# Lower Bound for Comparison Based Sorting

# Comparison Based Sorting Algorithms

**Definition**:

A **comparison based sorting algorithm** sorts objects by comparing pairs of them.

**Example**:

- **insertion sort, selection sort, bubble sort** $O(n^2)$
- **shell sort** $O(n^{1.5})$
- **mergesort, quicksort, heapsort** $O(n\log n)$

# Comparison Based Sorting Algorithms

**Theorem**:

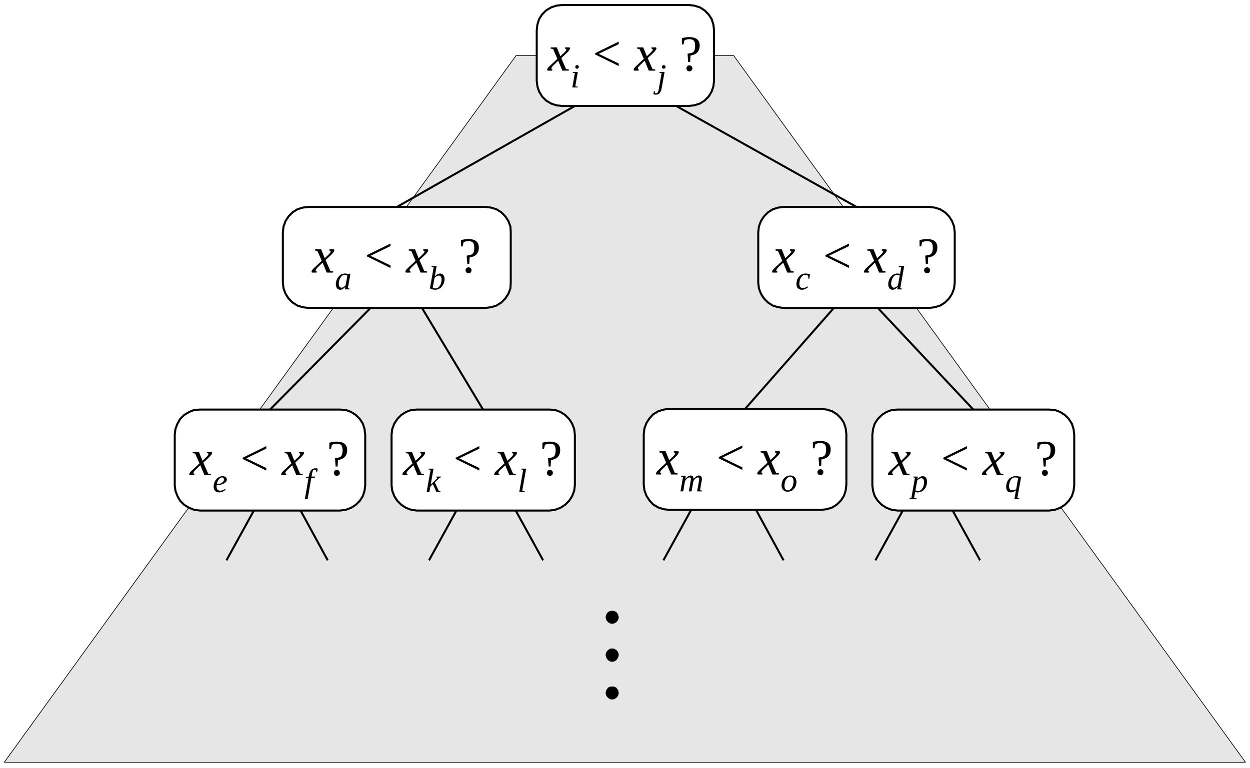
Any comparison based sorting algorithm performs $\Omega(\mathrm{n}\log\mathrm{n})$ comparisons in the **worst case** to sort $\mathrm{n}$ objects.

**In other words**:

For **any comparison based sorting algorithm**, there exists a list $\mathrm{A}[0..\mathrm{n}-1]$ such that the algorithm performs at least $\Omega(\mathrm{n}\log\mathrm{n})$ comparisons to sort $\mathrm{A}$.

