for $j = 1$ **to** $A.\mathrm{length}$: $C[A[j]] = C[A[j]] + 1$
// $C[i]$ now contains the number of elements equal to i
for $i = 1$ **to** k: $C[i] = C[i] + C[i-1]$
// $C[i]$ now contains the number of elements less than or equal to i
for $j = A.\mathrm{length}$ **downto** 1
    $B[C[A[j]]] = A[j]$; $C[A[j]] = C[A[j]] - 1$

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
A	5	3	10	5	4	5	7	7	9	3	10	8	5	3	3	8

	0	1	2	3	4	5	6	7	8	9	10
C	0	0	0	4	5	9	9	11	13	14	16

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
B																

CountingSort

CountingSort(A, k)
    // *Input: array* $A[1{:}n]$ *of integers in the range* $0 \ldots k$
    // *Output: array* $B[1{:}n]$ *which contains the elements of* A*, sorted*
for $i = 0$ **to** k: $C[i] = 0$
for $j = 1$ **to** $A.\text{length}$: $C[A[j]] = C[A[j]] + 1$
// $C[i]$ now contains the number of elements equal to i
for $i = 1$ **to** k: $C[i] = C[i] + C[i-1]$
// $C[i]$ now contains the number of elements less than or equal to i
→ **for** $j = A.\text{length}$ **downto** 1
        $B[C[A[j]]] = A[j]$; $C[A[j]] = C[A[j]] - 1$

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
A	5	3	10	5	4	5	7	7	9	3	10	8	5	3	3	8

	0	1	2	3	4	5	6	7	8	9	10
C	0	0	0	4	5	9	9	11	13	14	16

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
B													8			

CountingSort

CountingSort(A, k)

  // Input: array $A[1{:}n]$ of integers in the range $0 \dots k$
  // Output: array $B[1{:}n]$ which contains the elements of A, sorted

for $i = 0$ **to** k: $C[i] = 0$
for $j = 1$ **to** $A.\text{length}$: $C[A[j]] = C[A[j]] + 1$
// $C[i]$ now contains the number of elements equal to i
for $i = 1$ **to** k: $C[i] = C[i] + C[i-1]$
// $C[i]$ now contains the number of elements less than or equal to i
for $j = A.\text{length}$ **downto** 1
  $B[C[A[j]]] = A[j]$; $C[A[j]] = C[A[j]] - 1$

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
A	5	3	10	5	4	5	7	7	9	3	10	8	5	3	3	8

	0	1	2	3	4	5	6	7	8	9	10
C	0	0	0	4	5	9	9	11	12	14	16

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
B													8			

CountingSort

CountingSort(A, k)
// *Input: array $A[1{:}n]$ of integers in the range $0 \ldots k$*
// *Output: array $B[1{:}n]$ which contains the elements of A, sorted*
for $i = 0$ **to** k: $C[i] = 0$
for $j = 1$ **to** $A.\text{length}$: $C[A[j]] = C[A[j]] + 1$
// $C[i]$ now contains the number of elements equal to i
for $i = 1$ **to** k: $C[i] = C[i] + C[i-1]$
// $C[i]$ now contains the number of elements less than or equal to i
for $j = A.\text{length}$ **downto** 1
    $B[C[A[j]]] = A[j]$; $C[A[j]] = C[A[j]] - 1$

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
A	5	3	10	5	4	5	7	7	9	3	10	8	5	3	3	8

	0	1	2	3	4	5	6	7	8	9	10
C	0	0	0	4	5	9	9	11	12	14	16

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
B													8			

CountingSort

CountingSort(A, k)

// *Input: array* $A[1{:}n]$ *of integers in the range* $0 \ldots k$

// *Output: array* $B[1{:}n]$ *which contains the elements of* A*, sorted*

for $i = 0$ **to** k: $C[i] = 0$

for $j = 1$ **to** $A.\text{length}$: $C[A[j]] = C[A[j]] + 1$

// $C[i]$ now contains the number of elements equal to i

for $i = 1$ **to** k: $C[i] = C[i] + C[i-1]$

// $C[i]$ now contains the number of elements less than or equal to i

→ **for** $j = A.\text{length}$ **downto** 1

$\quad B[C[A[j]]] = A[j]$; $C[A[j]] = C[A[j]] - 1$

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
A	5	3	10	5	4	5	7	7	9	3	10	8	5	3	3	8

	0	1	2	3	4	5	6	7	8	9	10
C	0	0	0	4	5	9	9	11	12	14	16

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
B				3									8			

CountingSort

CountingSort(A, k)

