# Algorithms: Linear Time Sorting?

Ola Svensson

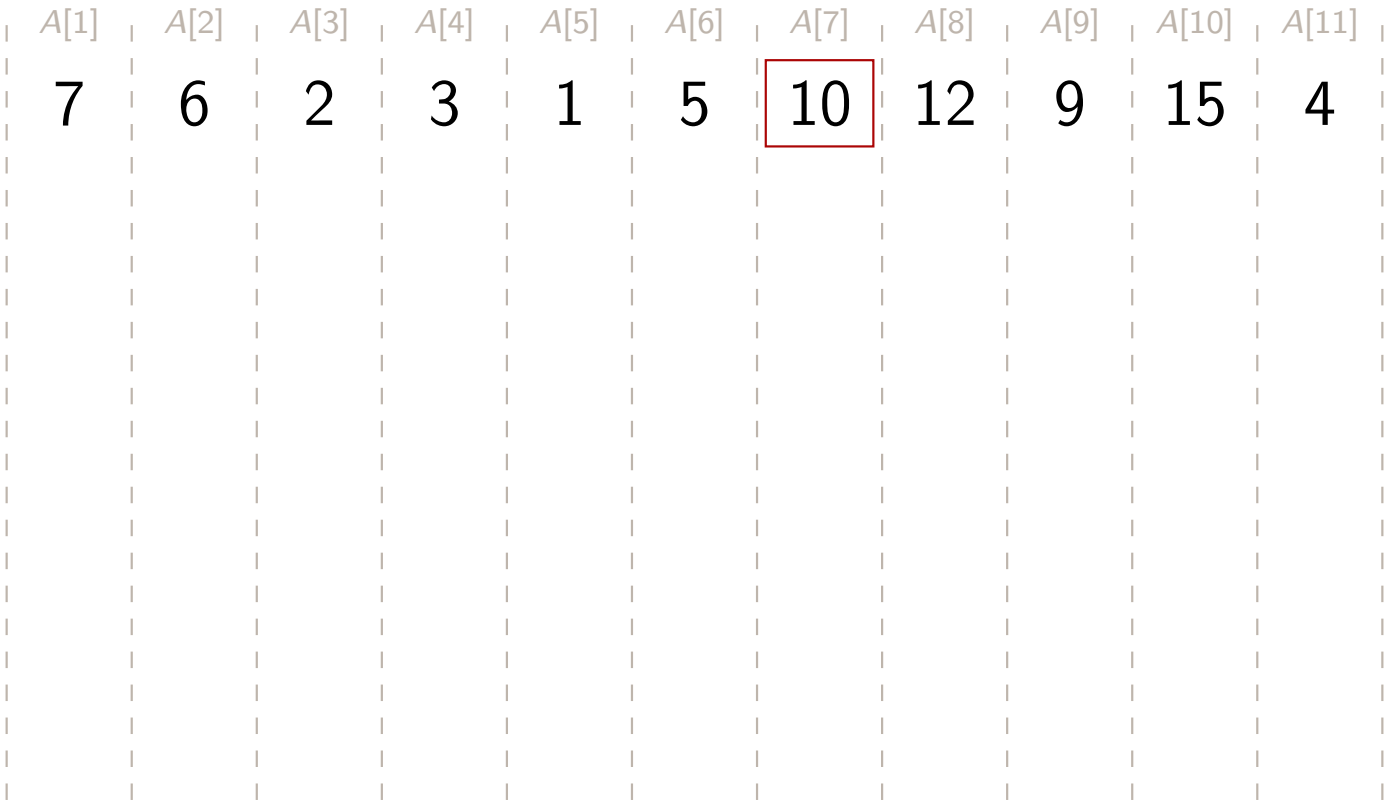
EPFL School of Computer and Communication Sciences

Lecture 25, 20.05.2025

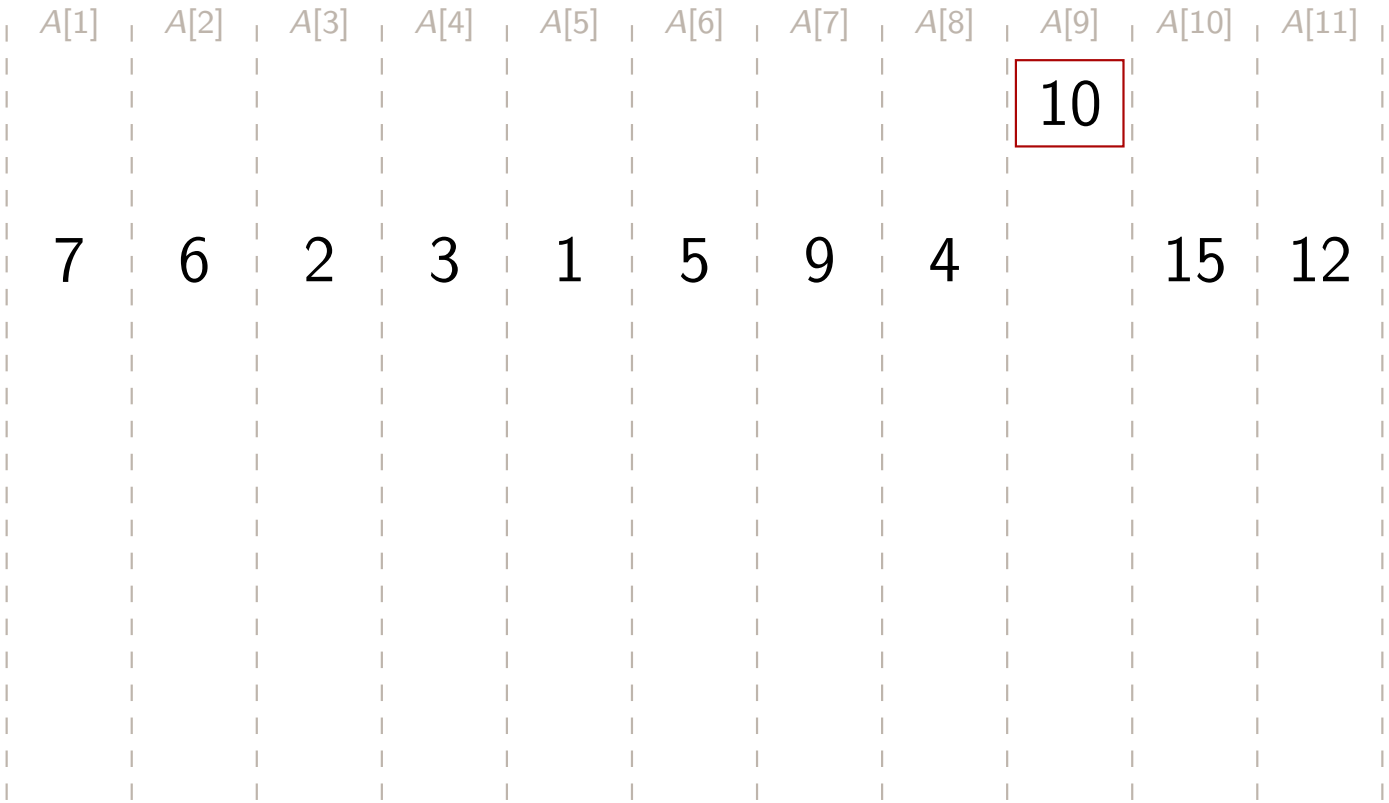
# Recall: Quick Sort

| $A[1]$ | $A[2]$ | $A[3]$ | $A[4]$ | $A[5]$ | $A[6]$ | $A[7]$ | $A[8]$ | $A[9]$ | $A[10]$ | $A[11]$ |
|---|---|---|---|---|---|---|---|---|---|---|
| 7 | 6 | 2 | 3 | 1 | 5 | 10 | 12 | 9 | 15 | 4 |