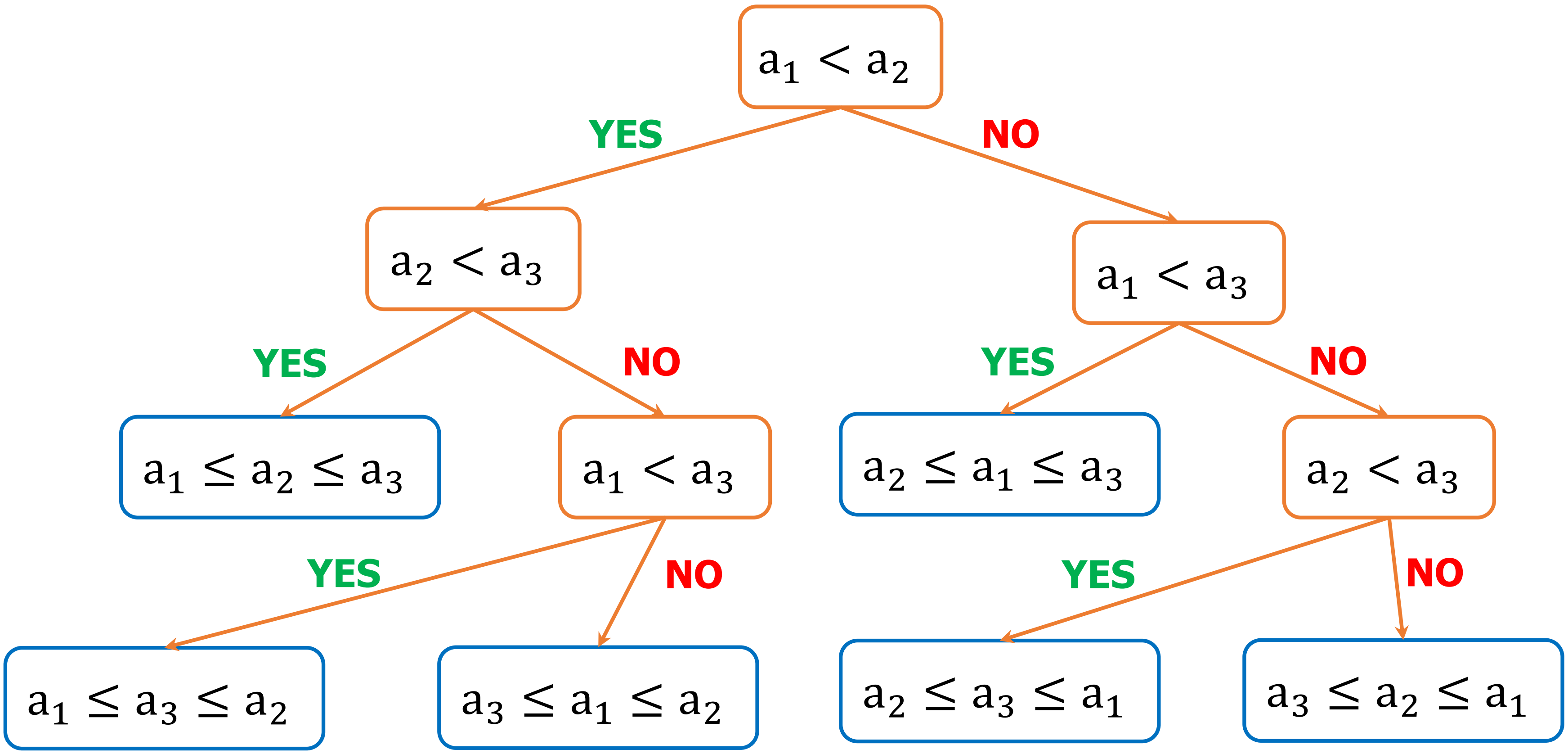
# Decision Tree

- Count only **comparisons**.
- Each possible run of the algorithm corresponds to a **root-to-leaf path** in a **decision tree**



# Decision Tree

**Example:** Sort $a_1, a_2, a_3$



# Decision Tree

- The **height of the decision tree** is a **lower bound** on the running time
- Every **input permutation** must lead to a **separate leaf output**
- If not, some input $\ldots 4 \ldots 5 \ldots$ would have same output ordering as $\ldots 5 \ldots 4 \ldots$ , which would be wrong
- Since there are $n! = 1 \cdot 2 \cdot \cdots \cdot n$ leaves, the **height** is **at least** $\log(n!)$
- Thus, any **comparison-based sorting algorithm** takes at least $\log(n!)$ time

