

## 9. Sorting III

---

Lower bounds for the comparison based sorting, radix- and bucket-sort

## 9.1 Lower bounds for comparison based sorting

[Ottman/Widmayer, Kap. 2.8, Cormen et al, Kap. 8.1]

# Lower bound for sorting

Up to here: worst case sorting takes  $\Omega(n \log n)$  steps.  
Is there a better way?

# Lower bound for sorting

Up to here: worst case sorting takes  $\Omega(n \log n)$  steps.

Is there a better way? No:

## *Theorem 14*

*Sorting procedures that are based on comparison require in the worst case and on average at least  $\Omega(n \log n)$  key comparisons.*

# Comparison based sorting

- An algorithm must identify the correct one of  $n!$  permutations of an array  $(A_i)_{i=1,\dots,n}$ .

# Comparison based sorting

- An algorithm must identify the correct one of  $n!$  permutations of an array  $(A_i)_{i=1,\dots,n}$ .
- At the beginning the algorithm knows nothing about the array structure.

# Comparison based sorting

- An algorithm must identify the correct one of  $n!$  permutations of an array  $(A_i)_{i=1,\dots,n}$ .
- At the beginning the algorithm knows nothing about the array structure.
- We consider the knowledge gain of the algorithm in the form of a decision tree:

# Comparison based sorting

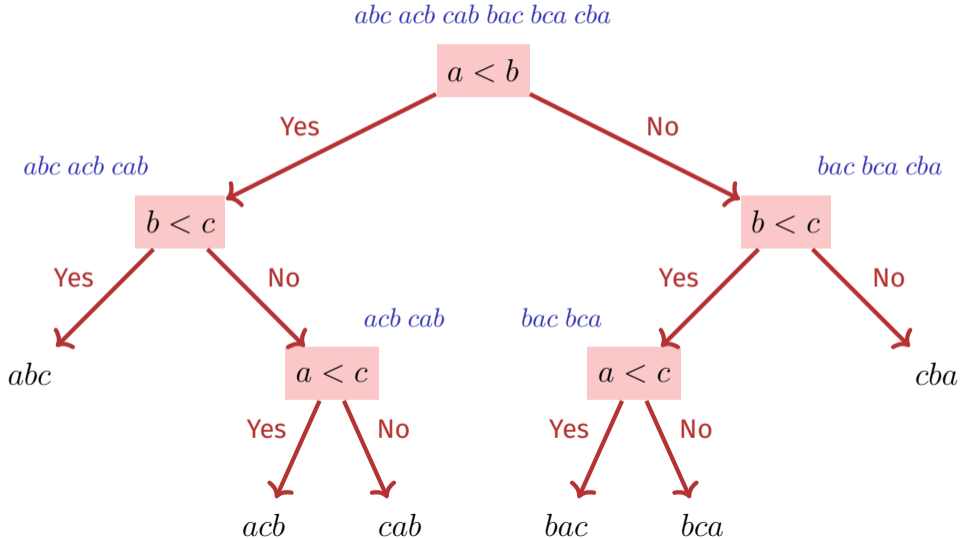
- An algorithm must identify the correct one of  $n!$  permutations of an array  $(A_i)_{i=1,\dots,n}$ .
- At the beginning the algorithm knows nothing about the array structure.
- We consider the knowledge gain of the algorithm in the form of a decision tree:
  - Nodes contain the remaining possibilities.



# Comparison based sorting

- An algorithm must identify the correct one of  $n!$  permutations of an array  $(A_i)_{i=1,\dots,n}$ .
- At the beginning the algorithm knows nothing about the array structure.
- We consider the knowledge gain of the algorithm in the form of a decision tree:
  - Nodes contain the remaining possibilities.
  - Edges contain the decisions.

# Decision tree



# Decision tree

A binary tree with  $L$  leaves provides  $K = L - 1$  inner nodes.<sup>10</sup>

The height of a binary tree with  $L$  leaves is at least  $\log_2 L$ .  $\Rightarrow$  The height of the decision tree  $h \geq \log n! \in \Omega(n \log n)$ .