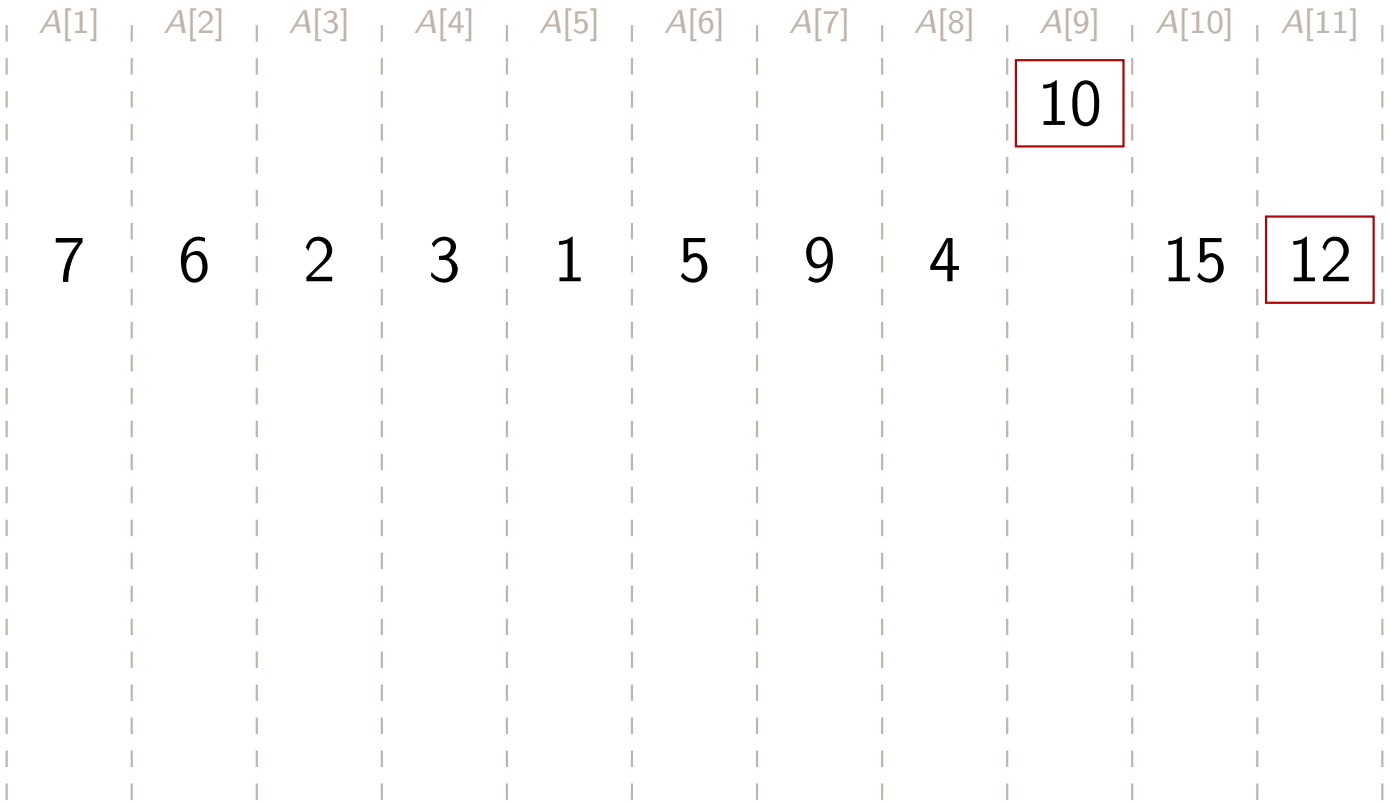
# Recall: Quick Sort



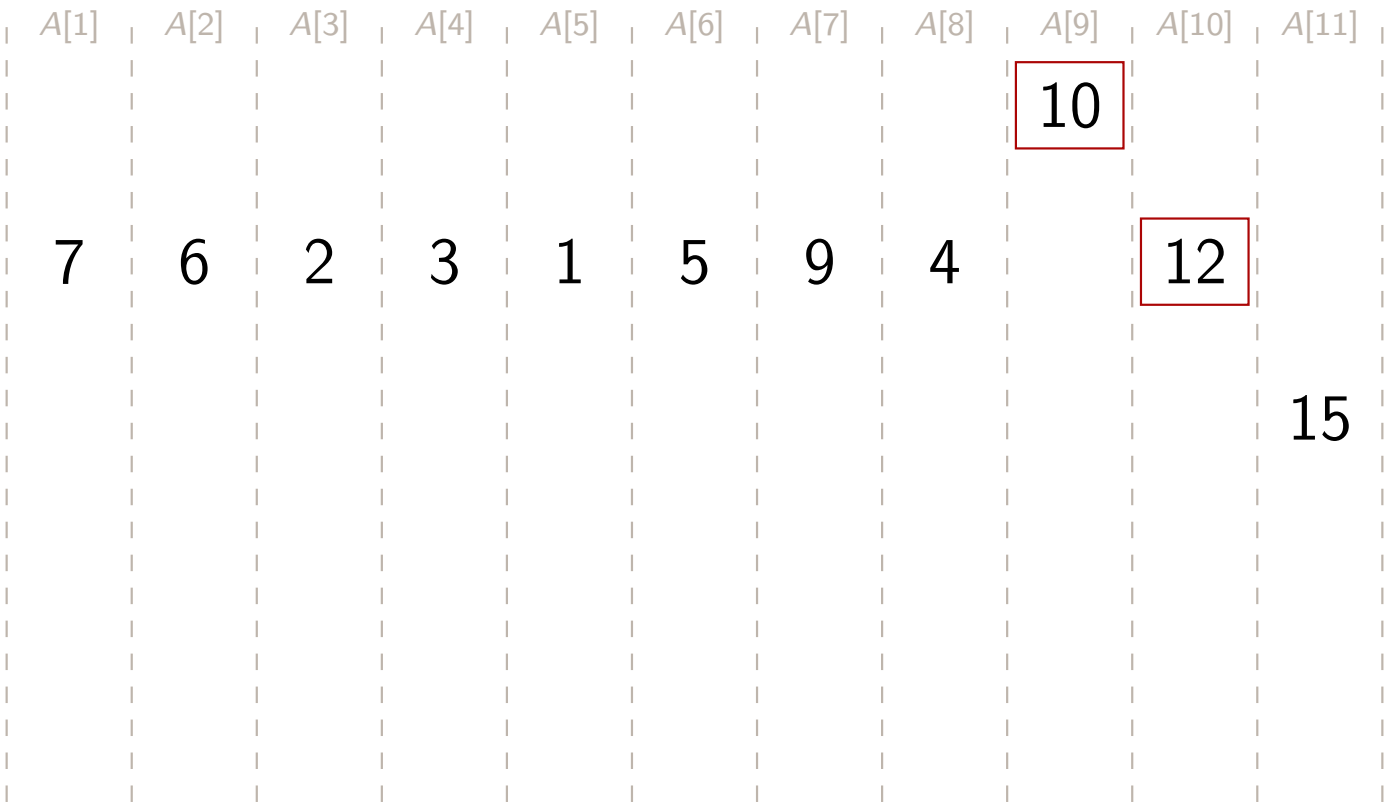
# Recall: Quick Sort



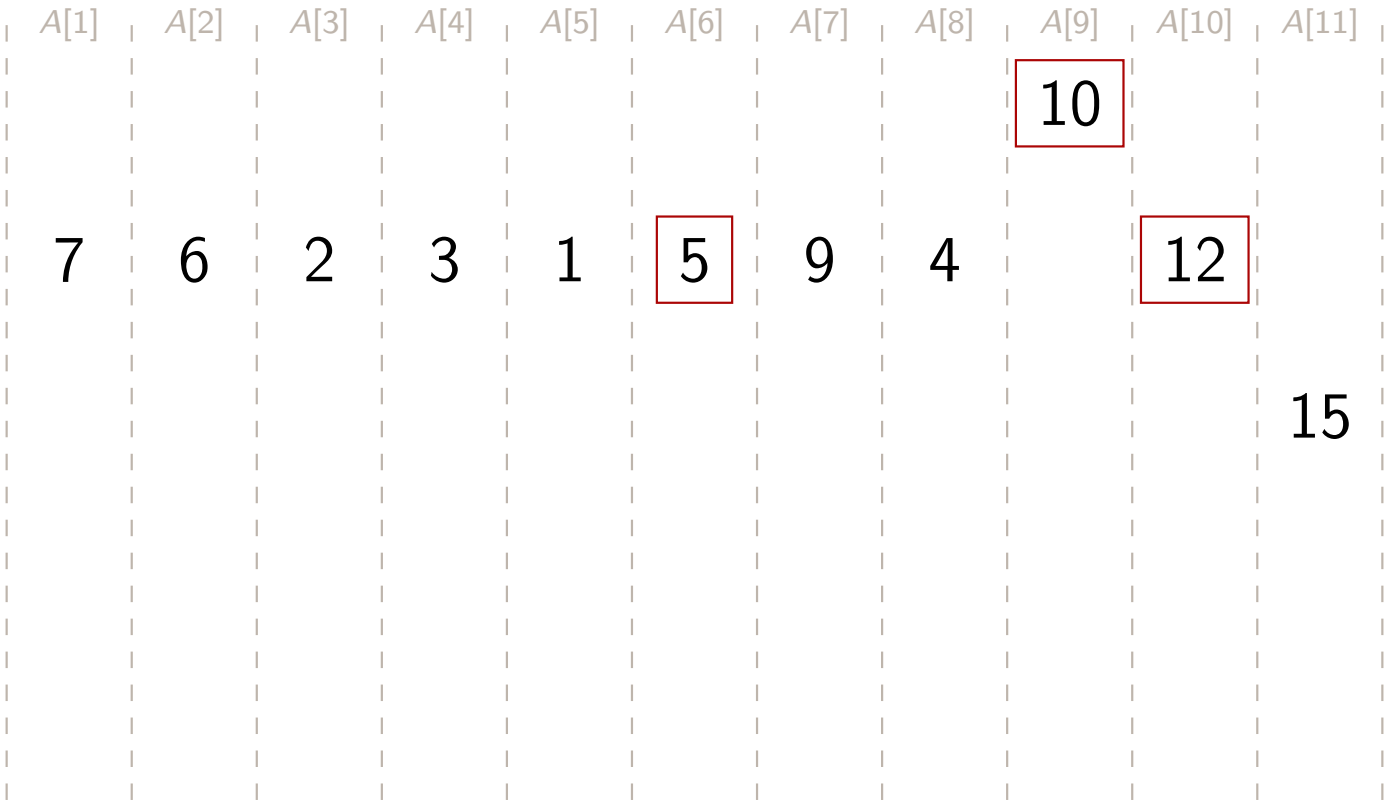
# Recall: Quick Sort



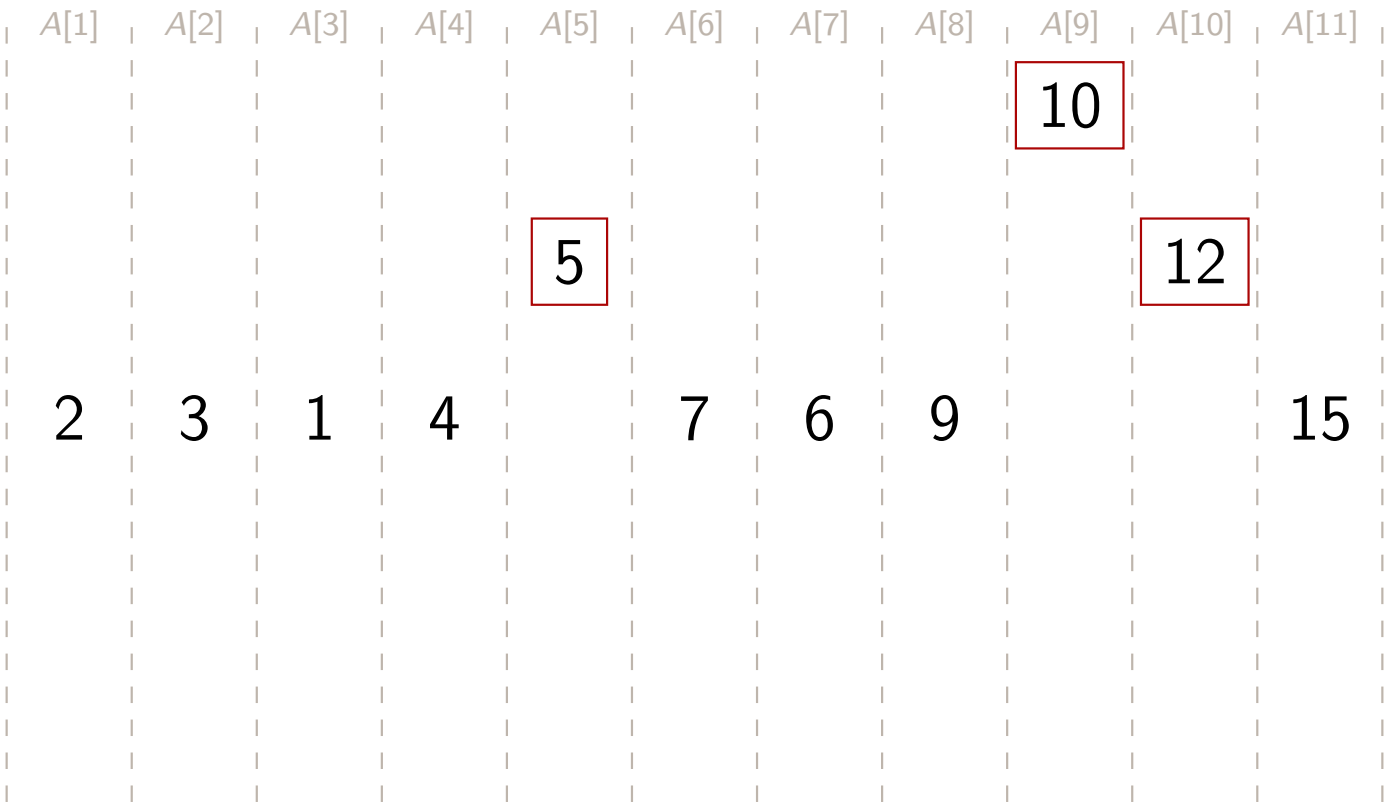
# Recall: Quick Sort



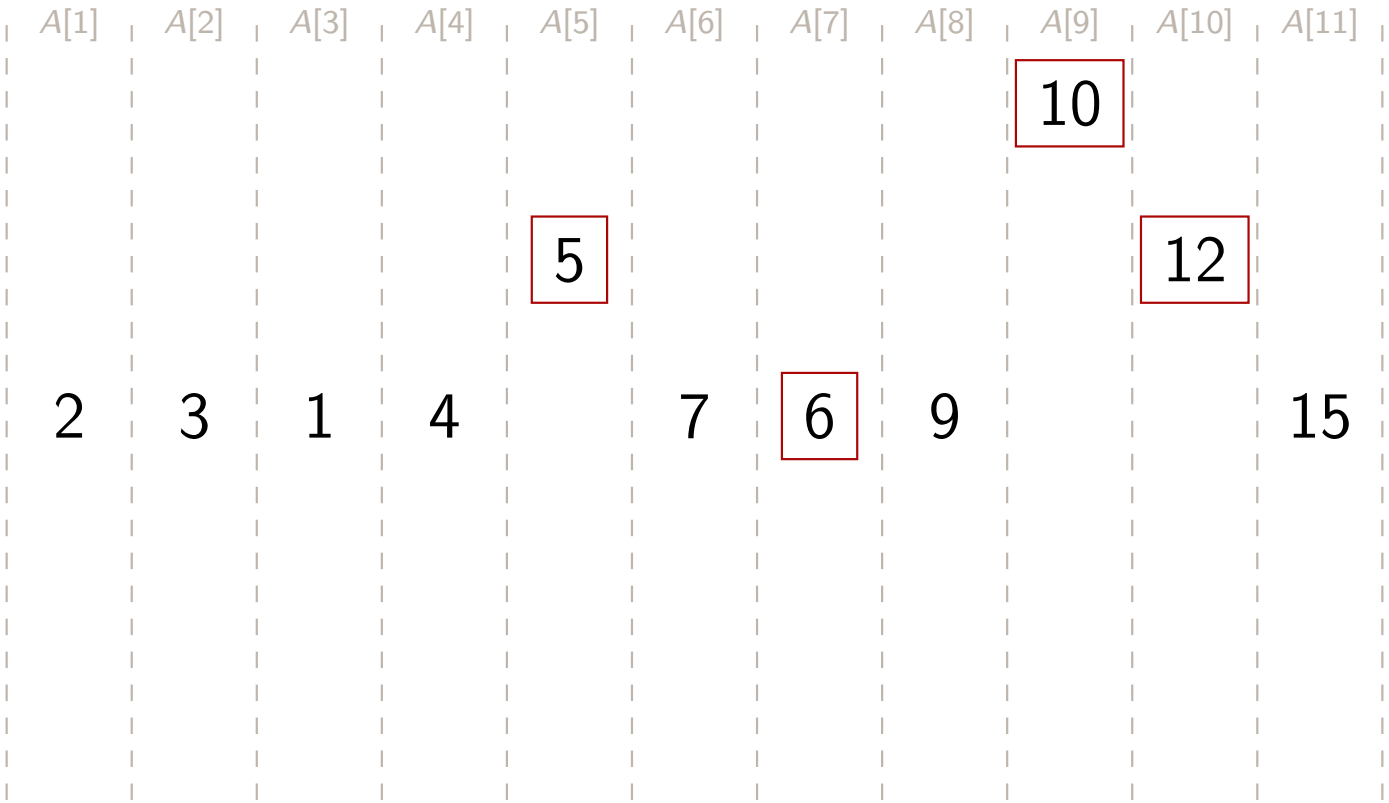
# Recall: Quick Sort



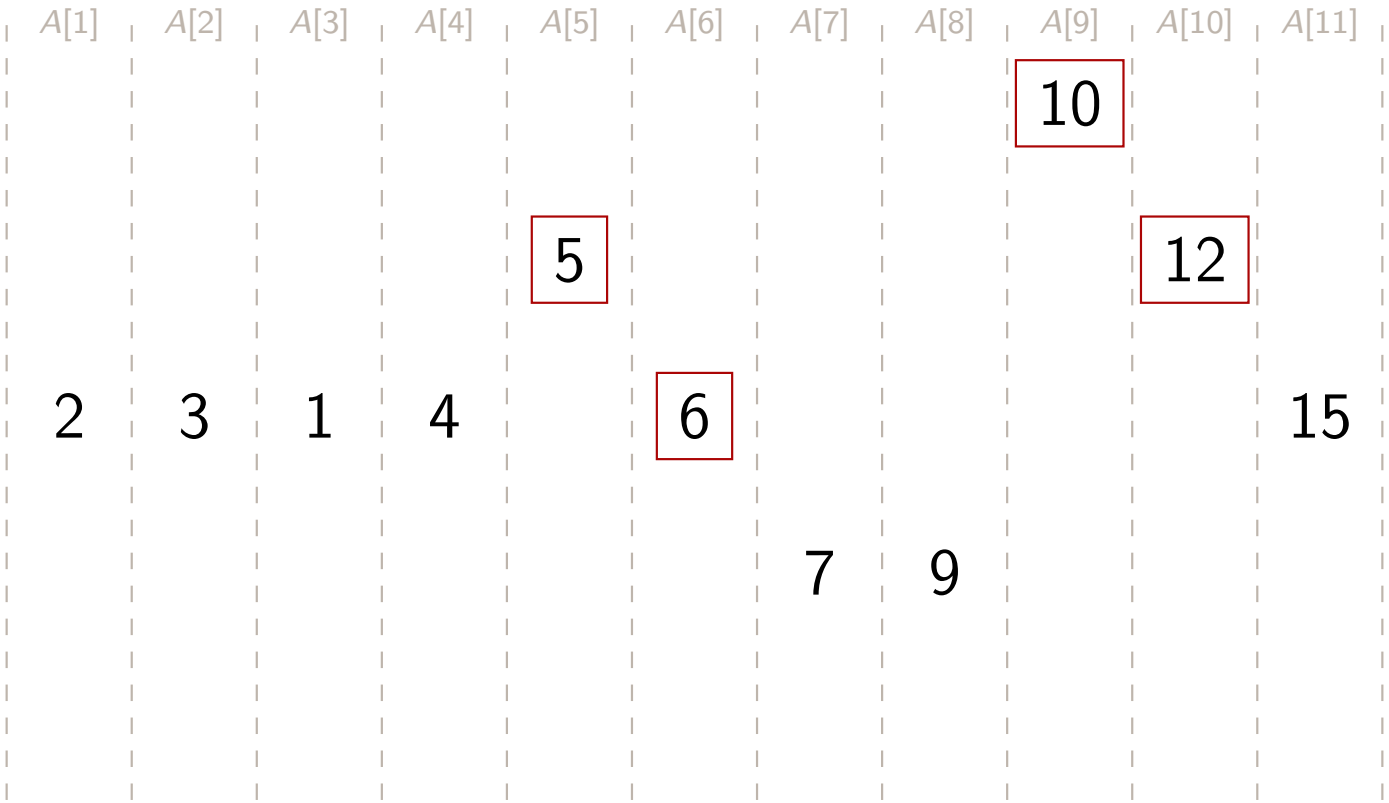
# Recall: Quick Sort



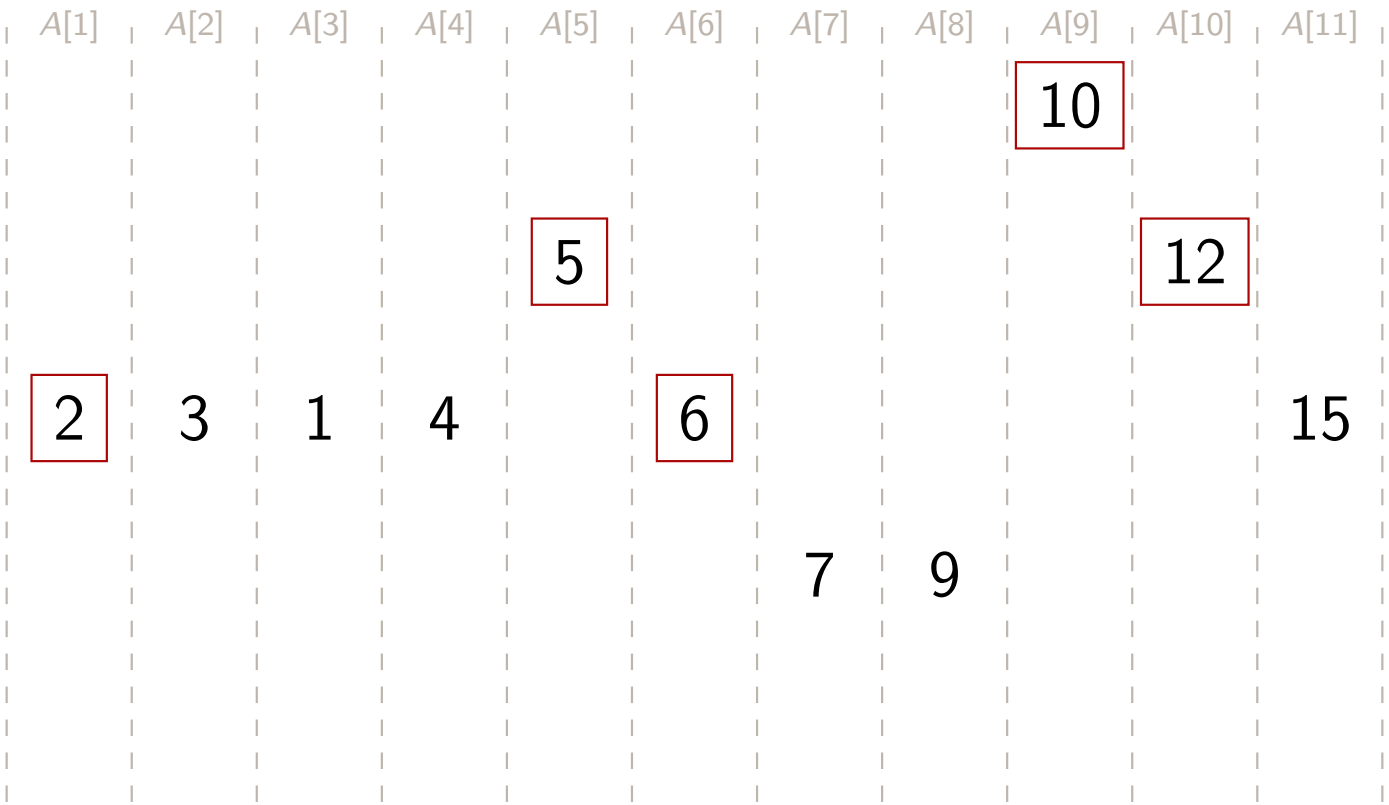
# Recall: Quick Sort



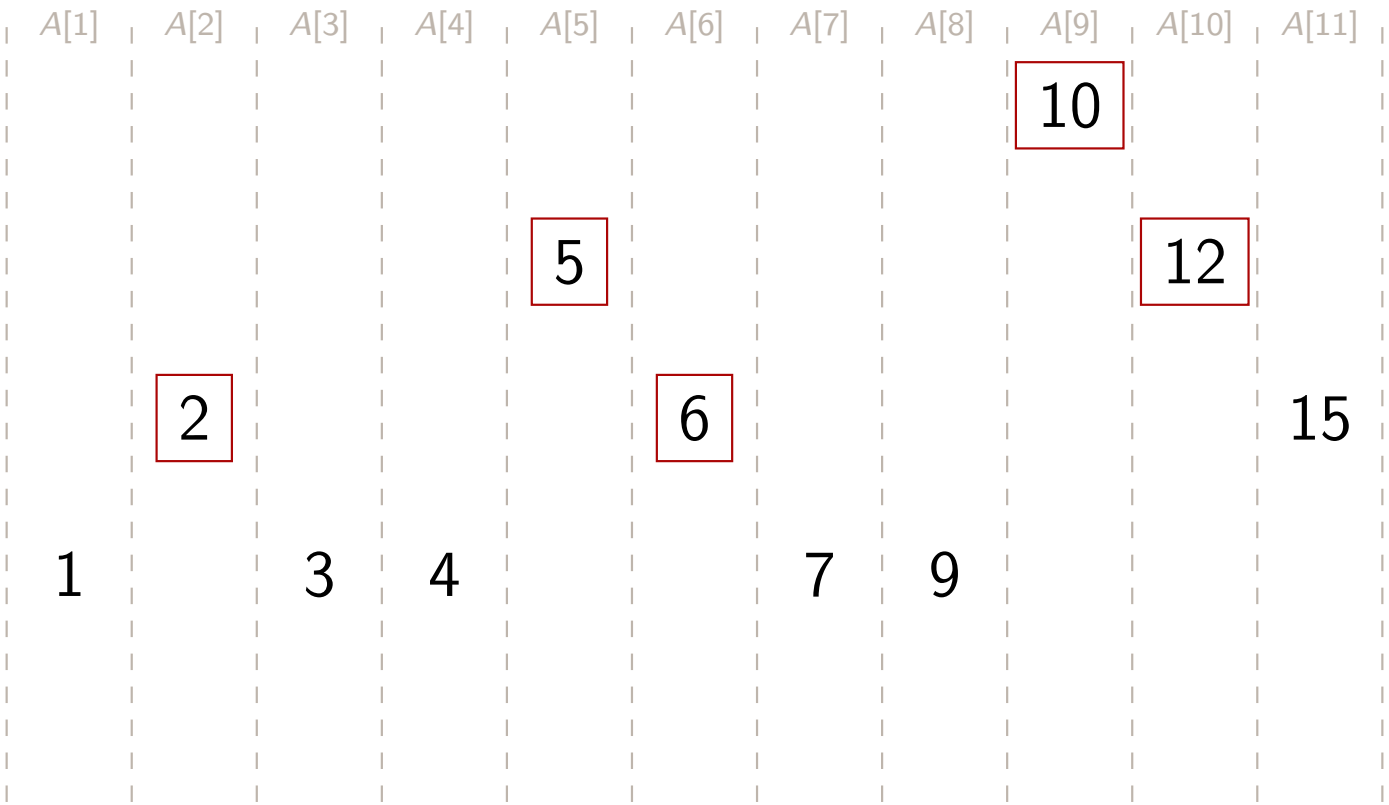
# Recall: Quick Sort



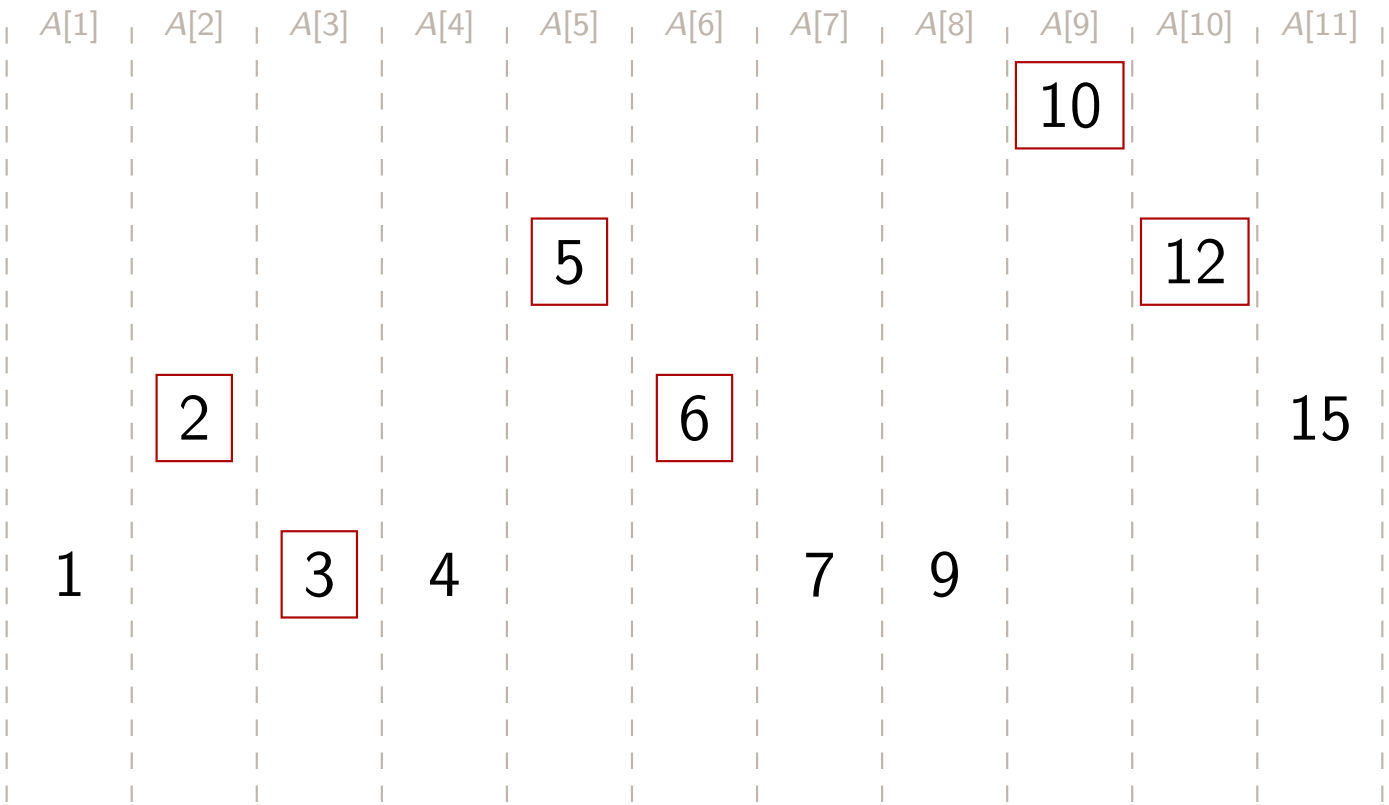
# Recall: Quick Sort



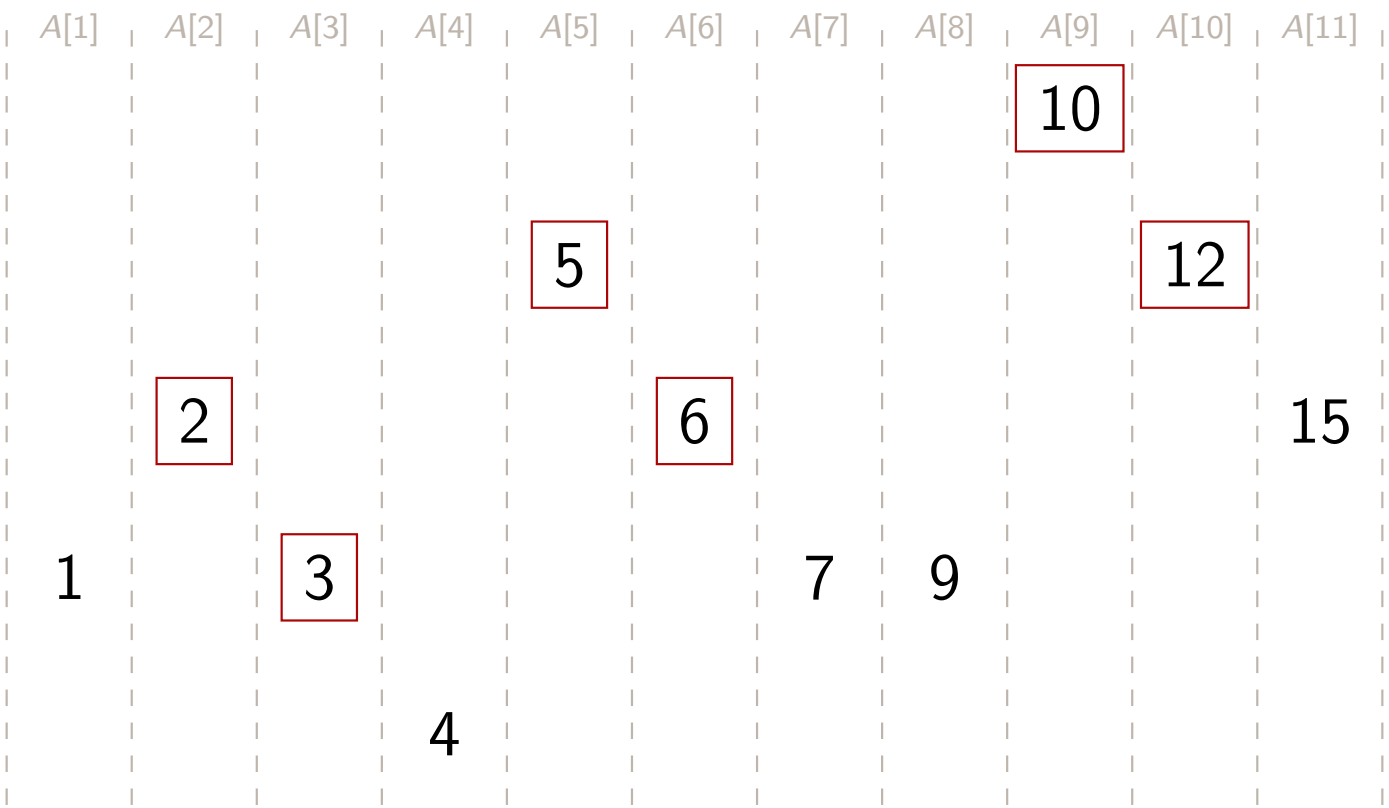
# Recall: Quick Sort



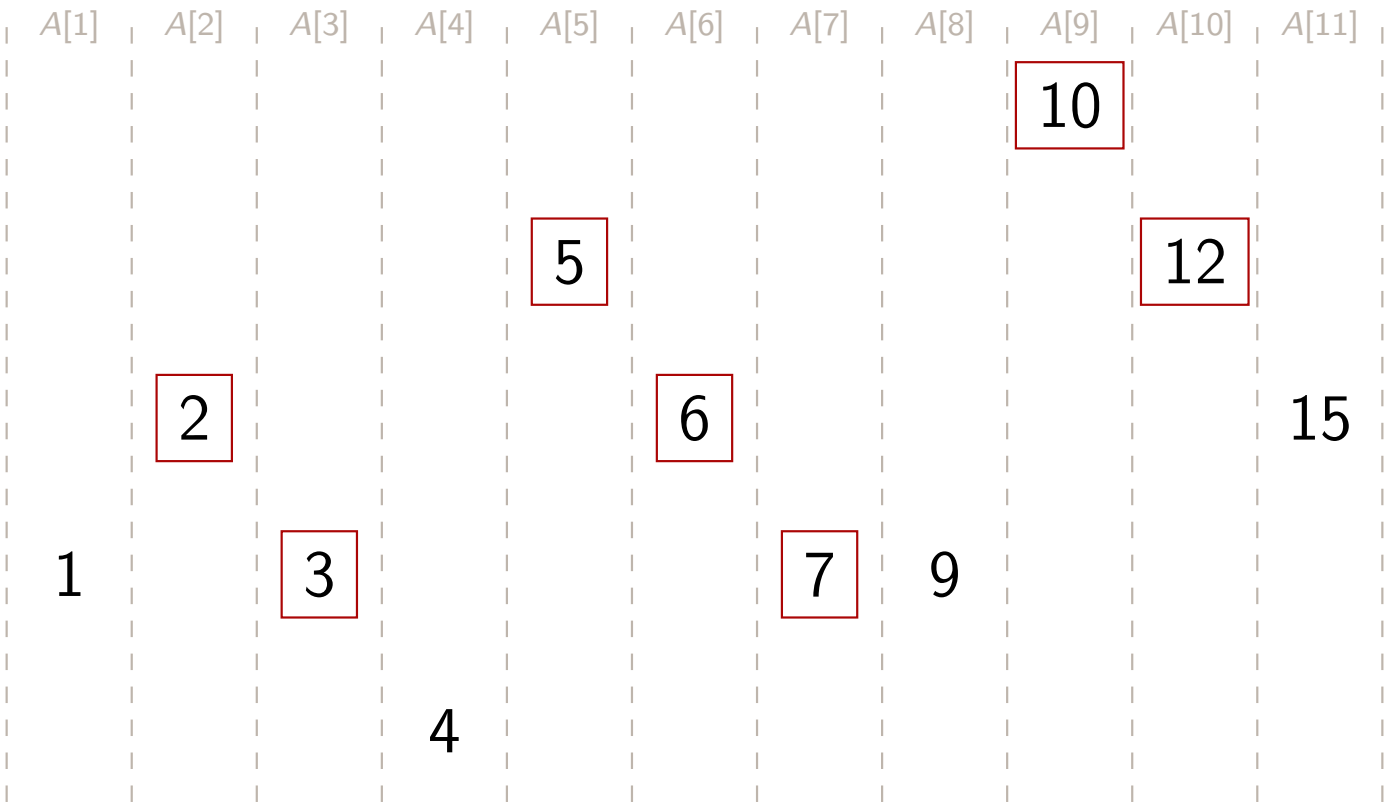
# Recall: Quick Sort



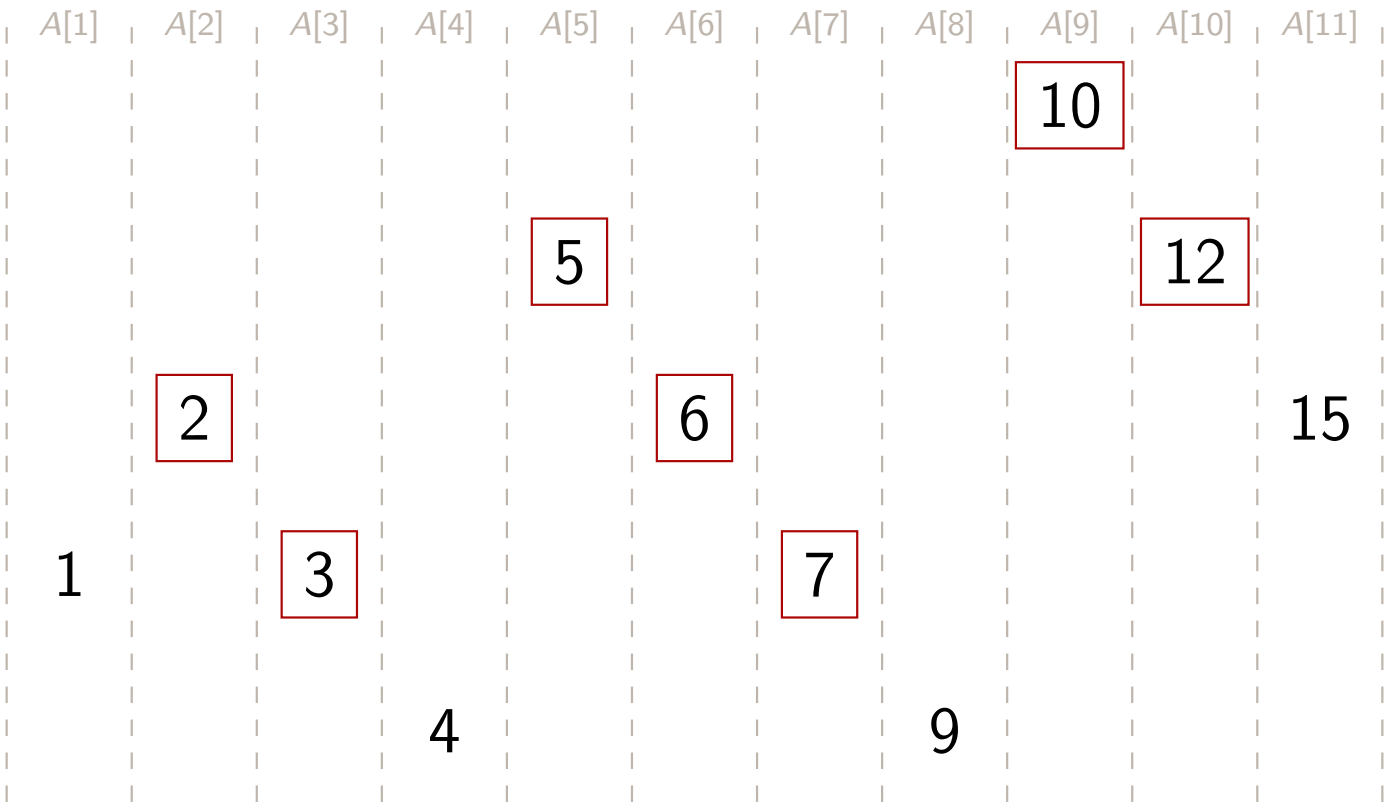
