

# Divide and Conquer

---

Sorting & Inversions



# Sorting Problem

---

- **Input:** A set of  $n$  integers
  - $x_1, x_2, x_3, \dots, x_n$
- **Output:** The same set of  $n$  integers in ascending order.



How many sorting algorithms  
you know?

---



# Insertion Sort

---

- **Input:** A set of  $n$  integers
  - $x_1, x_2, x_3, \dots, x_n$
- **Output:** The same set of  $n$  integers in **ascending order**.
- **Plan 1:**
  - Try **Insertion Sort**: fix the output one by one.
  - How fast is it?

1	10	5	26	3	4	16	4	2



# Insertion Sort

---

- **Input:** A set of  $n$  integers
  - $x_1, x_2, x_3, \dots, x_n$
- **Output:** The same set of  $n$  integers in **ascending order**.
- **Plan 1:**
  - Try **Insertion Sort**: fix the output one by one.
  - How fast is it?

1	10	5	26	3	4	16	4	2
---	----	---	----	---	---	----	---	---

1								
---	--	--	--	--	--	--	--	--



# Insertion Sort

---

- **Input:** A set of  $n$  integers
  - $x_1, x_2, x_3, \dots, x_n$
- **Output:** The same set of  $n$  integers in **ascending order**.
- **Plan 1:**
  - Try **Insertion Sort**: fix the output one by one.
  - How fast is it?

1	10	5	26	3	4	16	4	2
---	----	---	----	---	---	----	---	---

1	10							
---	----	--	--	--	--	--	--	--



# Insertion Sort

---

- **Input:** A set of  $n$  integers
  - $x_1, x_2, x_3, \dots, x_n$
- **Output:** The same set of  $n$  integers in **ascending order**.
- Plan 1:
  - Try **Insertion Sort**: fix the output one by one.
  - How fast is it?

1	10	5	26	3	4	16	4	2
---	----	---	----	---	---	----	---	---

1	5	10						
---	---	----	--	--	--	--	--	--



# Insertion Sort

---

- **Input:** A set of  $n$  integers
  - $x_1, x_2, x_3, \dots, x_n$
- **Output:** The same set of  $n$  integers in **ascending order**.
- Plan 1:
  - Try **Insertion Sort**: fix the output one by one.
  - How fast is it?

1	10	5	26	3	4	16	4	2
---	----	---	----	---	---	----	---	---

1	5	10	26					
---	---	----	----	--	--	--	--	--



# Insertion Sort

---

- **Input:** A set of  $n$  integers
  - $x_1, x_2, x_3, \dots, x_n$
- **Output:** The same set of  $n$  integers in **ascending order**.
- **Plan 1:**
  - Try **Insertion Sort**: fix the output one by one.
  - How fast is it?

1	10	5	26	3	4	16	4	2
---	----	---	----	---	---	----	---	---

1	3	5	10	26				
---	---	---	----	----	--	--	--	--



# Insertion Sort

---

- **Input:** A set of  $n$  integers
  - $x_1, x_2, x_3, \dots, x_n$
- **Output:** The same set of  $n$  integers in **ascending order**.
- **Plan 1:**
  - Try **Insertion Sort**: fix the output one by one.
  - How fast is it?

1	10	5	26	3	4	16	4	2
---	----	---	----	---	---	----	---	---

1	3	4	5	10	26			
---	---	---	---	----	----	--	--	--



# Insertion Sort

---

- **Input:** A set of  $n$  integers
  - $x_1, x_2, x_3, \dots, x_n$
- **Output:** The same set of  $n$  integers in **ascending order**.
- Plan 1:
  - Try **Insertion Sort**: fix the output one by one.
  - How fast is it?

1	10	5	26	3	4	16	4	2
---	----	---	----	---	---	----	---	---

1	3	4	5	10	16	26		
---	---	---	---	----	----	----	--	--



# Insertion Sort

---

- **Input:** A set of  $n$  integers
  - $x_1, x_2, x_3, \dots, x_n$
- **Output:** The same set of  $n$  integers in **ascending order**.
- **Plan 1:**
  - Try **Insertion Sort**: fix the output one by one.
  - How fast is it?