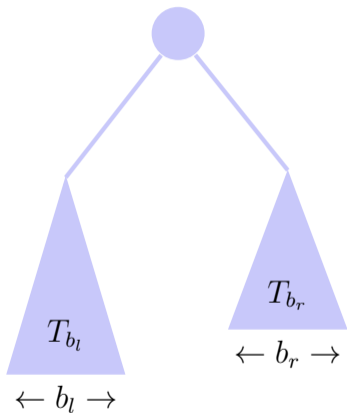
Thus the length of the longest path in the decision tree  $\in \Omega(n \log n)$ .

Remaining to show: mean length  $M(n)$  of a path  $M(n) \in \Omega(n \log n)$ .

---

<sup>10</sup>Proof: start with empty tree ( $K = 0, L = 1$ ). Each added node replaces a leaf by two leaves, i.e.}  $K \rightarrow K + 1 \Rightarrow L \rightarrow L + 1$ .

# Average lower bound



- Decision tree  $T_n$  with  $n$  leaves, average height of a leaf  $m(T_n)$
- Assumption  $m(T_n) \geq \log n$  not for all  $n$ .
- Choose smallest  $b$  with  $m(T_b) < \log b \Rightarrow b \geq 2$
- $b_l + b_r = b$  with  $b_l > 0$  and  $b_r > 0 \Rightarrow b_l < b, b_r < b \Rightarrow m(T_{b_l}) \geq \log b_l$  und  $m(T_{b_r}) \geq \log b_r$

# Average lower bound

Average height of a leaf:

$$\begin{aligned}m(T_b) &= \frac{b_l}{b}(m(T_{b_l}) + 1) + \frac{b_r}{b}(m(T_{b_r}) + 1) \\ &\geq \frac{1}{b}(b_l(\log b_l + 1) + b_r(\log b_r + 1)) = \frac{1}{b}(b_l \log 2b_l + b_r \log 2b_r) \\ &\geq \frac{1}{b}(b \log b) = \log b.\end{aligned}$$

Contradiction. ■

The last inequality holds because  $f(x) = x \log x$  is convex ( $f''(x) = 1/x > 0$ ) and for a convex function it holds that  $f((x+y)/2) \leq 1/2f(x) + 1/2f(y)$  ( $x = 2b_l$ ,  $y = 2b_r$ ).<sup>11</sup> Enter  $x = 2b_l$ ,  $y = 2b_r$ , and  $b_l + b_r = b$ .

---

<sup>11</sup>generally  $f(\lambda x + (1 - \lambda)y) \leq \lambda f(x) + (1 - \lambda)f(y)$  for  $0 \leq \lambda \leq 1$ .

## 9.2 Radixsort and Bucketsort

---

Radixsort, Bucketsort [Ottman/Widmayer, Kap. 2.5, Cormen et al, Kap. 8.3]

# Radix Sort

Sorting based on comparison: comparable keys ( $<$  or  $>$ , often  $=$ ). No further assumptions.

# Radix Sort

Sorting based on comparison: comparable keys ( $<$  or  $>$ , often  $=$ ). No further assumptions.

Different idea: use more information about the keys.



# Assumptions

Assumption: keys representable as words from an alphabet containing  $m$  elements.

## Examples

$m = 10$     decimal numbers     $183 = 183_{10}$

$m$  is called the radix of the representation.

# Assumptions

Assumption: keys representable as words from an alphabet containing  $m$  elements.

## Examples

$m = 10$	decimal numbers	$183 = 183_{10}$
$m = 2$	dual numbers	$101_2$

$m$  is called the radix of the representation.

# Assumptions

Assumption: keys representable as words from an alphabet containing  $m$  elements.

## Examples

$m = 10$	decimal numbers	$183 = 183_{10}$
$m = 2$	dual numbers	$101_2$
$m = 16$	hexadecimal numbers	$A0_{16}$

$m$  is called the radix of the representation.