// Input: array $A[1{:}n]$ of integers in the range $0 \ldots k$

// Output: array $B[1{:}n]$ which contains the elements of A, sorted

for $i = 0$ **to** k: $C[i] = 0$

for $j = 1$ **to** $A.\text{length}$: $C[A[j]] = C[A[j]] + 1$

// $C[i]$ now contains the number of elements equal to i

for $i = 1$ **to** k: $C[i] = C[i] + C[i-1]$

// $C[i]$ now contains the number of elements less than or equal to i

for $j = A.\text{length}$ **downto** 1

  $B[C[A[j]]] = A[j]$; $C[A[j]] = C[A[j]] - 1$

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
A	5	3	10	5	4	5	7	7	9	3	10	8	5	3	3	8

	0	1	2	3	4	5	6	7	8	9	10
C	0	0	0	3	5	9	9	11	12	14	16

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
B				3									8			

CountingSort

CountingSort(A, k)

// Input: array $A[1{:}n]$ of integers in the range $0 \dots k$
// Output: array $B[1{:}n]$ which contains the elements of A, sorted

for $i = 0$ **to** k: $C[i] = 0$
for $j = 1$ **to** $A.\text{length}$: $C[A[j]] = C[A[j]] + 1$
// $C[i]$ now contains the number of elements equal to i
for $i = 1$ **to** k: $C[i] = C[i] + C[i-1]$
// $C[i]$ now contains the number of elements less than or equal to i
for $j = A.\text{length}$ **downto** 1
    $B[C[A[j]]] = A[j]$; $C[A[j]] = C[A[j]] - 1$

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
A	5	3	10	5	4	5	7	7	9	3	10	8	5	3	3	8

	0	1	2	3	4	5	6	7	8	9	10
C	0	0	0	3	5	9	9	11	12	14	16

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
B				3									8			

CountingSort

CountingSort(A, k)
  // *Input: array* $A[1{:}n]$ *of integers in the range* $0 \dots k$
  // *Output: array* $B[1{:}n]$ *which contains the elements of* A*, sorted*
for $i = 0$ **to** k: $C[i] = 0$
for $j = 1$ **to** $A.\text{length}$: $C[A[j]] = C[A[j]] + 1$
// $C[i]$ now contains the number of elements equal to i
for $i = 1$ **to** k: $C[i] = C[i] + C[i-1]$
// $C[i]$ now contains the number of elements less than or equal to i
for $j = A.\text{length}$ **downto** 1
  $B[C[A[j]]] = A[j]$; $C[A[j]] = C[A[j]] - 1$

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
A	5	3	10	5	4	5	7	7	9	3	10	8	5	3	3	8

	0	1	2	3	4	5	6	7	8	9	10
C	0	0	0	3	5	9	9	11	12	14	16

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
B			3	3									8			

CountingSort

CountingSort(A, k)

// Input: array $A[1{:}n]$ of integers in the range $0 \ldots k$

// Output: array $B[1{:}n]$ which contains the elements of A, sorted

for $i = 0$ **to** k: $C[i] = 0$

for $j = 1$ **to** $A.\mathrm{length}$: $C[A[j]] = C[A[j]] + 1$

// $C[i]$ now contains the number of elements equal to i

for $i = 1$ **to** k: $C[i] = C[i] + C[i-1]$

// $C[i]$ now contains the number of elements less than or equal to i

for $j = A.\mathrm{length}$ **downto** 1

$\quad B[C[A[j]]] = A[j]$; $C[A[j]] = C[A[j]] - 1$

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
A	5	3	10	5	4	5	7	7	9	3	10	8	5	3	3	8

	0	1	2	3	4	5	6	7	8	9	10
C	0	0	0	2	5	9	9	11	12	14	16

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
B			3	3									8			

CountingSort

CountingSort(A, k)

// Input: array $A[1{:}n]$ of integers in the range $0 \ldots k$

// Output: array $B[1{:}n]$ which contains the elements of A, sorted

for $i = 0$ **to** k: $C[i] = 0$

for $j = 1$ **to** $A.\text{length}$: $C[A[j]] = C[A[j]] + 1$

// $C[i]$ now contains the number of elements equal to i

for $i = 1$ **to** k: $C[i] = C[i] + C[i-1]$

// $C[i]$ now contains the number of elements less than or equal to i

→ **for** $j = A.\text{length}$ **downto** 1

  $B[C[A[j]]] = A[j]$; $C[A[j]] = C[A[j]] - 1$

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
A	5	3	10	5	4	5	7	7	9	3	10	8	5	3	3	8