1	10	5	26	3	4	16	4	2
---	----	---	----	---	---	----	---	---

1	3	4	4	5	10	16	26	
---	---	---	---	---	----	----	----	--



# Insertion Sort

---

- **Input:** A set of  $n$  integers
  - $x_1, x_2, x_3, \dots, x_n$
- **Output:** The same set of  $n$  integers in **ascending order**.
- **Plan 1:**
  - Try **Insertion Sort**: fix the output one by one.
  - How fast is it?

1	10	5	26	3	4	16	4	2
---	----	---	----	---	---	----	---	---

1	2	3	4	4	5	10	16	26
---	---	---	---	---	---	----	----	----



# Insertion Sort

- **Input:** A set of  $n$  integers
  - $x_1, x_2, x_3, \dots, x_n$
- **Output:** The same set of  $n$  integers in **ascending order**.
- **Plan 1:**
  - Try **Insertion Sort**: fix the output one by one.
  - How fast is it?

How many number  
we should insert?

1	10	5	26	3	4	16	4	2
---	----	---	----	---	---	----	---	---

1	2	3	4	4	5	10	16	26
---	---	---	---	---	---	----	----	----

How long it takes to  
insert one number?



# Insertion Sort

- **Input:** A set of  $n$  integers
  - $x_1, x_2, x_3, \dots, x_n$
- **Output:** The same set of  $n$  integers in **ascending order**.
- Plan 1:
  - Try **Insertion Sort**: fix the output one by one.
  - How fast is it?

$n$  numbers

1	10	5	26	3	4	16	4	2
---	----	---	----	---	---	----	---	---

1	2	3	4	4	5	10	16	26
---	---	---	---	---	---	----	----	----

At most  $n$  operations



# Don't forget the correctness!

---

- Is it **correct**?
- How to **prove** it!
  - Using Induction?