# Assumptions

Assumption: keys representable as words from an alphabet containing  $m$  elements.

## Examples

$m = 10$	decimal numbers	$183 = 183_{10}$
$m = 2$	dual numbers	$101_2$
$m = 16$	hexadecimal numbers	$A0_{16}$
$m = 26$	words	"INFORMATIK"

$m$  is called the radix of the representation.

# Assumptions

- keys =  $m$ -adic numbers with same length.

# Assumptions

- keys =  $m$ -adic numbers with same length.
- Procedure  $z$  for the extraction of digit  $k$  in  $\mathcal{O}(1)$  steps.

## Example

$$z_{10}(0, 85) = 5$$

$$z_{10}(1, 85) = 8$$

$$z_{10}(2, 85) = 0$$

# Radix-Exchange-Sort

Keys with radix 2.

Observation: if for some  $k \geq 0$ :

$$z_2(i, x) = z_2(i, y) \text{ for all } i > k$$

and

$$z_2(k, x) < z_2(k, y),$$

then it holds that  $x < y$ .

# Radix-Exchange-Sort

Idea:

- Start with a maximal  $k$ .
- Binary partition the data sets with  $z_2(k, \cdot) = 0$  vs.  $z_2(k, \cdot) = 1$  like with quicksort.
- $k \leftarrow k - 1$ .