# Recall: Quick Sort



# Recall: Quick Sort



# Recall: Quick Sort



# Recall: Quick Sort

- Randomized quick sort has expected running time $\Theta(n \lg n)$ for any input.

# Recall: Quick Sort

- Randomized quick sort has expected running time $\Theta(n \lg n)$ for any input.
- The algorithm is in-place

# Recall: Quick Sort

- Randomized quick sort has expected running time $\Theta(n \lg n)$ for any input.
- The algorithm is in-place
- Very efficient and easy to implement in practice

# Recall: Quick Sort

- Randomized quick sort has expected running time $\Theta(n \lg n)$ for any input.
- The algorithm is in-place
- Very efficient and easy to implement in practice
- Based on divide-and-conquer paradigm (as merge-sort)


# WHY ALWAYS $\Theta(n \lg n)$?

## Quick-Sort, Merge-Sort, Heap-Sort ...

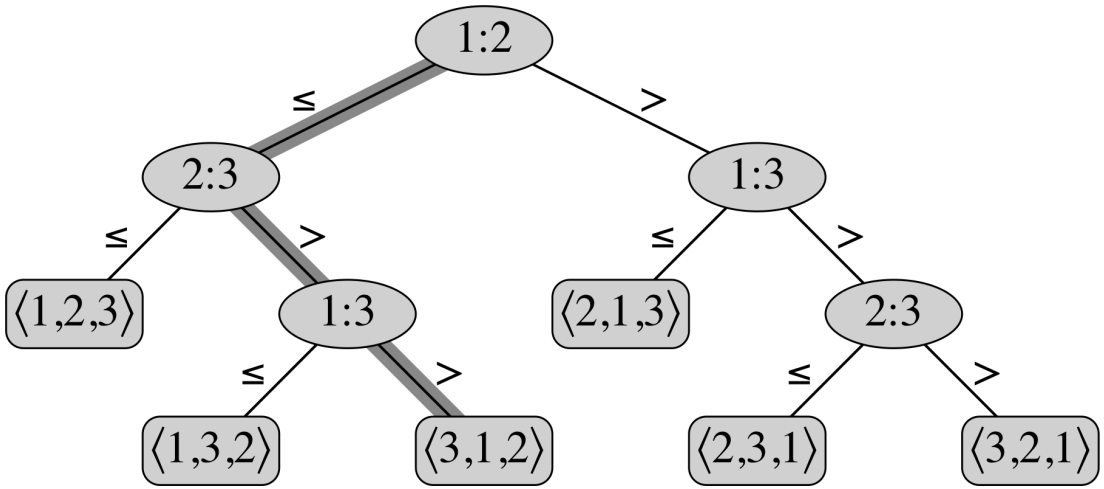