	0	1	2	3	4	5	6	7	8	9	10
C	0	0	0	0	4	5	9	9	11	13	14

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
B	3	3	3	3	4	5	5	5	5	7	7	8	8	9	10	10

CountingSort

CountingSort(A, k)

// *Input: array* $A[1{:}n]$ *of integers in the range* $0 \ldots k$

// *Output: array* $B[1{:}n]$ *which contains the elements of* A*, sorted*

1 **for** $i = 0$ **to** k: $C[i] = 0$

2 **for** $j = 1$ **to** $A.\mathrm{length}$: $C[A[j]] = C[A[j]] + 1$

3 // $C[i]$ now contains the number of elements equal to i

4 **for** $i = 1$ **to** k: $C[i] = C[i] + C[i-1]$

5 // $C[i]$ now contains the number of elements less than or equal to i

6 **for** $j = A.\mathrm{length}$ **downto** 1

7 $\quad B[C[A[j]]] = A[j]$; $C[A[j]] = C[A[j]] - 1$

Correctness Lines 6/7: Invariant

$\mathrm{Inv}(j)$: for $j + 1 \le i \le n$: $B[\mathrm{position}(i)]$ contains $A[i]$

for $0 \le i \le k$: $C[i] =$ (# numbers smaller than i) + (# numbers equal to i in $A[1{:}j]$)

$\mathrm{Inv}(j)$ holds before loop is executed, $\mathrm{Inv}(j-1)$ holds afterwards

CountingSort: running time

CountingSort(A, k)

// Input: array $A[1{:}n]$ of integers in the range $0 \ldots k$

// Output: array $B[1{:}n]$ which contains the elements of A, sorted

1 **for** $i = 0$ **to** k: $C[i] = 0$

2 **for** $j = 1$ **to** $A.\text{length}$: $C[A[j]] = C[A[j]] + 1$

3 // $C[i]$ now contains the number of elements equal to i

4 **for** $i = 1$ **to** k: $C[i] = C[i] + C[i-1]$

5 // $C[i]$ now contains the number of elements less than or equal to i

6 **for** $j = A.\text{length}$ **downto** 1

7 $\quad B[C[A[j]]] = A[j];\ C[A[j]] = C[A[j]] - 1$

Line 1: $\sum_{0 \le i \le k} \Theta(1) = \Theta(k)$

Line 2: $\sum_{0 \le i \le n} \Theta(1) = \Theta(n)$

Line 4: $\sum_{0 \le i \le k} \Theta(1) = \Theta(k)$

Lines 6/7: $\sum_{0 \le i \le n} \Theta(1) = \Theta(n)$

$\boxed{\text{Total: } \Theta(n+k) \Rightarrow \Theta(n) \text{ if } k = O(n)}$

CountingSort

Theorem

CountingSort is a stable sorting algorithm that sorts an array of n integers in the range $0 \dots k$ in $\Theta(n + k)$ time.

RadixSort

Input: array $A[1..n]$ of numbers

Assumption: the input elements are integers with d digits

example ($d = 4$): 3288, 1193, 9999, 0654, 7243, 4321

d^{th} digit　　1^{st} digit

RadixSort(A, d)

for $i = 1$ **to** d

  use a stable sort to sort array A on digit i

RadixSort: example

329	720	720	329
457	355	329	355
657	436	436	436
839	457	839	457
436	657	355	657
720	329	457	720
355	839	657	839

sort on 1st digit | sort on 2nd digit | sort on 3rd digit

Correctness: Practice set

RadixSort

Running time: If we use CountingSort as stable sorting algorithm

➡ $\Theta(n + k)$ per digit

each digit is an integer in the range $0 \dots k$

Theorem

Given n d-digit numbers in which each digit can take up to k possible values, RadixSort correctly sorts these numbers in $\Theta(d(n+k))$ time.

BucketSort

Input: array $A[1{:}n]$ of numbers
Assumption: the input elements lie in the interval $[0 \dots 1)$ (no integers!)