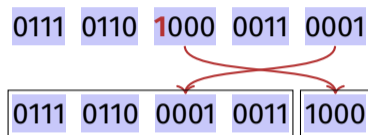
# Radix-Exchange-Sort

0111 0110 1000 0011 0001

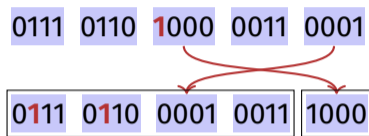
# Radix-Exchange-Sort

0111 0110 1000 0011 0001

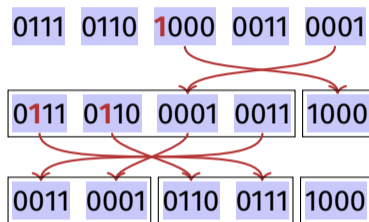
# Radix-Exchange-Sort



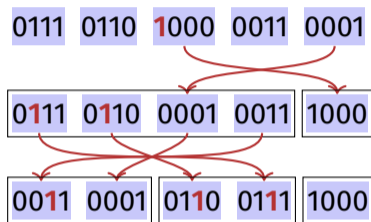
# Radix-Exchange-Sort



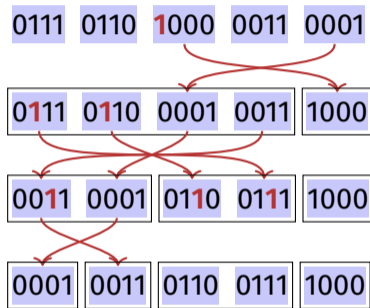
# Radix-Exchange-Sort



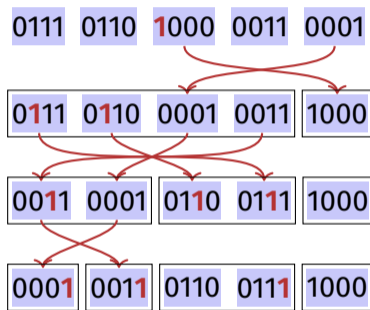
# Radix-Exchange-Sort



# Radix-Exchange-Sort

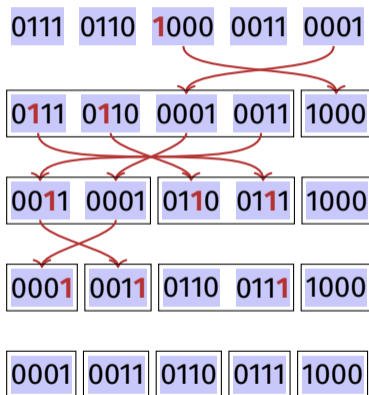


# Radix-Exchange-Sort





# Radix-Exchange-Sort



# Algorithm RadixExchangeSort( $A, l, r, b$ )

**Input:** Array  $A$  with length  $n$ , left and right bounds  $1 \leq l \leq r \leq n$ , bit position  $b$

**Output:** Array  $A$ , sorted in the domain  $[l, r]$  by bits  $[0, \dots, b]$ .

**if**  $l < r$  **and**  $b \geq 0$  **then**

$i \leftarrow l - 1$

$j \leftarrow r + 1$

**repeat**

**repeat**  $i \leftarrow i + 1$  **until**  $z_2(b, A[i]) = 1$  **or**  $i \geq j$

**repeat**  $j \leftarrow j - 1$  **until**  $z_2(b, A[j]) = 0$  **or**  $i \geq j$

**if**  $i < j$  **then** swap( $A[i], A[j]$ )

**until**  $i \geq j$

    RadixExchangeSort( $A, l, i - 1, b - 1$ )

    RadixExchangeSort( $A, i, r, b - 1$ )

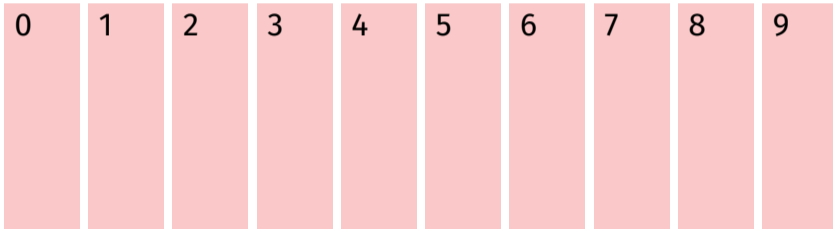
# Analysis

RadixExchangeSort provides recursion with maximal recursion depth = maximal number of digits  $p$ .

Worst case run time  $\mathcal{O}(p \cdot n)$ .