# Lower bounds for sorting

- $\Omega(n)$ to even examine the inputs

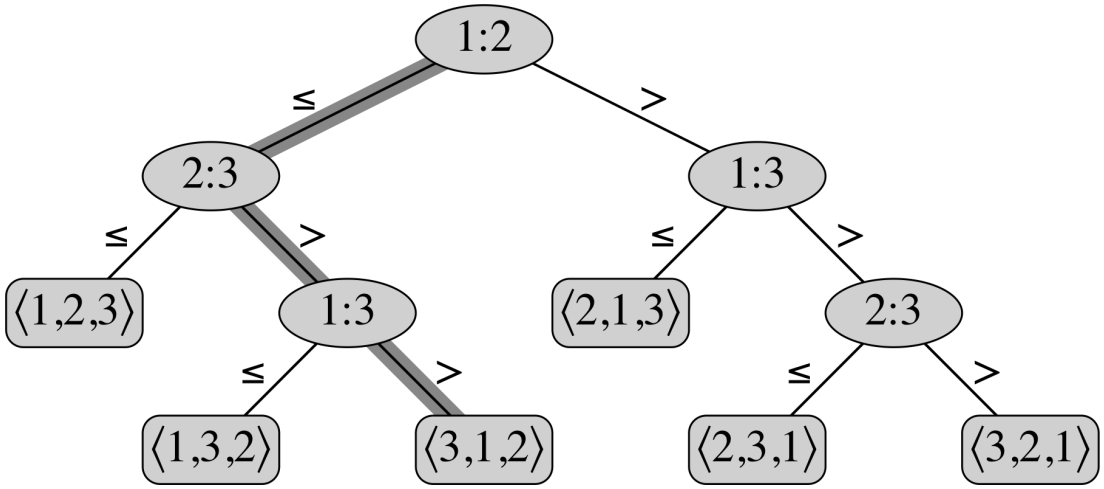
Better lower bound for "comparison" sorting:

- The only operation that may be used to gain order information about a sequence is comparison of pairs of elements.
- All sorts seen so far are comparison sorts: insertion sort, merge sort, quicksort, heapsort

# Decision tree

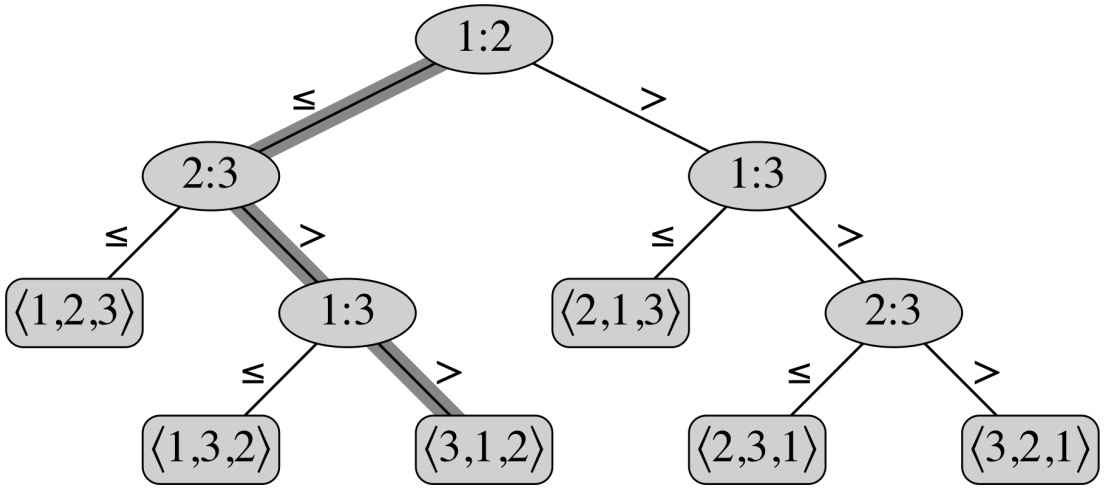


# Decision tree



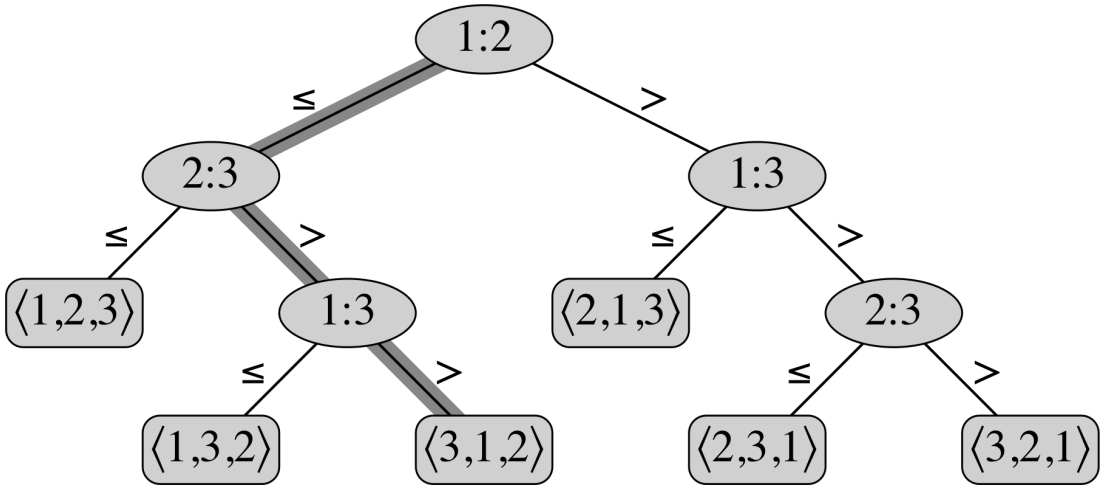

- Abstraction of any comparison sort

# Decision tree



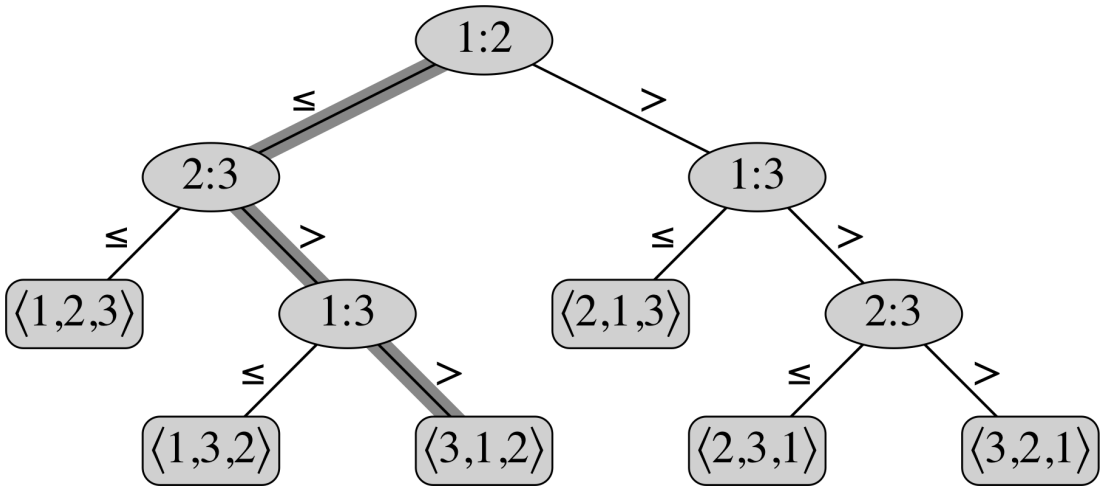

- Abstraction of any comparison sort
- Represents comparisons made by
  - a specific sorting algorithm
  - on inputs of a given size