# Comparison Based Sorting Algorithms

**Theorem**:

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

**Proof**:

$$\begin{aligned}
\log(n!) &= \log(1 \cdot 2 \cdot \cdots \cdot n) \\
&= \log 1 + \log 2 + \cdots + \log n \\
&\geq \log \frac{n}{2} + \log\left(\frac{n}{2} + 1\right) + \cdots + \log n \\
&\geq \frac{n}{2} \log \frac{n}{2} = \Omega(n \log n)
\end{aligned}$$

https://stackoverflow.com/questions/2095395/is-logn-Θn-logn

https://www.mcs.sdsmt.edu/ecorwin/cs372/handouts/theta_n_factorial.htm

# Non-Comparison Based Sorting Algorithms

# Sorting Small Integers

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A | 2 | 3 | 2 | 1 | 3 | 2 | 2 | 3 | 2 | 2 | 2 | 1 |

# Sorting Small Integers

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A | **2** | **3** | **2** | **1** | **3** | **2** | **2** | **3** | **2** | **2** | **2** | **1** |

| | 1 | 2 | 3 |
|---|---|---|---|
| **Count** | | | |

# Sorting Small Integers

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A | 2 | 3 | 2 | 1 | 3 | 2 | 2 | 3 | 2 | 2 | 2 | 1 |

| | 1 | 2 | 3 |
|---|---|---|---|
| Count | 2 | 7 | 3 |

# Sorting Small Integers

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A | **2** | **3** | **2** | **1** | **3** | **2** | **2** | **3** | **2** | **2** | **2** | **1** |

| | 1 | 2 | 3 |
|---|---|---|---|
| **Count** | **2** | **7** | **3** |

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A′ | | | | | | | | | | | | |

# Sorting Small Integers

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A | 2 | 3 | 2 | 1 | 3 | 2 | 2 | 3 | 2 | 2 | 2 | 1 |

| | 1 | 2 | 3 |
|---|---|---|---|
| **Count** | 2 | 7 | 3 |

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A′ | 1 | 1 | | | | | | | | | | |

# Sorting Small Integers

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A | 2 | 3 | 2 | 1 | 3 | 2 | 2 | 3 | 2 | 2 | 2 | 1 |

| | 1 | 2 | 3 |
|---|---|---|---|
| **Count** | 2 | 7 | 3 |

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A′ | 1 | 1 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | | | |

# Sorting Small Integers

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A | 2 | 3 | 2 | 1 | 3 | 2 | 2 | 3 | 2 | 2 | 2 | 1 |

| | 1 | 2 | 3 |
|---|---|---|---|
| **Count** | 2 | 7 | 3 |

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

# Sorting Small Integers

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

**we have sorted the array of numbers without comparing them**

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $A'$ | **1** | **1** | **2** | **2** | **2** | **2** | **2** | **2** | **2** | **3** | **3** | **3** |

# Counting Sort

**Idea:**

- Assume that all elements of $A[1 \dots n]$ are integers from $1$ to $M$.
- By a single scan of the array $A$, **count the number of occurrences of each** $1 \leq k \leq M$ in the array $A$ and store it in $Count[k]$.
- Using this information, fill in the sorted array $A'$.

# Counting Sort

**Algorithm** CountingSort(A[1..n])

$\quad \text{Count}[1..M] \leftarrow [0..0]$

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

$\quad\quad \text{Count}\big[A[i]\big] \leftarrow \text{Count}\big[A[i]\big] + 1$

$\quad \text{Pos}[1..M] \leftarrow [0..0]$

$\quad \text{Pos}[1] \leftarrow 1$

$\quad$ **for** $j \leftarrow 2$ **to** $M$ **do**

$\quad\quad \text{Pos}[j] \leftarrow \text{Pos}[j-1] + \text{Count}[j-1]$

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

$\quad\quad A'\Big[\text{Pos}\big[A[i]\big]\Big] \leftarrow A[i]$

$\quad\quad \text{Pos}\big[A[i]\big] \leftarrow \text{Pos}\big[A[i]\big] + 1$

# Counting Sort

**Analysis:**

Provided that all elements of $A[1 \dots n]$ are integers from $1$ to $M$, $\mathrm{CountSort}(A[1..n])$ sorts $A[1 \dots n]$ in $O(n + M)$ time.

**Remark:**

If $M = O(n)$, then the running time of the algorithm is $O(n)$.

# Counting Sort

**Example:** sort the list of integers

$$10, 1000\ 1, 10, 10000, 100, 1000, 1, 1000, 10000$$

using **counting sort**.