BucketSort is fast if the elements are uniformly distributed in $[0 \dots 1)$

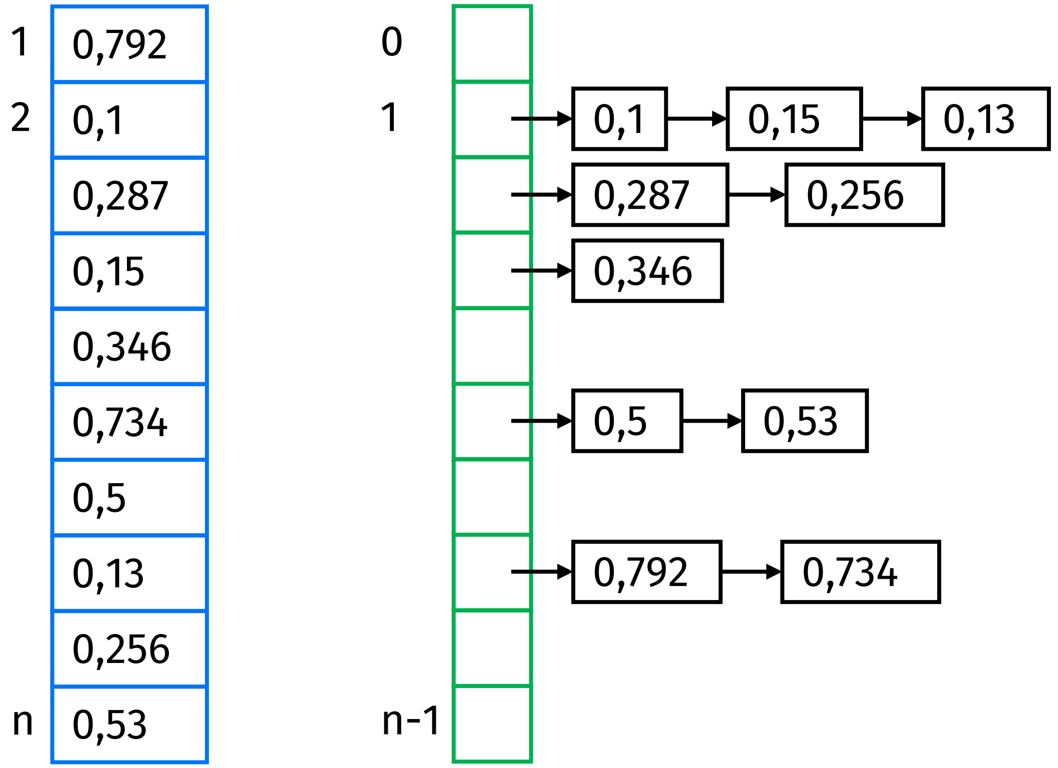
BucketSort

Throw input elements in “buckets”, sort buckets, concatenate ...