# Prove correctness by induction.

---

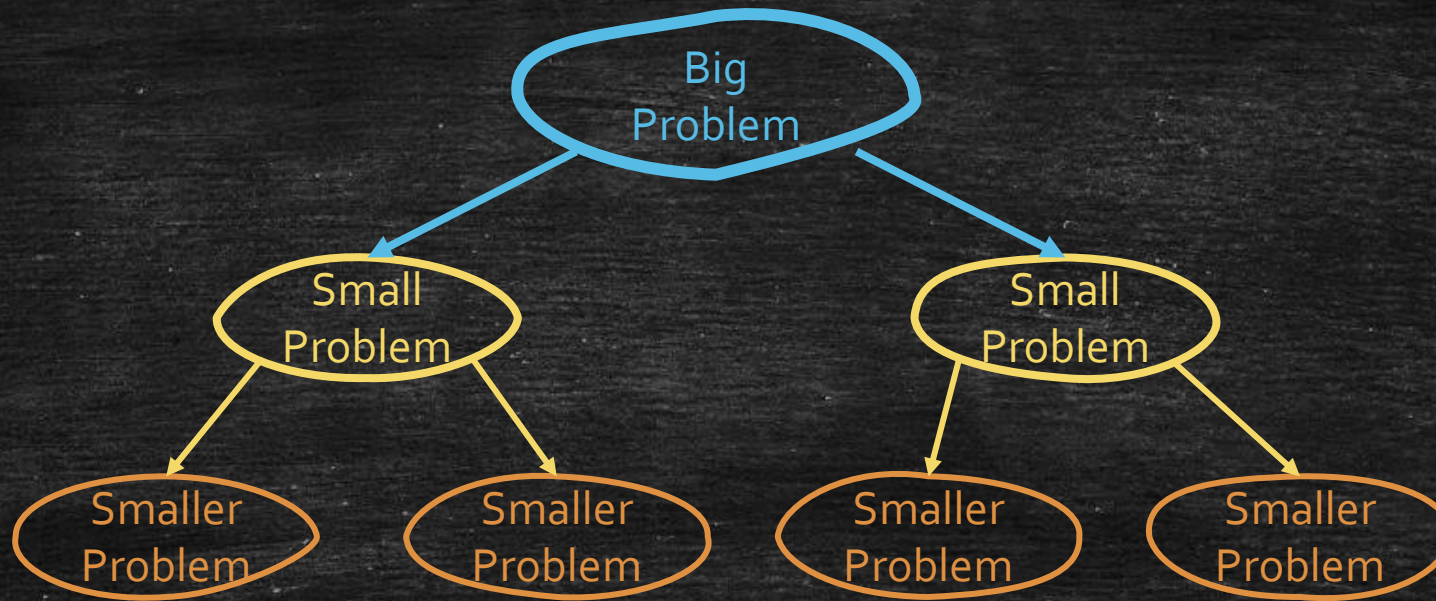
- Base Case

- If we only insert one integer, it is sorted.

- Induction

- Assume the list is sorted after the  $k - 1$ -th insertion.
- Prove the list is sorted after the  $k$ -th insertion.
- Assume the integer  $j$  is inserted at location  $i$ .
  - $j \geq a_i \geq a_{\leq i}$ .
  - $j < a_{i+1} \leq a_{\geq i+1}$ .





Ok! Let's move to divide and conquer!

---



# Divide and Conquer

---

- Recall the divide and conquer
- **Divide**
  - Divide the problem into small size subproblems.
- **Recurse**
  - Solve the small size subproblems.
- **Combine**
  - Combine the output of small size subproblems to get the answer of the original problem.
- **Basic solver**
  - If the problem size is small enough, we should solve it directly.



# Merge sort

---

- **Input:** A set of  $n$  integers
  - $x_1, x_2, x_3, \dots, x_n$
- **Output:** The same set of  $n$  integers in **ascending order**.
- Plan 2: Divide and Conquer (Merge Sort)
  - **Divide:** Divide the input into two subsets:
    - $x_1, x_2, \dots, x_{n/2}; x_{n/2+1}, x_{n/2+2}, \dots, x_n$
  - **Recurse:** Sort two subsets (smaller size problems).
    - Let  $y_1, y_2, \dots, y_{n/2}; y_{n/2+1}, y_{n/2+2}, \dots, y_n$  be the output (sorted list).
  - **Combine:** Merge two sorted lists to one long sorted list.