**What is $\mathrm{n}$ and $\mathrm{M}$?**

- $\mathrm{n} = 10$
- $\mathrm{M} = 10000$

# Radix Sort

# Radix Sort

- Radix sort (a.k.a. bin sort) is a **fast distribution sorting algorithm** that orders keys by examining the **individual components** (**digits**) of the keys instead of **comparing or counting the keys themselves.**

- The individual digits of the keys are observed **from least significant to most significant**.

- Radix sort can also be used to sort strings, floating-point values

# Radix Sort : Example

- **Bins** are used to store various keys based on **individual column values**
- Consider an **array of positive integers**
- We use **10 bins**, one for each digit

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 23 | 10 | 18 | 51 | 5 | 13 | 31 | 54 | 48 | 62 | 29 | 8 | 37 |

# Distribute the Keys

- The process starts by **distributing the keys** among the various **bins**:
  - Based on the **digits in the ones column**
  - Stored **in the order** they occur in the array

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 23 | 10 | 18 | 51 | 5 | 13 | 31 | 54 | 48 | 62 | 29 | 8 | 37 |

# Distribute the Keys

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 23 | 10 | 18 | 51 | 5 | 13 | 31 | 54 | 48 | 62 | 29 | 8 | 37 |

| | |
|---|---|
| **bin 0** | 10 |
| **bin 1** | 51 31 |
| **bin 2** | 62 |
| **bin 3** | 23 13 |
| **bin 4** | 54 |
| **bin 5** | 5 |
| **bin 6** | |
| **bin 7** | 37 |
| **bin 8** | 18 48 8 |
| **bin 9** | 29 |

# Gather the Keys

- The keys are then **gathered back into the array** **one bin at a time**:
  - Start with the **bin 0** and continue **in bin order**
  - Gathered **without rearranging**

# Gather the Keys

| | | | |
|---|---|---|---|
| **bin 0** | 10 | **bin 5** | 5 |
| **bin 1** | 51 31 | **bin 6** | |
| **bin 2** | 62 | **bin 7** | 37 |
| **bin 3** | 23 13 | **bin 8** | 18 48 8 |
| **bin 4** | 54 | **bin 9** | 29 |

# Gather the Keys

| | | | |
|---|---|---|---|
| **bin 0** | 10 | **bin 5** | 5 |
| **bin 1** | 51 31 | **bin 6** | |
| **bin 2** | 62 | **bin 7** | 37 |
| **bin 3** | 23 13 | **bin 8** | 18 48 8 |
| **bin 4** | 54 | **bin 9** | 29 |

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **10** | | | | | | | | | | | | |

# Gather the Keys

| | | | |
|---|---|---|---|
| **bin 0** | 10 | **bin 5** | 5 |
| **bin 1** | 51 31 | **bin 6** | |
| **bin 2** | 62 | **bin 7** | 37 |
| **bin 3** | 23 13 | **bin 8** | 18 48 8 |
| **bin 4** | 54 | **bin 9** | 29 |

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **10** | **51** | **31** | | | | | | | | | | |

# Gather the Keys

| | |
|---|---|
| **bin 0** | 10 |
| **bin 1** | 51 31 |
| **bin 2** | 62 |
| **bin 3** | 23 13 |
| **bin 4** | 54 |
| **bin 5** | 5 |
| **bin 6** | |
| **bin 7** | 37 |
| **bin 8** | 18 48 8 |
| **bin 9** | 29 |

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **10** | **51** | **31** | **62** | | | | | | | | | |

# Gather the Keys

| | |
|---|---|
| **bin 0** | 10 |
| **bin 1** | 51 31 |
| **bin 2** | 62 |
| **bin 3** | 23 13 |
| **bin 4** | 54 |
| **bin 5** | 5 |
| **bin 6** | |
| **bin 7** | 37 |
| **bin 8** | 18 48 8 |
| **bin 9** | 29 |

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **10** | **51** | **31** | **62** | **23** | **13** | | | | | | | |

# Gather the Keys

| | |
|---|---|
| **bin 0** | 10 |
| **bin 1** | 51 31 |
| **bin 2** | 62 |
| **bin 3** | 23 13 |
| **bin 4** | 54 |
| **bin 5** | 5 |
| **bin 6** | |
| **bin 7** | 37 |
| **bin 8** | 18 48 8 |
| **bin 9** | 29 |

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 10 | 51 | 31 | 62 | 23 | 13 | 54 | | | | | | |