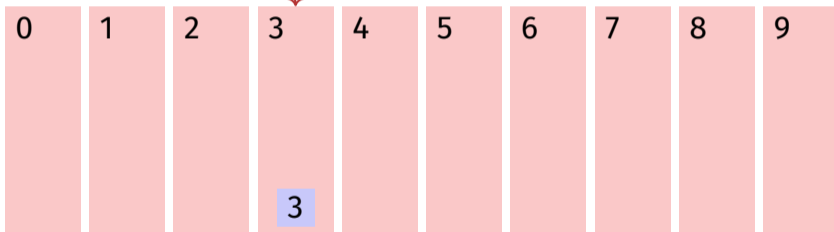
# Bucket Sort

3 8 18 122 121 131 23 21 19 29



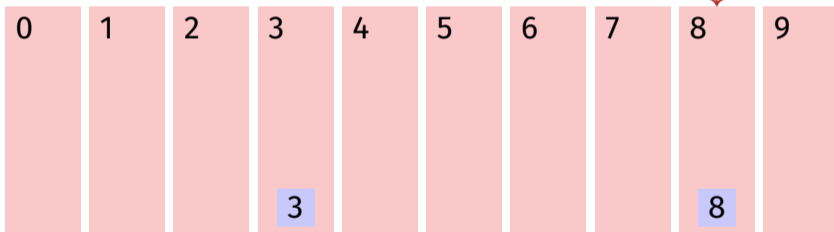
# Bucket Sort

3 8 18 122 121 131 23 21 19 29



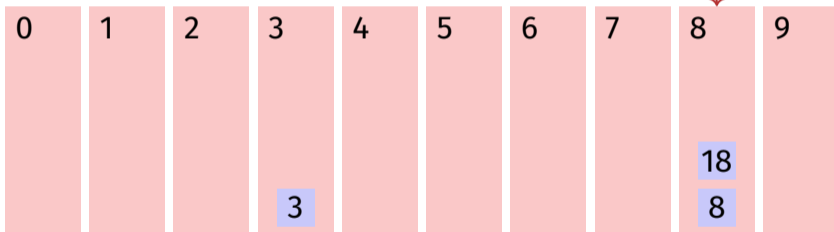
# Bucket Sort

3 8 18 122 121 131 23 21 19 29



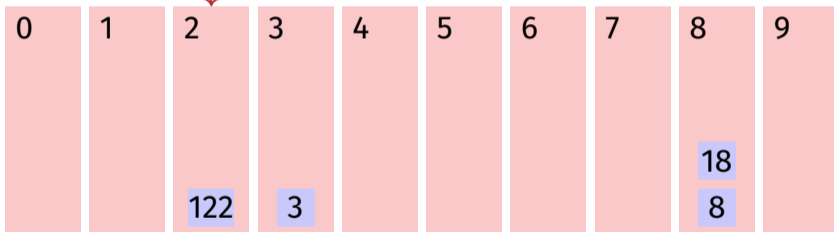
# Bucket Sort

3 8 18 122 121 131 23 21 19 29



# Bucket Sort

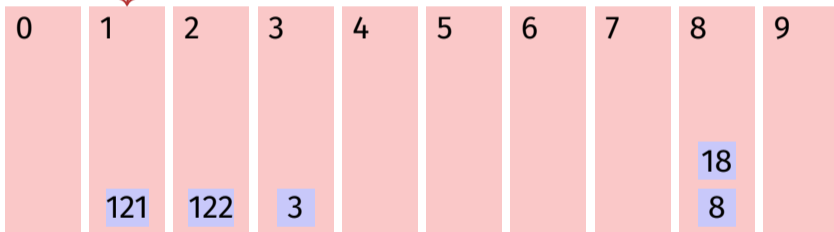
3 8 18 122 121 131 23 21 19 29





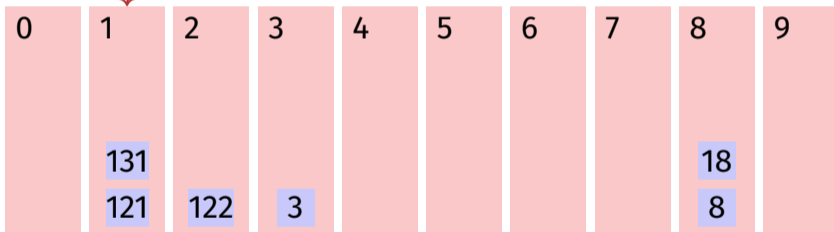
# Bucket Sort

3 8 18 122 121 131 23 21 19 29



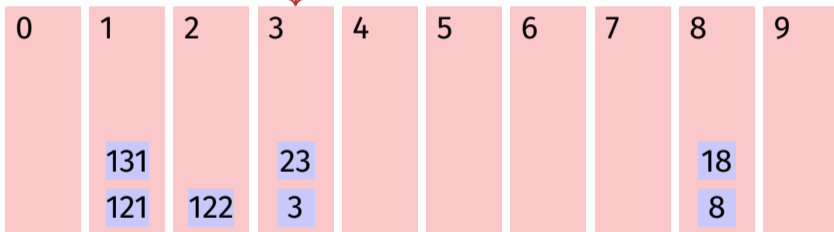
# Bucket Sort

3 8 18 122 121 131 23 21 19 29



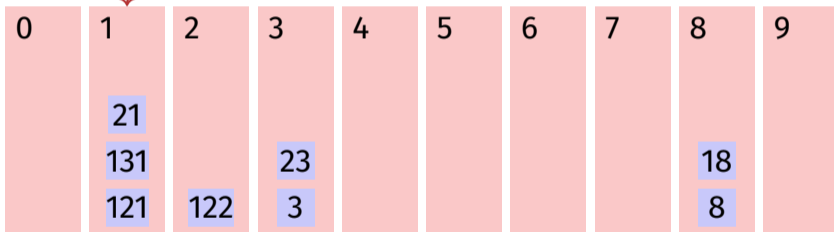
# Bucket Sort

3 8 18 122 121 131 23 21 19 29



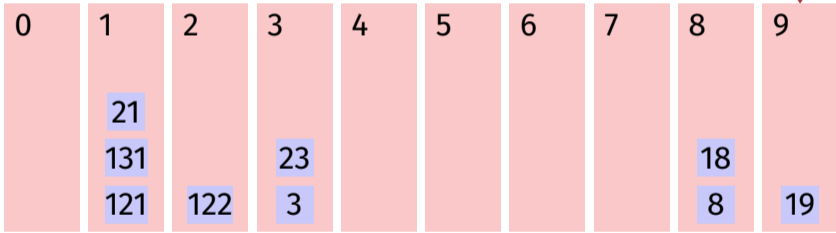
# Bucket Sort

3 8 18 122 121 131 23 21 19 29



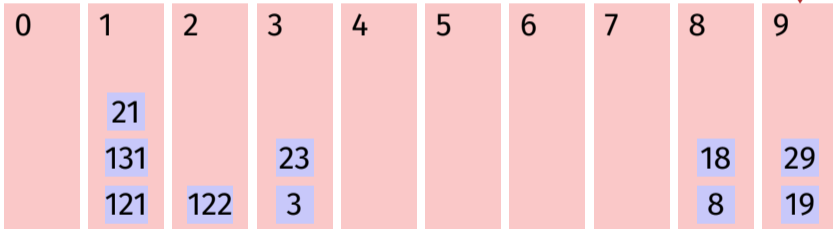