# Merge Sort

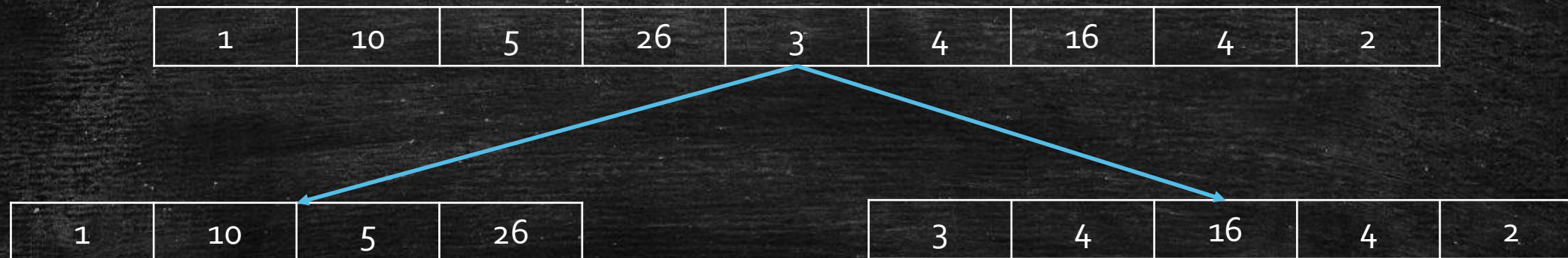
---

1	10	5	26	3	4	16	4	2
---	----	---	----	---	---	----	---	---