# Decision tree

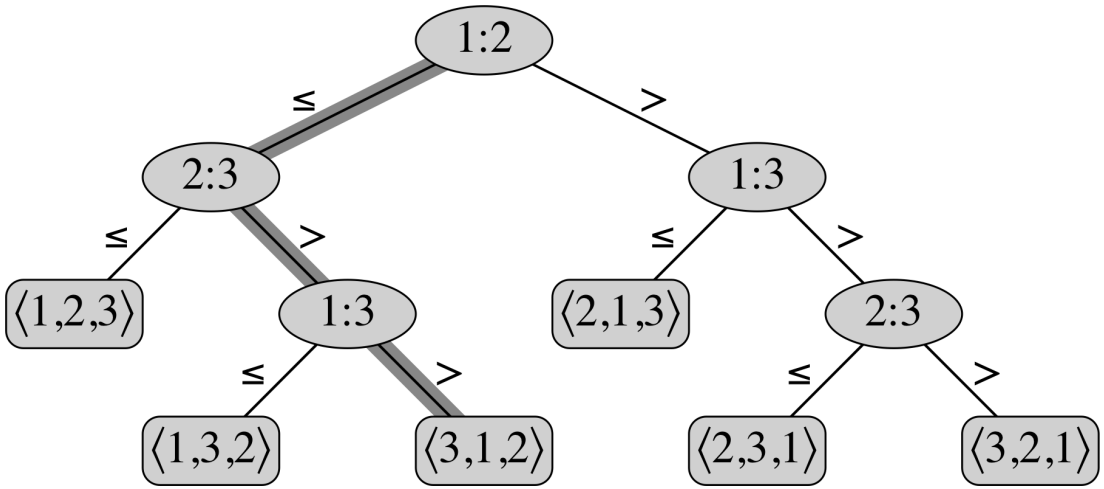



- Abstraction of any comparison sort
- Represents comparisons made by
  - a specific sorting algorithm
  - on inputs of a given size
- We are only counting #comparisons (OK since looking for lower bound)

# Decision tree

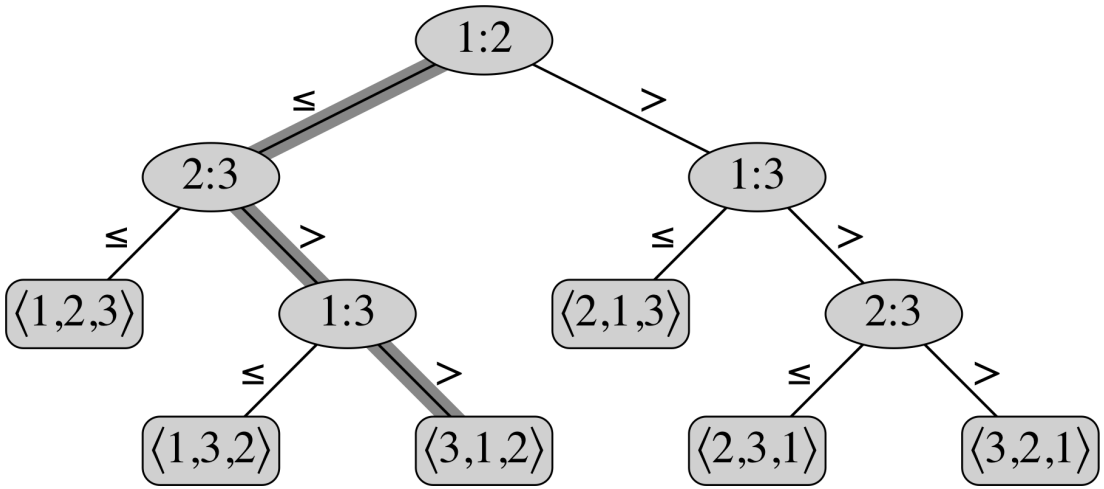


# Decision tree



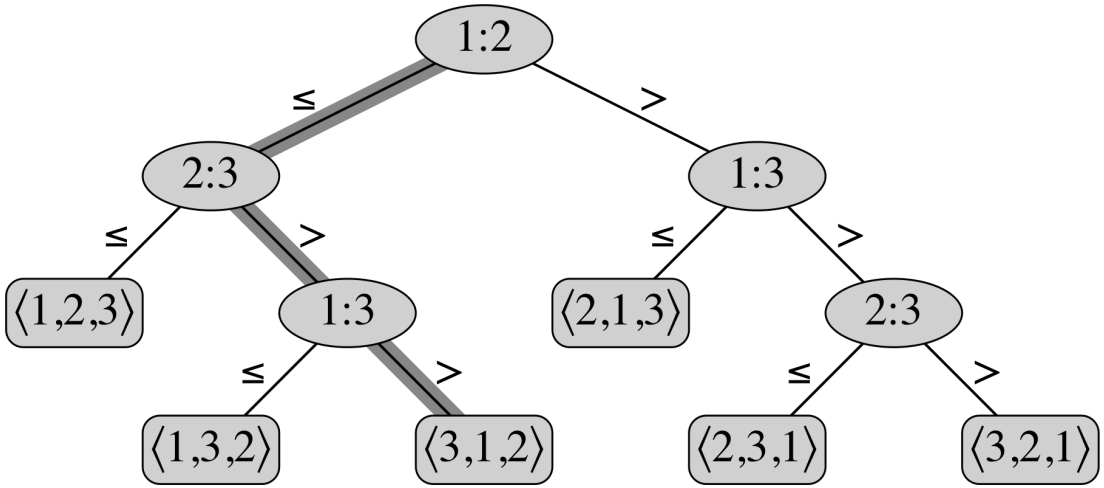

What is the number of leaves of the decision tree?

# Decision tree




What is the number of leaves of the decision tree? $n!$

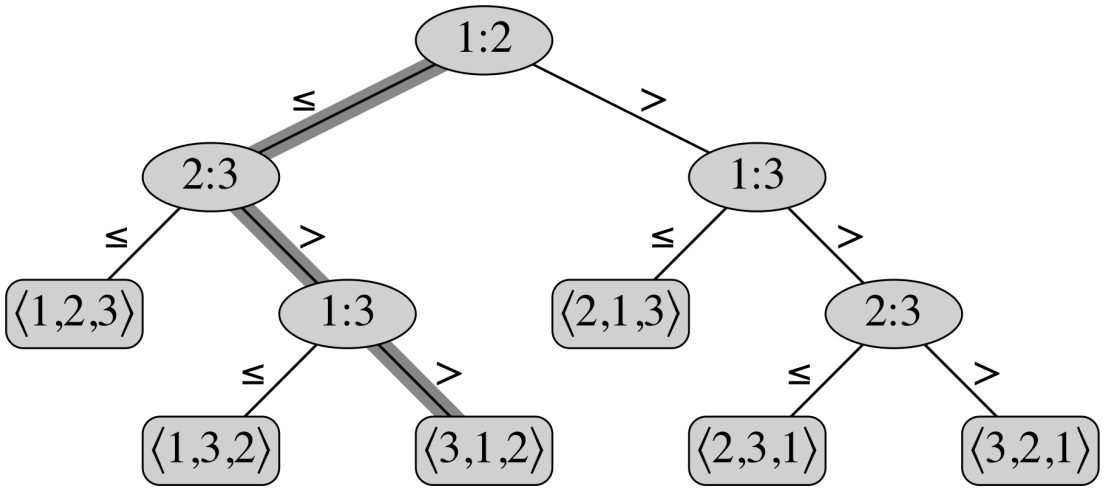
# Decision tree




What is the number of leaves of the decision tree? $n!$

What is the height of the decision tree?

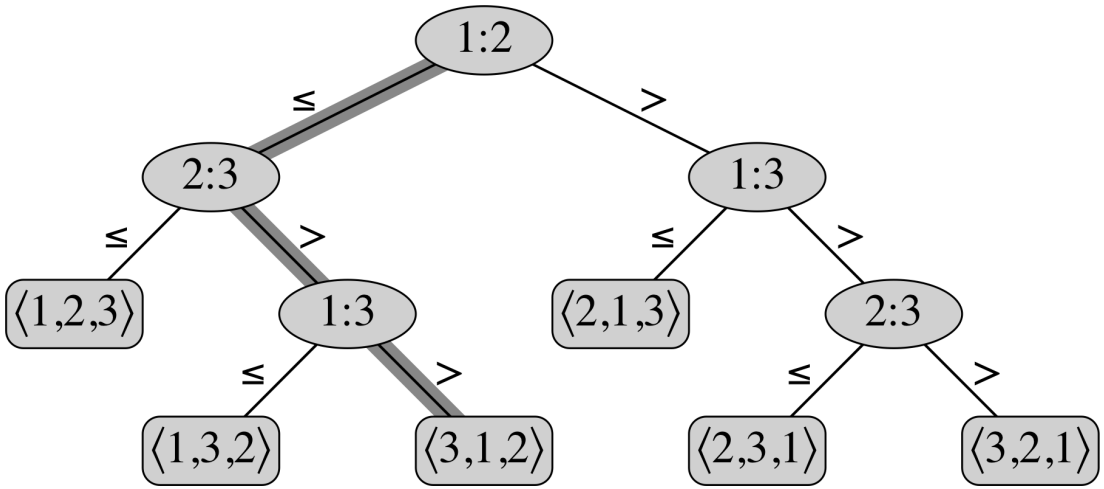
# Decision tree




What is the number of leaves of the decision tree? $n!$

What is the height of the decision tree? $\Omega(\log n!) = \Omega(n \lg n)$

# Decision tree




What is the number of leaves of the decision tree? $n!$

What is the height of the decision tree? $\Omega(\log n!) = \Omega(n \lg n)$

Therefore, exists input permutation so that we need to do $\Omega(n \lg n)$ comparisons

Similar argument implies that $\Omega(n \lg n)$ comparisons are necessary for most inputs

# Comparison sorting

- Any such algorithm takes (expected) time $\Omega(n \lg n)$
- In some sense, merge-sort, heapsort, and quicksort are optimal

# LINEAR TIME SORTING
# (non-comparison sorts)

# Assumption about keys

Input: $A[1 \ldots n]$, where $A[j] \in \{0, 1, \ldots, k\}$ for $j = 1, 2, \ldots, n$.
Array $A$ and values $n$ and $k$ are given as parameters

Output: $B[1 \ldots n]$ sorted

# Assumption about keys

Input: $A[1 \ldots n]$, where $A[j] \in \{0, 1, \ldots, k\}$ for $j = 1, 2, \ldots, n$.
Array $A$ and values $n$ and $k$ are given as parameters

Output: $B[1 \ldots n]$ sorted

Example $k = 5, n = 8$:

| A: | 2 | 5 | 3 | 0 | 2' | 3' | 0' | 3'' |
|---|---|---|---|---|---|---|---|---|

# Assumption about keys

Input: $A[1 \ldots n]$, where $A[j] \in \{0, 1, \ldots, k\}$ for $j = 1, 2, \ldots, n$.
Array $A$ and values $n$ and $k$ are given as parameters

Output: $B[1 \ldots n]$ sorted

Example $k = 5, n = 8$:

| A: | 2 | 5 | 3 | 0 | 2' | 3' | 0' | 3'' |
|---|---|---|---|---|---|---|---|---|

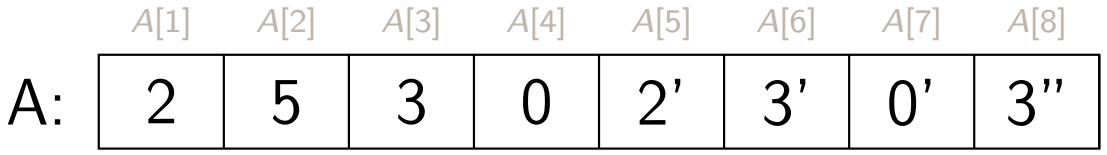
| B: | 0 | 0' | 2 | 2' | 3 | 3' | 3'' | 5 |
|---|---|---|---|---|---|---|---|---|

# Algorithm

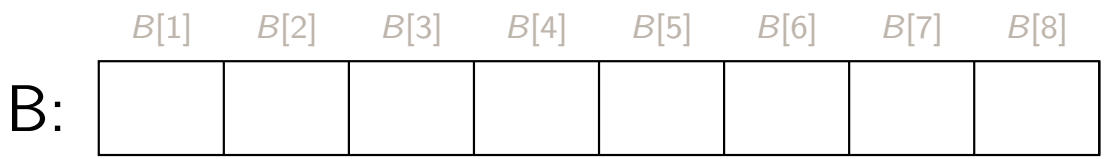
## Use an auxiliar array $C[0 \ldots k]$

COUNTING-SORT$(A, B, n, k)$

let $C[0..k]$ be a new array







# Algorithm

## Use an auxiliar array $C[0 \ldots k]$

```
COUNTING-SORT(A, B, n, k)
  let C[0..k] be a new array
  for i = 0 to k
      C[i] = 0
```

A:

| $A[1]$ | $A[2]$ | $A[3]$ | $A[4]$ | $A[5]$ | $A[6]$ | $A[7]$ | $A[8]$ |
|---|---|---|---|---|---|---|---|
| 2 | 5 | 3 | 0 | 2' | 3' | 0' | 3'' |

C:

| $C[0]$ | $C[1]$ | $C[2]$ | $C[3]$ | $C[4]$ | $C[5]$ |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 |