# Gather the Keys

| | | | |
|---|---|---|---|
| **bin 0** | 10 | **bin 5** | 5 |
| **bin 1** | 51 31 | **bin 6** | |
| **bin 2** | 62 | **bin 7** | 37 |
| **bin 3** | 23 13 | **bin 8** | 18 48 8 |
| **bin 4** | 54 | **bin 9** | 29 |

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 10 | 51 | 31 | 62 | 23 | 13 | 54 | 5 | | | | | |

# Gather the Keys

| | |
|---|---|
| **bin 0** | 10 |
| **bin 1** | 51 31 |
| **bin 2** | 62 |
| **bin 3** | 23 13 |
| **bin 4** | 54 |
| **bin 5** | 5 |
| **bin 6** | |
| **bin 7** | 37 |
| **bin 8** | 18 48 8 |
| **bin 9** | 29 |

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 10 | 51 | 31 | 62 | 23 | 13 | 54 | 5 | 37 | | | | |

# Gather the Keys

| | |
|---|---|
| **bin 0** | 10 |
| **bin 1** | 51 31 |
| **bin 2** | 62 |
| **bin 3** | 23 13 |
| **bin 4** | 54 |
| **bin 5** | 5 |
| **bin 6** | |
| **bin 7** | 37 |
| **bin 8** | 18 48 8 |
| **bin 9** | 29 |

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 10 | 51 | 31 | 62 | 23 | 13 | 54 | 5 | 37 | 18 | 48 | 8 | |

# Gather the Keys

| | | | |
|---|---|---|---|
| **bin 0** | 10 | **bin 5** | 5 |
| **bin 1** | 51 31 | **bin 6** | |
| **bin 2** | 62 | **bin 7** | 37 |
| **bin 3** | 23 13 | **bin 8** | 18 48 8 |
| **bin 4** | 54 | **bin 9** | 29 |

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 10 | 51 | 31 | 62 | 23 | 13 | 54 | 5 | 37 | 18 | 48 | 8 | 29 |

# Repeat the Process

- Repeat the process, but this time based on **the tens column**

# Repeat the Process

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 10 | 51 | 31 | 62 | 23 | 13 | 54 | 5 | 37 | 18 | 48 | 8 | 29 |

| | |
|---|---|
| bin 0 | 5 8 |
| bin 1 | 10 13 18 |
| bin 2 | 23 29 |
| bin 3 | 31 37 |
| bin 4 | 48 |
| bin 5 | 51 |
| bin 6 | 62 |
| bin 7 | |
| bin 8 | |
| bin 9 | |

# Gather the Keys

| | |
|---|---|
| bin 0 | 5 8 |
| bin 1 | 10 13 18 |
| bin 2 | 23 29 |
| bin 3 | 31 37 |
| bin 4 | 48 |
| bin 5 | 51 54 |
| bin 6 | 62 |
| bin 7 | |
| bin 8 | |
| bin 9 | |

# Gather the Keys

| | | | |
|---|---|---|---|
| **bin 0** | 5 8 | **bin 5** | 51 54 |
| **bin 1** | 10 13 18 | **bin 6** | 62 |
| **bin 2** | 23 29 | **bin 7** | |
| **bin 3** | 31 37 | **bin 8** | |
| **bin 4** | 48 | **bin 9** | |

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 5 | 8 | 10 | 13 | 18 | 23 | 29 | 31 | 37 | 48 | 51 | 54 | 62 |

# Radix Sort Analysis

- Assume:
  - a **sequence of $n$ keys**
  - each key consists of $d$ **components**
  - each component contains a value **between $0$ and $m$**
  - use **queue**

# Radix Sort Analysis

- The construction of the **array and queues:** $O(m)$
- The **distribution** and **gathering**: $O(d \cdot n)$
  - **distribute** the $n$ keys across $m$ bins: $O(n)$
  - **gather** the $n$ keys back into the array: $O(n)$
- **Total time**: $O(m + d \cdot n)$
  - In practice $m$ and $d$ are constant
  - When sorting integers, the time only depends on the number of keys: $O(n)$



https://www.youtube.com/watch?v=XiuSW_mEn7g

# Exercises

1. Apply the **counting sort** to the list of integers

$$\mathbf{4, 1, 3, 4, 3, 4, 1, 1, 2, 3, 4, 3, 2, 1}$$

2. Apply the **radix sort** to the list of integers

$$\mathbf{329, 3, 457, 657, 92, 839, 87, 436, 720, 355, 444}$$

https://www.youtube.com/watch?v=XiuSW_mEn7g