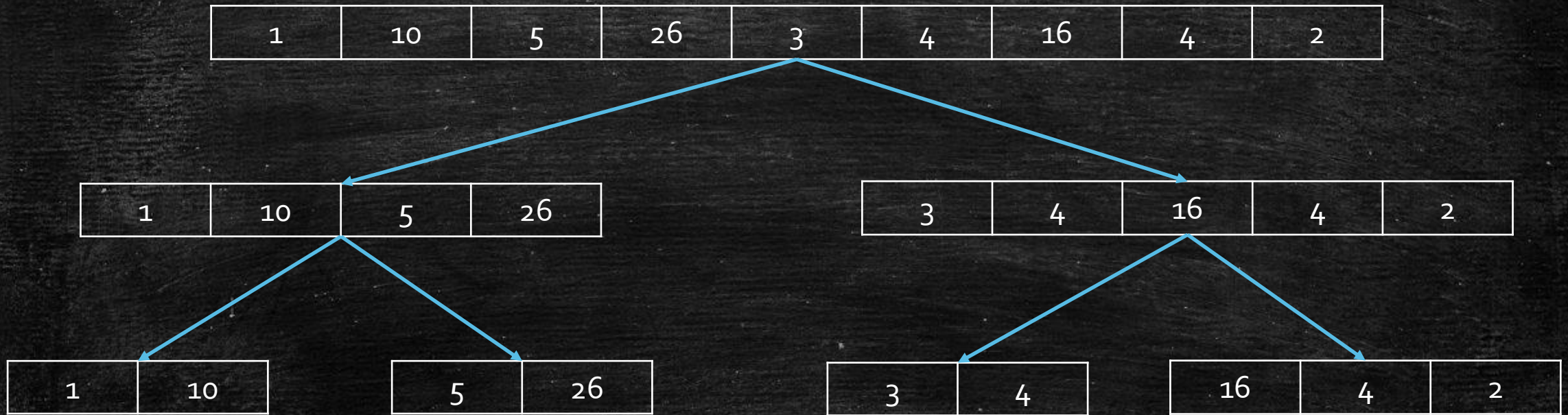
# Merge Sort

---



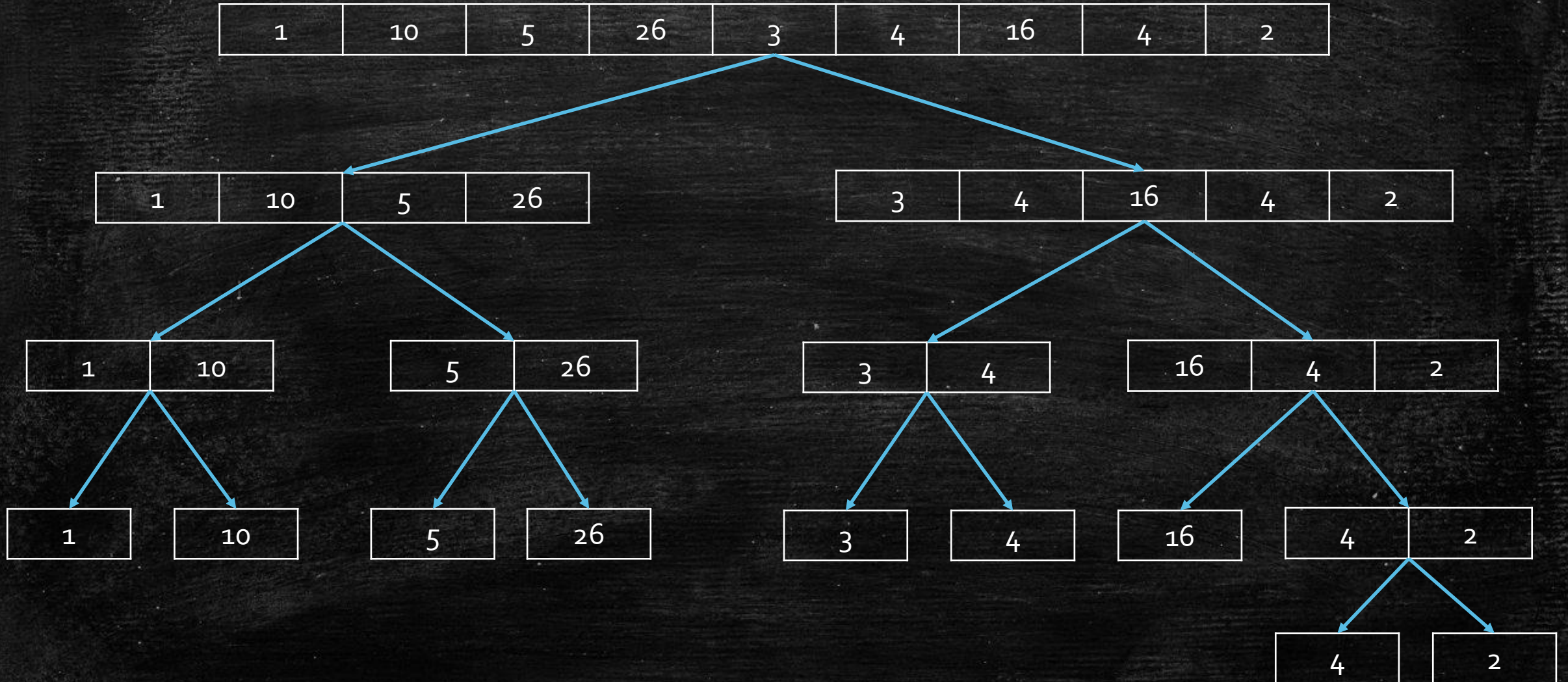