# Bucket Sort

3 8 18 122 121 131 23 21 19 29



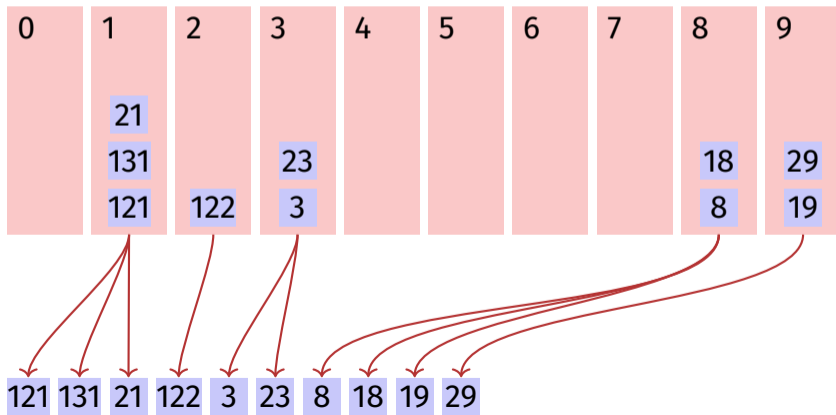
# Bucket Sort

3 8 18 122 121 131 23 21 19 29



# Bucket Sort

3 8 18 122 121 131 23 21 19 29



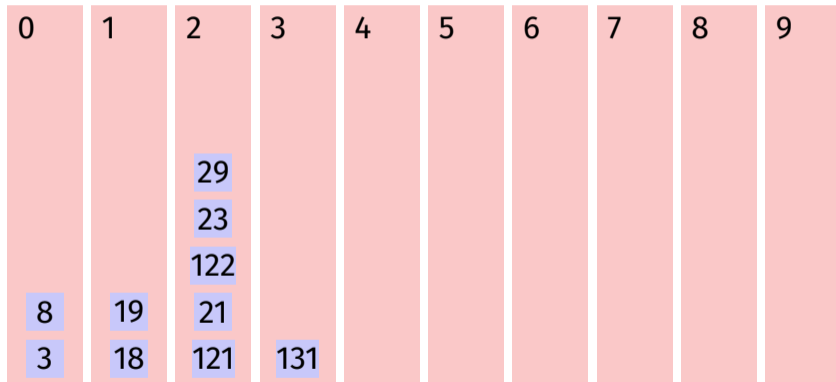
# Bucket Sort

121 131 21 122 3 23 8 18 19 29



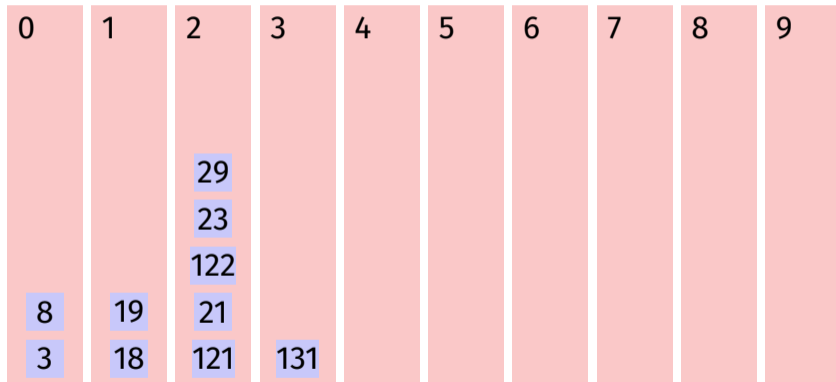
# Bucket Sort

121 131 21 122 3 23 8 18 19 29



# Bucket Sort

121 131 21 122 3 23 8 18 19 29



3 8 18 19 121 21 122 23 29

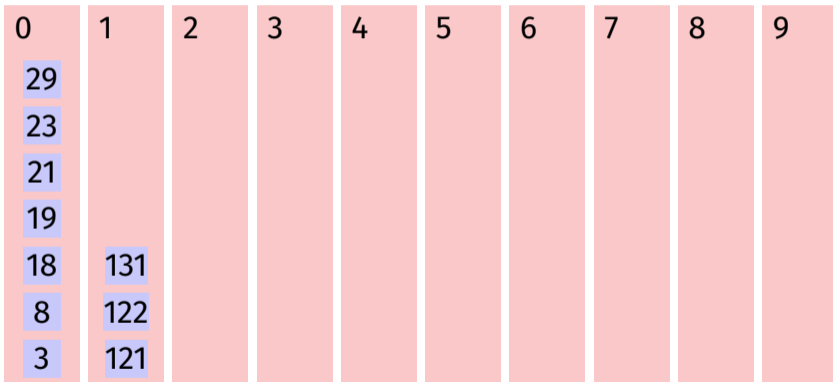
# Bucket Sort

3 8 18 19 121 21 122 23 29



# Bucket Sort

3 8 18 19 121 21 122 23 29



3 8 18 19 21 23 29 121 122 131 😊

# implementation details

Bucket size varies greatly. Possibilities

- Linked list or dynamic array for each digit.
- One array of length  $n$ . compute offsets for each digit in the first iteration.

Assumptions: Input length  $n$ , Number bits / integer:  $k$ , Number Buckets:  $2^b$

Asymptotic running time  $\mathcal{O}(\frac{k}{b} \cdot (n + 2^b))$ .

For Example:  $k = 32, 2^b = 256 : \frac{k}{b} \cdot (n + 2^b) = 4n + 1024$ .

# Bucket Sort – Different Assumption

**Hypothesis:** uniformly distributed data e.g. from  $[0, 1)$

**Input:** Array  $A$  with length  $n$ ,  $A_i \in [0, 1)$ , constant  $M \in \mathbb{N}^+$

**Output:** Sorted array

$k \leftarrow \lceil n/M \rceil$

$B \leftarrow$  new array of  $k$  empty lists

**for**  $i \leftarrow 1$  **to**  $n$  **do**

$B[\lfloor A_i \cdot k \rfloor].\text{append}(A[i])$

**for**  $i \leftarrow 1$  **to**  $k$  **do**

$\text{sort } B[i]$  // e.g. insertion sort, running time  $\mathcal{O}(M^2)$

**return**  $B[0] \circ B[1] \circ \dots \circ B[k]$  // concatenated

Expected asymptotic running time  $\mathcal{O}(n)$  (Proof in Cormen et al, Kap. 8.4)