B:

| $B[1]$ | $B[2]$ | $B[3]$ | $B[4]$ | $B[5]$ | $B[6]$ | $B[7]$ | $B[8]$ |
|---|---|---|---|---|---|---|---|
| | | | | | | | |

# Algorithm

## Use an auxiliar array $C[0 \ldots k]$

```
COUNTING-SORT(A, B, n, k)
  let C[0..k] be a new array
  for i = 0 to k
      C[i] = 0
  for j = 1 to n
      C[A[j]] = C[A[j]] + 1
```

A:

| A[1] | A[2] | A[3] | A[4] | A[5] | A[6] | A[7] | A[8] |
|---|---|---|---|---|---|---|---|
| 2 | 5 | 3 | 0 | 2' | 3' | 0' | 3'' |

C:

| C[0] | C[1] | C[2] | C[3] | C[4] | C[5] |
|---|---|---|---|---|---|
| 2 | 0 | 2 | 3 | 0 | 1 |

B:

| B[1] | B[2] | B[3] | B[4] | B[5] | B[6] | B[7] | B[8] |
|---|---|---|---|---|---|---|---|
| | | | | | | | |

# Algorithm

## Use an auxiliar array $C[0 \ldots k]$

```
COUNTING-SORT(A, B, n, k)
  let C[0..k] be a new array
  for i = 0 to k
      C[i] = 0
  for j = 1 to n
      C[A[j]] = C[A[j]] + 1
  for i = 1 to k
      C[i] = C[i] + C[i − 1]
```

| | A[1] | A[2] | A[3] | A[4] | A[5] | A[6] | A[7] | A[8] |
|---|---|---|---|---|---|---|---|---|
| A: | 2 | 5 | 3 | 0 | 2' | 3' | 0' | 3'' |

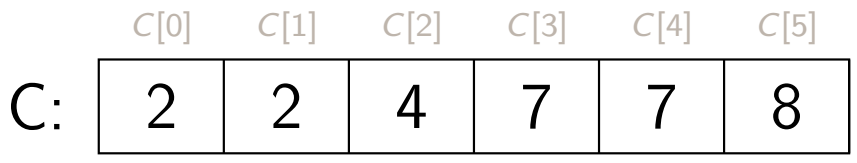
| | C[0] | C[1] | C[2] | C[3] | C[4] | C[5] |
|---|---|---|---|---|---|---|
| C: | 2 | 2 | 4 | 7 | 7 | 8 |

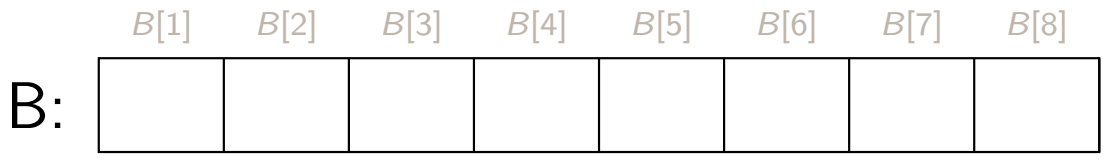
| | B[1] | B[2] | B[3] | B[4] | B[5] | B[6] | B[7] | B[8] |
|---|---|---|---|---|---|---|---|---|
| B: | | | | | | | | |

# Algorithm

## Use an auxiliar array $C[0 \ldots k]$

```
COUNTING-SORT(A, B, n, k)
  let C[0..k] be a new array
  for i = 0 to k
      C[i] = 0
  for j = 1 to n
      C[A[j]] = C[A[j]] + 1
  for i = 1 to k
      C[i] = C[i] + C[i − 1]
  for j = n downto 1
      B[C[A[j]]] = A[j]
      C[A[j]] = C[A[j]] − 1
```

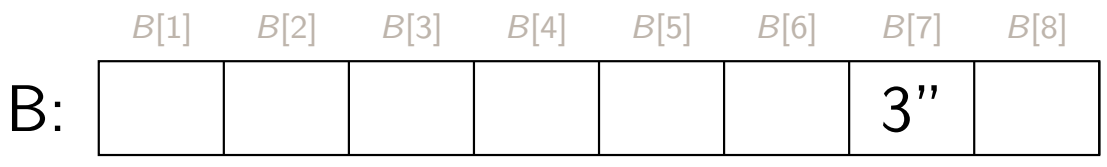



# Algorithm

## Use an auxiliar array $C[0 \ldots k]$

```
COUNTING-SORT(A, B, n, k)
  let C[0..k] be a new array
  for i = 0 to k
      C[i] = 0
  for j = 1 to n
      C[A[j]] = C[A[j]] + 1
  for i = 1 to k
      C[i] = C[i] + C[i − 1]
  for j = n downto 1
      B[C[A[j]]] = A[j]
      C[A[j]] = C[A[j]] − 1
```

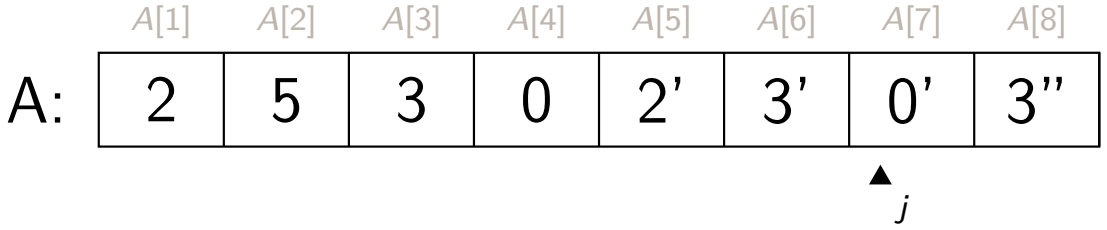


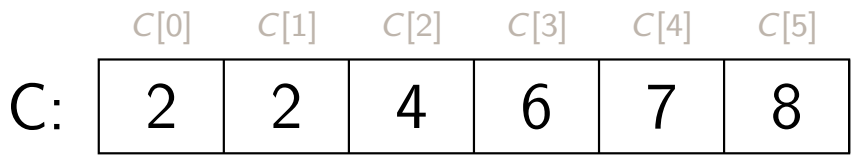


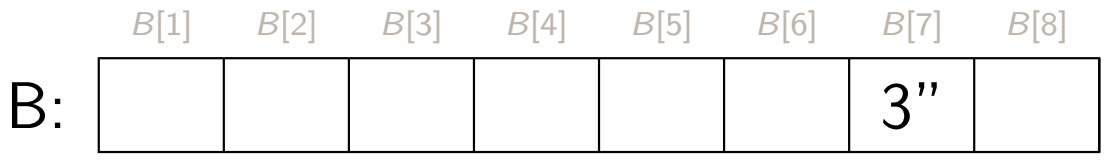
# Algorithm

## Use an auxiliar array $C[0 \ldots k]$

```
COUNTING-SORT(A, B, n, k)
  let C[0..k] be a new array
  for i = 0 to k
      C[i] = 0
  for j = 1 to n
      C[A[j]] = C[A[j]] + 1
  for i = 1 to k
      C[i] = C[i] + C[i − 1]
  for j = n downto 1
      B[C[A[j]]] = A[j]
      C[A[j]] = C[A[j]] − 1
```



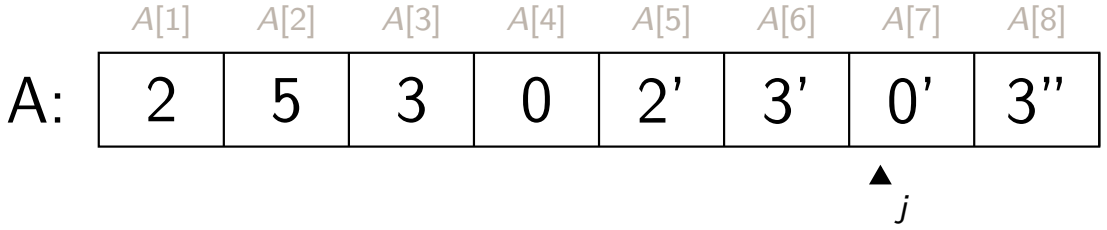



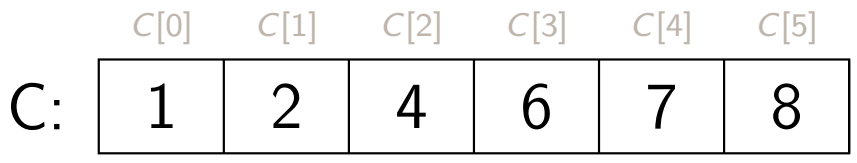



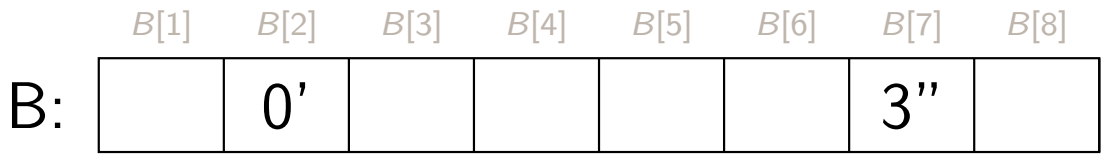
# Algorithm

## Use an auxiliar array $C[0 \ldots k]$

```
COUNTING-SORT(A, B, n, k)
  let C[0..k] be a new array
  for i = 0 to k
      C[i] = 0
  for j = 1 to n
      C[A[j]] = C[A[j]] + 1
  for i = 1 to k
      C[i] = C[i] + C[i − 1]
  for j = n downto 1
      B[C[A[j]]] = A[j]
      C[A[j]] = C[A[j]] − 1
```



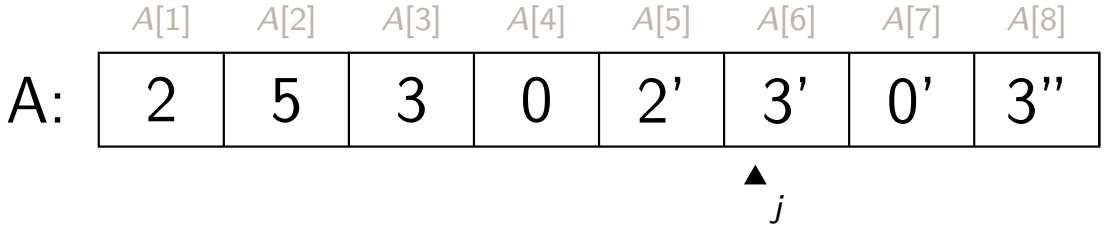



# Algorithm

## Use an auxiliar array $C[0 \ldots k]$

```
COUNTING-SORT(A, B, n, k)
  let C[0..k] be a new array
  for i = 0 to k
      C[i] = 0
  for j = 1 to n
      C[A[j]] = C[A[j]] + 1
  for i = 1 to k
      C[i] = C[i] + C[i − 1]
  for j = n downto 1
      B[C[A[j]]] = A[j]
      C[A[j]] = C[A[j]] − 1
```

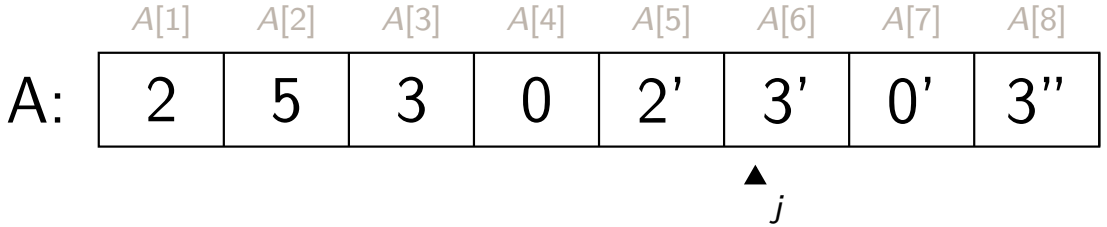




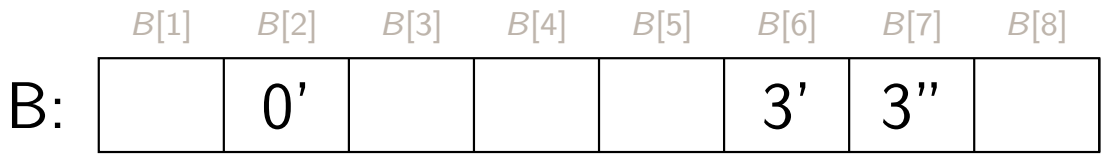
# Algorithm

## Use an auxiliar array $C[0 \ldots k]$

```
COUNTING-SORT(A, B, n, k)
  let C[0..k] be a new array
  for i = 0 to k
      C[i] = 0
  for j = 1 to n
      C[A[j]] = C[A[j]] + 1
  for i = 1 to k
      C[i] = C[i] + C[i − 1]
  for j = n downto 1
      B[C[A[j]]] = A[j]
      C[A[j]] = C[A[j]] − 1
```



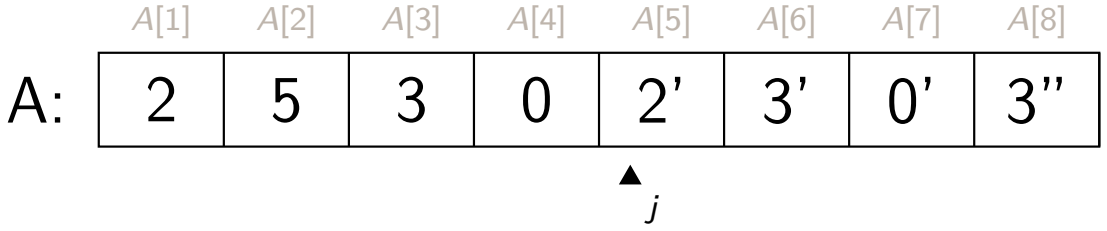


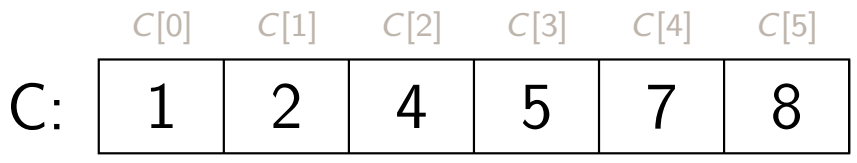


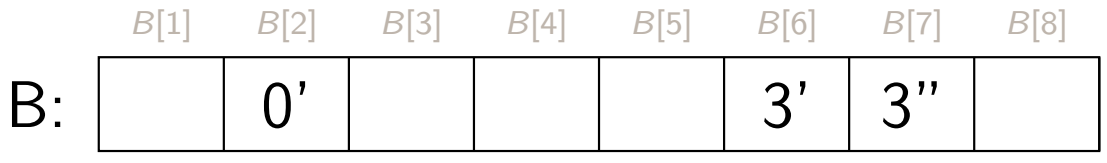
# Algorithm

## Use an auxiliar array $C[0 \ldots k]$

```
COUNTING-SORT(A, B, n, k)
  let C[0..k] be a new array
  for i = 0 to k
      C[i] = 0
  for j = 1 to n
      C[A[j]] = C[A[j]] + 1
  for i = 1 to k
      C[i] = C[i] + C[i − 1]
  for j = n downto 1
      B[C[A[j]]] = A[j]
      C[A[j]] = C[A[j]] − 1
```