input array $A[1{:}n]$;
numbers in $[0 \dots 1)$

auxiliary array $B[0{:}n-1]$

bucket $B[i]$ contains numbers in $[i/n \dots (i+1)/n]$



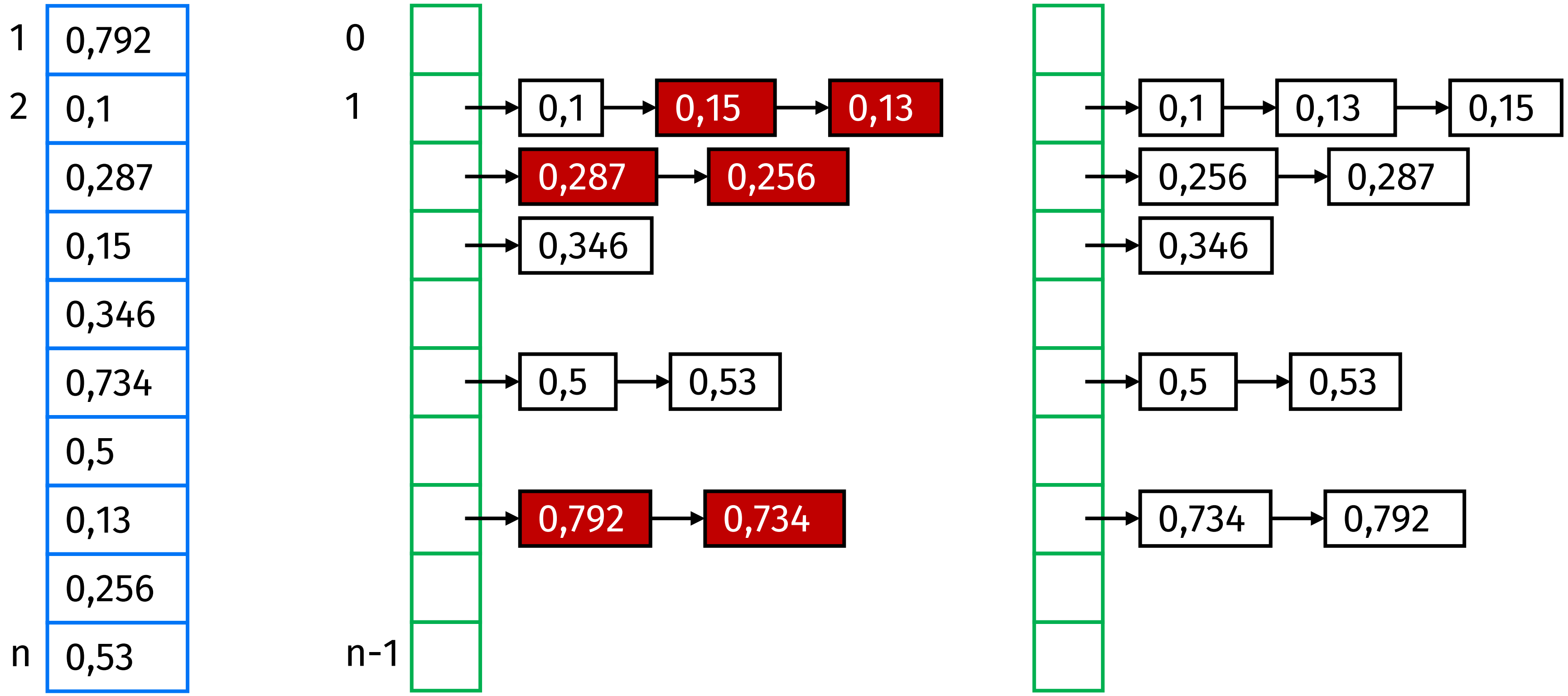
BucketSort

Throw input elements in “buckets”, sort buckets, concatenate ...

input array $A[1{:}n]$;
numbers in $[0 \ldots 1)$

auxiliary array $B[0{:}n-1]$

bucket $B[i]$ contains numbers in $[i/n \ldots (i+1)/n)$



BucketSort

BucketSort(A)

// Input: array $A[1{:}n]$ of numbers with $0 \leq A[i] < 1$

// Output: sorted list, which contains the elements of A

$n = A.\mathrm{length}$

initialize auxiliary array $B[0{:}n-1]$; each $B[i]$ is a linked list of numbers

for $i = 1$ **to** n

    insert $A[i]$ into list $B[\lfloor n \cdot A[i] \rfloor]$

for $i = 0$ **to** $n - 1$

    sort list $B[i]$, for example with InsertionSort

concatenate the lists $B[0], B[1], \ldots, B[n-1]$ together in order

BucketSort

Running time?

Define n_i = number of elements in bucket $B[i]$

➡ running time $= \Theta(n) + \sum_{0 \le i \le n-1} \Theta(n_i^2)$

worst case: all numbers fall into the same bucket ➡ $\Theta(n^2)$

best case: all numbers fall into different buckets ➡ $\Theta(n)$

expected running time if the numbers are randomly distributed?

BucketSort: expected running time

Define n_i = number of elements in bucket $B[i]$

➡ running time $= \Theta(n) + \sum_{0 \le i \le n-1} \Theta(n_i^2)$

Assumption: $\Pr\{A[j] \text{ falls in bucket } B[i]\} = 1/n$ for each i

$$
\begin{aligned}
E[\text{running time}] &= E\left[\Theta(n) + \textstyle\sum_{0 \le i \le n-1} \Theta(n_i^2)\right] \\
&= \Theta\left(n + \textstyle\sum_{0 \le i \le n-1} E[n_i^2]\right)
\end{aligned}
$$

What is $E[n_i^2]$? We have $E[n_i] = 1$ … but $E\left[n_i^2\right] \neq E[n_i]^2$

(*some math with indicator random variables – see book for details*)

➡ $E\left[n_i^2\right] = 2 - 1/n = \Theta(1)$

➡ expected running time = $\Theta(n)$

Linear time sorting

Sorting in linear time

Only if assumptions hold!

CountingSort

Assumption: input elements are integers in the range 0 to k

Running time: $\Theta(n + k)$ ➡ $\Theta(n)$ if $k = O(n)$

RadixSort

Assumption: input elements are integers with d digits

Running time: $\Theta(d(n + k))$

Can be $\Theta(n)$ for bounded integers with good choice of base

BucketSort

Assumption: input elements lie in the interval $[0 \ldots 1)$

Expected $\Theta(n)$ for uniform input, not for worst case!