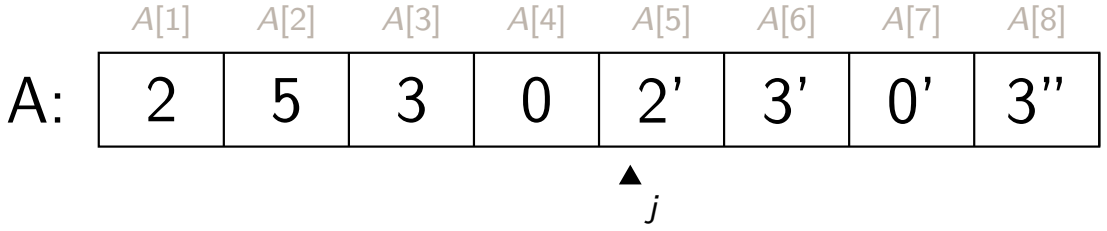



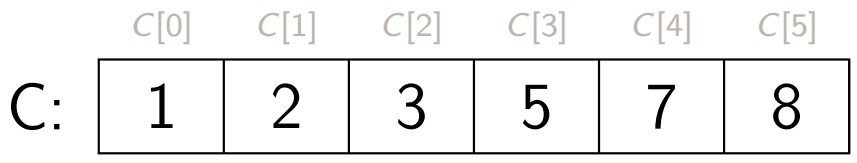



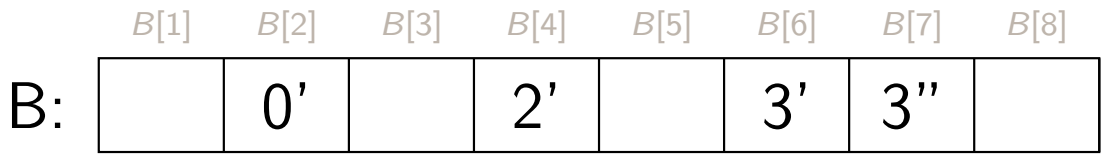
# Algorithm

## Use an auxiliar array $C[0 \ldots k]$

```
COUNTING-SORT(A, B, n, k)
  let C[0..k] be a new array
  for i = 0 to k
      C[i] = 0
  for j = 1 to n
      C[A[j]] = C[A[j]] + 1
  for i = 1 to k
      C[i] = C[i] + C[i − 1]
  for j = n downto 1
      B[C[A[j]]] = A[j]
      C[A[j]] = C[A[j]] − 1
```



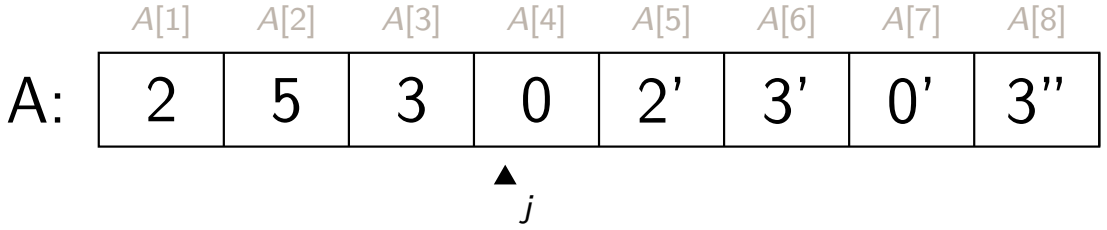



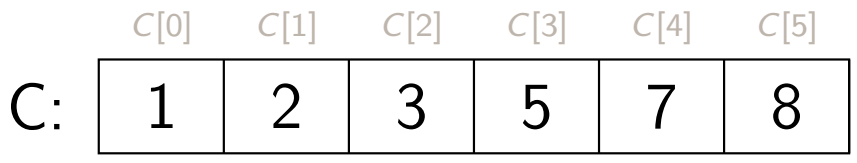



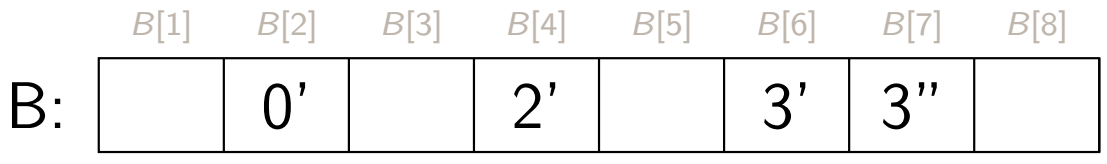
# Algorithm

## Use an auxiliar array $C[0 \ldots k]$

```
COUNTING-SORT(A, B, n, k)
  let C[0..k] be a new array
  for i = 0 to k
      C[i] = 0
  for j = 1 to n
      C[A[j]] = C[A[j]] + 1
  for i = 1 to k
      C[i] = C[i] + C[i − 1]
  for j = n downto 1
      B[C[A[j]]] = A[j]
      C[A[j]] = C[A[j]] − 1
```



# Algorithm

## Use an auxiliar array $C[0 \ldots k]$

```
COUNTING-SORT(A, B, n, k)
  let C[0..k] be a new array
  for i = 0 to k
      C[i] = 0
  for j = 1 to n
      C[A[j]] = C[A[j]] + 1
  for i = 1 to k
      C[i] = C[i] + C[i − 1]
  for j = n downto 1
      B[C[A[j]]] = A[j]
      C[A[j]] = C[A[j]] − 1
```

A:

| A[1] | A[2] | A[3] | A[4] | A[5] | A[6] | A[7] | A[8] |
|---|---|---|---|---|---|---|---|
| 2 | 5 | 3 | 0 | 2' | 3' | 0' | 3'' |

▲ $j$

C:

| C[0] | C[1] | C[2] | C[3] | C[4] | C[5] |
|---|---|---|---|---|---|
| 0 | 2 | 3 | 5 | 7 | 8 |

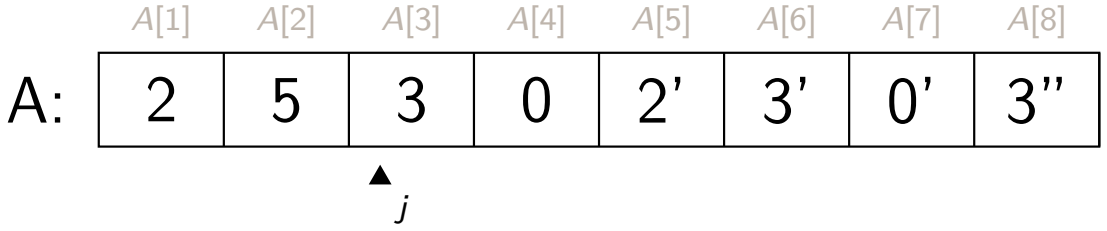
B:

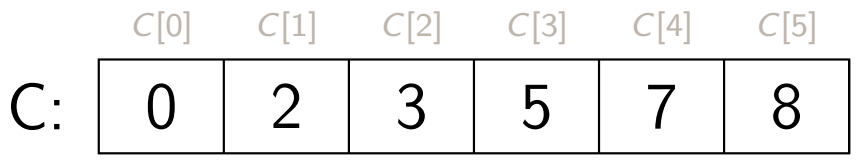
| B[1] | B[2] | B[3] | B[4] | B[5] | B[6] | B[7] | B[8] |
|---|---|---|---|---|---|---|---|
| 0 | 0' | | 2' | | 3' | 3'' | |

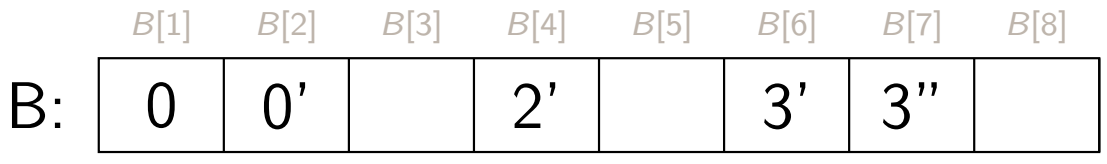
# Algorithm

## Use an auxiliar array $C[0 \ldots k]$

```
COUNTING-SORT(A, B, n, k)
  let C[0..k] be a new array
  for i = 0 to k
      C[i] = 0
  for j = 1 to n
      C[A[j]] = C[A[j]] + 1
  for i = 1 to k
      C[i] = C[i] + C[i − 1]
  for j = n downto 1
      B[C[A[j]]] = A[j]
      C[A[j]] = C[A[j]] − 1
```



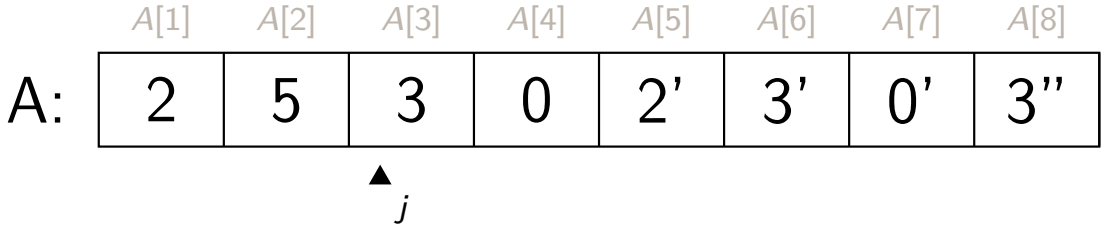



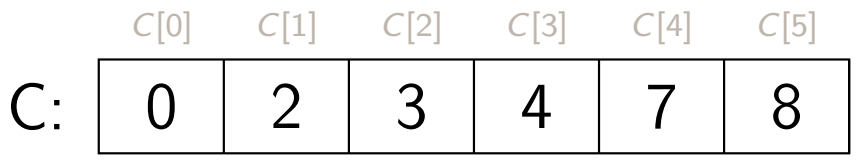



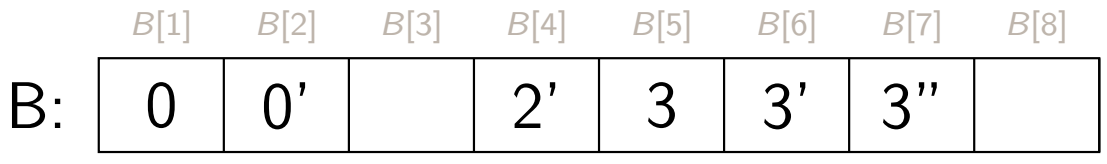
# Algorithm

## Use an auxiliar array $C[0 \ldots k]$

```
COUNTING-SORT(A, B, n, k)
  let C[0..k] be a new array
  for i = 0 to k
      C[i] = 0
  for j = 1 to n
      C[A[j]] = C[A[j]] + 1
  for i = 1 to k
      C[i] = C[i] + C[i − 1]
  for j = n downto 1
      B[C[A[j]]] = A[j]
      C[A[j]] = C[A[j]] − 1
```



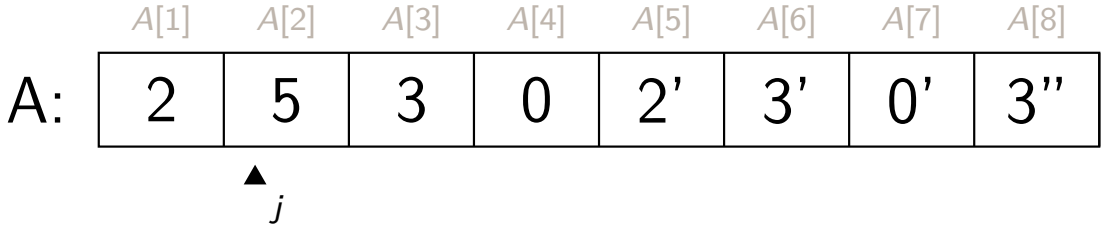



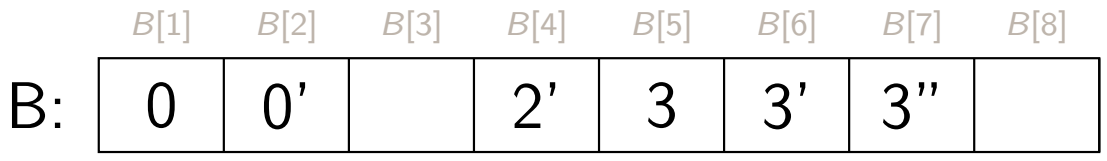
# Algorithm

## Use an auxiliar array $C[0 \ldots k]$

```
COUNTING-SORT(A, B, n, k)
  let C[0..k] be a new array
  for i = 0 to k
      C[i] = 0
  for j = 1 to n
      C[A[j]] = C[A[j]] + 1
  for i = 1 to k
      C[i] = C[i] + C[i − 1]
  for j = n downto 1
      B[C[A[j]]] = A[j]
      C[A[j]] = C[A[j]] − 1
```



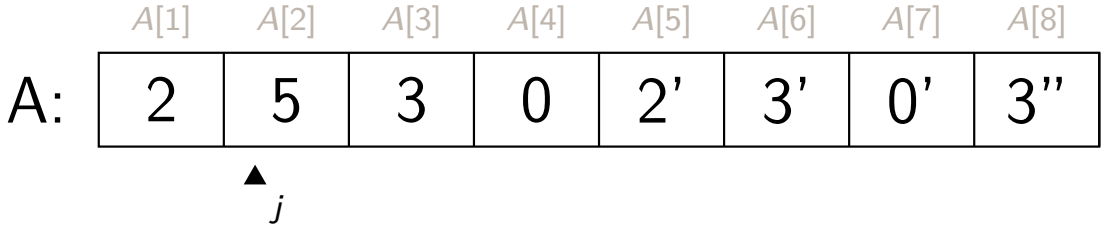




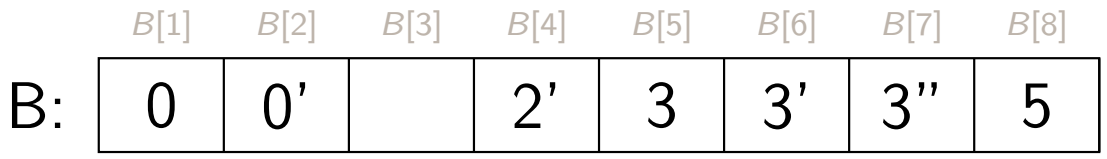
# Algorithm

## Use an auxiliar array $C[0 \ldots k]$

```
COUNTING-SORT(A, B, n, k)
  let C[0..k] be a new array
  for i = 0 to k
      C[i] = 0
  for j = 1 to n
      C[A[j]] = C[A[j]] + 1
  for i = 1 to k
      C[i] = C[i] + C[i − 1]
  for j = n downto 1
      B[C[A[j]]] = A[j]
      C[A[j]] = C[A[j]] − 1
```



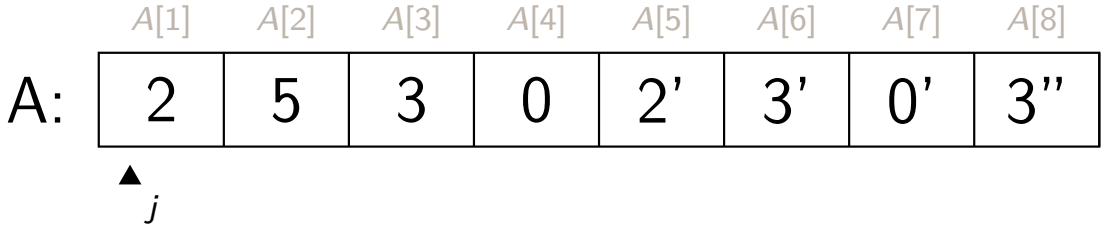


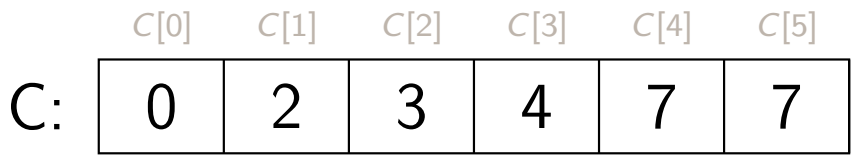


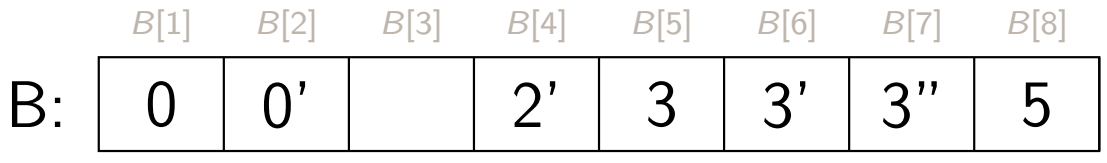
# Algorithm

## Use an auxiliar array $C[0 \ldots k]$

```
COUNTING-SORT(A, B, n, k)
  let C[0..k] be a new array
  for i = 0 to k
      C[i] = 0
  for j = 1 to n
      C[A[j]] = C[A[j]] + 1
  for i = 1 to k
      C[i] = C[i] + C[i − 1]
  for j = n downto 1
      B[C[A[j]]] = A[j]
      C[A[j]] = C[A[j]] − 1
```



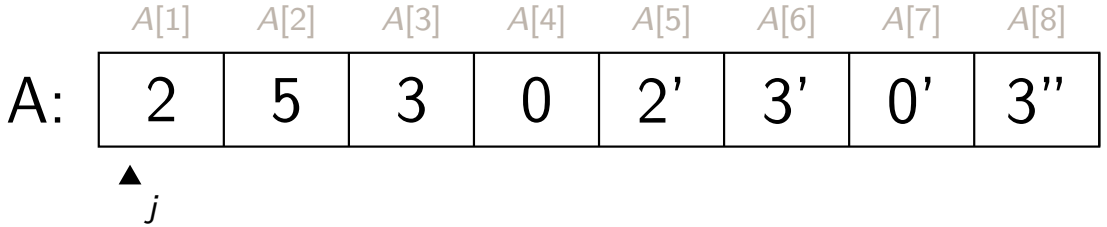



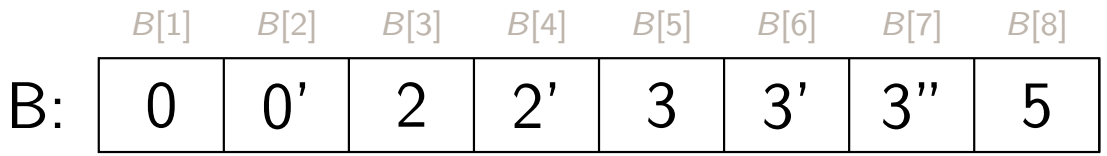
# Algorithm

## Use an auxiliar array $C[0 \ldots k]$

```
COUNTING-SORT(A, B, n, k)
  let C[0..k] be a new array
  for i = 0 to k
      C[i] = 0
  for j = 1 to n
      C[A[j]] = C[A[j]] + 1
  for i = 1 to k
      C[i] = C[i] + C[i − 1]
  for j = n downto 1
      B[C[A[j]]] = A[j]
      C[A[j]] = C[A[j]] − 1
```







# Algorithm

## Use an auxiliar array $C[0 \ldots k]$

```
COUNTING-SORT(A, B, n, k)
  let C[0..k] be a new array
  for i = 0 to k
      C[i] = 0
  for j = 1 to n
      C[A[j]] = C[A[j]] + 1
  for i = 1 to k
      C[i] = C[i] + C[i − 1]
  for j = n downto 1
      B[C[A[j]]] = A[j]
      C[A[j]] = C[A[j]] − 1
```

| | A[1] | A[2] | A[3] | A[4] | A[5] | A[6] | A[7] | A[8] |
|---|---|---|---|---|---|---|---|---|
| A: | 2 | 5 | 3 | 0 | 2' | 3' | 0' | 3'' |

| | C[0] | C[1] | C[2] | C[3] | C[4] | C[5] |
|---|---|---|---|---|---|---|
| C: | 0 | 2 | 2 | 4 | 7 | 7 |

| | B[1] | B[2] | B[3] | B[4] | B[5] | B[6] | B[7] | B[8] |
|---|---|---|---|---|---|---|---|---|
| B: | 0 | 0' | 2 | 2' | 3 | 3' | 3'' | 5 |

# Runtime analysis

COUNTING-SORT$(A, B, n, k)$

```
let C[0..k] be a new array
for i = 0 to k
    C[i] = 0
for j = 1 to n
    C[A[j]] = C[A[j]] + 1
for i = 1 to k
    C[i] = C[i] + C[i − 1]
for j = n downto 1
    B[C[A[j]]] = A[j]
    C[A[j]] = C[A[j]] − 1
```

# Runtime analysis

```
COUNTING-SORT(A, B, n, k)
  let C[0..k] be a new array
  for i = 0 to k
      C[i] = 0
  for j = 1 to n
      C[A[j]] = C[A[j]] + 1
  for i = 1 to k
      C[i] = C[i] + C[i − 1]
  for j = n downto 1
      B[C[A[j]]] = A[j]
      C[A[j]] = C[A[j]] − 1
```

- First **for**-loop:

# Runtime analysis

```
COUNTING-SORT(A, B, n, k)
  let C[0..k] be a new array
  for i = 0 to k
      C[i] = 0
  for j = 1 to n
      C[A[j]] = C[A[j]] + 1
  for i = 1 to k
      C[i] = C[i] + C[i − 1]
  for j = n downto 1
      B[C[A[j]]] = A[j]
      C[A[j]] = C[A[j]] − 1
```

- First **for**-loop: $\Theta(k)$

# Runtime analysis

```
COUNTING-SORT(A, B, n, k)
  let C[0..k] be a new array
  for i = 0 to k
      C[i] = 0
  for j = 1 to n
      C[A[j]] = C[A[j]] + 1
  for i = 1 to k
      C[i] = C[i] + C[i − 1]
  for j = n downto 1
      B[C[A[j]]] = A[j]
      C[A[j]] = C[A[j]] − 1
```

- First **for**-loop: $\Theta(k)$
- 2nd **for**-loop:

# Runtime analysis

```
COUNTING-SORT(A, B, n, k)
  let C[0..k] be a new array
  for i = 0 to k
      C[i] = 0
  for j = 1 to n
      C[A[j]] = C[A[j]] + 1
  for i = 1 to k
      C[i] = C[i] + C[i − 1]
  for j = n downto 1
      B[C[A[j]]] = A[j]
      C[A[j]] = C[A[j]] − 1
```

- First **for**-loop: $\Theta(k)$
- 2nd **for**-loop: $\Theta(n)$

# Runtime analysis

```
COUNTING-SORT(A, B, n, k)
  let C[0..k] be a new array
  for i = 0 to k
      C[i] = 0
  for j = 1 to n
      C[A[j]] = C[A[j]] + 1
  for i = 1 to k
      C[i] = C[i] + C[i − 1]
  for j = n downto 1
      B[C[A[j]]] = A[j]
      C[A[j]] = C[A[j]] − 1
```

- First **for**-loop: $\Theta(k)$
- 2nd **for**-loop: $\Theta(n)$
- 3rd **for**-loop:

# Runtime analysis

```
COUNTING-SORT(A, B, n, k)
  let C[0..k] be a new array
  for i = 0 to k
      C[i] = 0
  for j = 1 to n
      C[A[j]] = C[A[j]] + 1
  for i = 1 to k
      C[i] = C[i] + C[i − 1]
  for j = n downto 1
      B[C[A[j]]] = A[j]
      C[A[j]] = C[A[j]] − 1
```

- First **for**-loop: $\Theta(k)$
- 2nd **for**-loop: $\Theta(n)$
- 3rd **for**-loop: $\Theta(k)$

# Runtime analysis

```
COUNTING-SORT(A, B, n, k)
  let C[0..k] be a new array
  for i = 0 to k
      C[i] = 0
  for j = 1 to n
      C[A[j]] = C[A[j]] + 1
  for i = 1 to k
      C[i] = C[i] + C[i − 1]
  for j = n downto 1
      B[C[A[j]]] = A[j]
      C[A[j]] = C[A[j]] − 1
```

- First **for**-loop: $\Theta(k)$
- 2nd **for**-loop: $\Theta(n)$
- 3rd **for**-loop: $\Theta(k)$
- 4th **for**-loop:

# Runtime analysis

```
COUNTING-SORT(A, B, n, k)
  let C[0..k] be a new array
  for i = 0 to k
      C[i] = 0
  for j = 1 to n
      C[A[j]] = C[A[j]] + 1
  for i = 1 to k
      C[i] = C[i] + C[i − 1]
  for j = n downto 1
      B[C[A[j]]] = A[j]
      C[A[j]] = C[A[j]] − 1
```

- First **for**-loop: $\Theta(k)$
- 2nd **for**-loop: $\Theta(n)$
- 3rd **for**-loop: $\Theta(k)$
- 4th **for**-loop: $\Theta(n)$

# Runtime analysis

```
COUNTING-SORT(A, B, n, k)
  let C[0..k] be a new array
  for i = 0 to k
      C[i] = 0
  for j = 1 to n
      C[A[j]] = C[A[j]] + 1
  for i = 1 to k
      C[i] = C[i] + C[i − 1]
  for j = n downto 1
      B[C[A[j]]] = A[j]
      C[A[j]] = C[A[j]] − 1
```

- First **for**-loop: $\Theta(k)$
- 2nd **for**-loop: $\Theta(n)$
- 3rd **for**-loop: $\Theta(k)$
- 4th **for**-loop: $\Theta(n)$
- Total: $\Theta(n + k)$

# Runtime analysis

```
COUNTING-SORT(A, B, n, k)
  let C[0..k] be a new array
  for i = 0 to k
      C[i] = 0
  for j = 1 to n
      C[A[j]] = C[A[j]] + 1
  for i = 1 to k
      C[i] = C[i] + C[i − 1]
  for j = n downto 1
      B[C[A[j]]] = A[j]
      C[A[j]] = C[A[j]] − 1
```

How big $k$ is practical?

- First **for**-loop: $\Theta(k)$
- 2nd **for**-loop: $\Theta(n)$
- 3rd **for**-loop: $\Theta(k)$
- 4th **for**-loop: $\Theta(n)$
- Total: $\Theta(n + k)$

# Runtime analysis

```
COUNTING-SORT(A, B, n, k)
  let C[0..k] be a new array
  for i = 0 to k
      C[i] = 0
  for j = 1 to n
      C[A[j]] = C[A[j]] + 1
  for i = 1 to k
      C[i] = C[i] + C[i − 1]
  for j = n downto 1
      B[C[A[j]]] = A[j]
      C[A[j]] = C[A[j]] − 1
```

- First **for**-loop: $\Theta(k)$
- 2nd **for**-loop: $\Theta(n)$
- 3rd **for**-loop: $\Theta(k)$
- 4th **for**-loop: $\Theta(n)$
- Total: $\Theta(n + k)$

How big $k$ is practical?

- 32-bit values?

# Runtime analysis

```
COUNTING-SORT(A, B, n, k)
  let C[0..k] be a new array
  for i = 0 to k
      C[i] = 0
  for j = 1 to n
      C[A[j]] = C[A[j]] + 1
  for i = 1 to k
      C[i] = C[i] + C[i − 1]
  for j = n downto 1
      B[C[A[j]]] = A[j]
      C[A[j]] = C[A[j]] − 1
```

- First **for**-loop: $\Theta(k)$
- 2nd **for**-loop: $\Theta(n)$
- 3rd **for**-loop: $\Theta(k)$
- 4th **for**-loop: $\Theta(n)$
- Total: $\Theta(n + k)$

How big $k$ is practical?

- 32-bit values? No
- 16-bit values?

# Runtime analysis

```
COUNTING-SORT(A, B, n, k)
  let C[0..k] be a new array
  for i = 0 to k
      C[i] = 0
  for j = 1 to n
      C[A[j]] = C[A[j]] + 1
  for i = 1 to k
      C[i] = C[i] + C[i − 1]
  for j = n downto 1
      B[C[A[j]]] = A[j]
      C[A[j]] = C[A[j]] − 1
```

- First **for**-loop: $\Theta(k)$
- 2nd **for**-loop: $\Theta(n)$
- 3rd **for**-loop: $\Theta(k)$
- 4th **for**-loop: $\Theta(n)$
- Total: $\Theta(n + k)$

How big $k$ is practical?

- 32-bit values? No
- 16-bit values? Probably not
- 8-bit values?

# Runtime analysis

```
COUNTING-SORT(A, B, n, k)
  let C[0..k] be a new array
  for i = 0 to k
      C[i] = 0
  for j = 1 to n
      C[A[j]] = C[A[j]] + 1
  for i = 1 to k
      C[i] = C[i] + C[i − 1]
  for j = n downto 1
      B[C[A[j]]] = A[j]
      C[A[j]] = C[A[j]] − 1
```

- First **for**-loop: $\Theta(k)$
- 2nd **for**-loop: $\Theta(n)$
- 3rd **for**-loop: $\Theta(k)$
- 4th **for**-loop: $\Theta(n)$
- Total: $\Theta(n + k)$

How big $k$ is practical?

- 32-bit values? No
- 16-bit values? Probably not
- 8-bit values? Maybe depending on $n$
- 4-bit values? Probably unless $n$ is very small

# Summary

- Comparison sort can not beat $O(n \lg n)$
- Other sorting methods available when we now structure about the input
- Counting sort runs in time $\Theta(n + k)$ when all numbers are in $\{0, 1, \ldots, k\}$