# Merge Sort



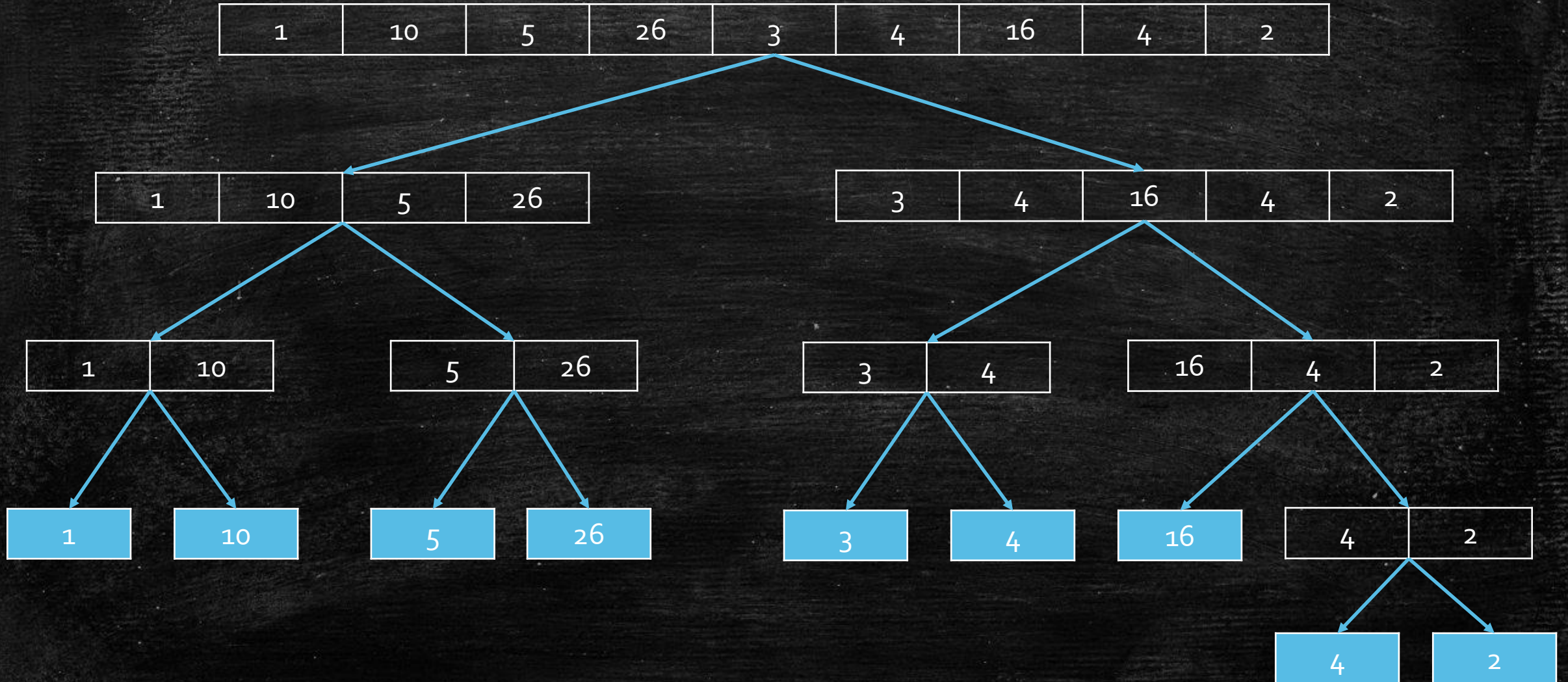


# Merge Sort



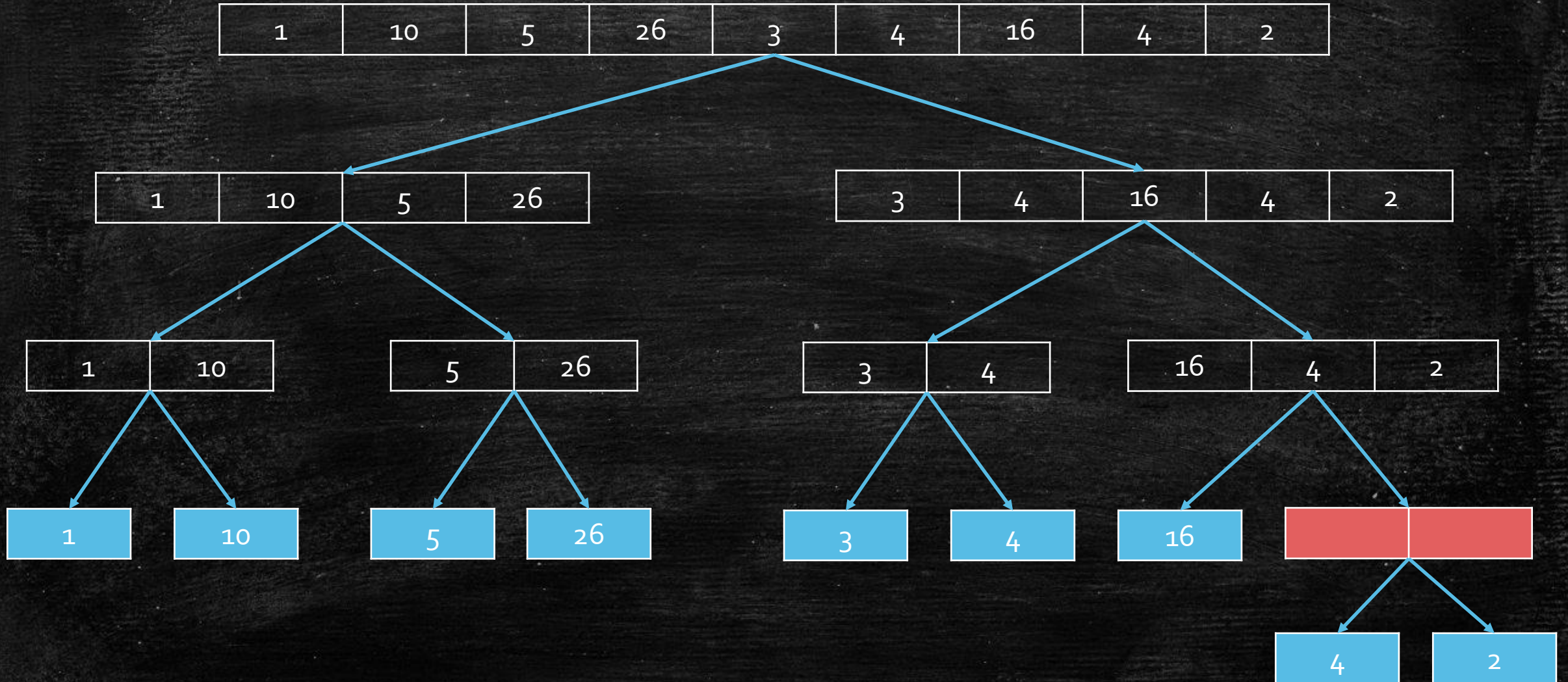


# Merge Sort



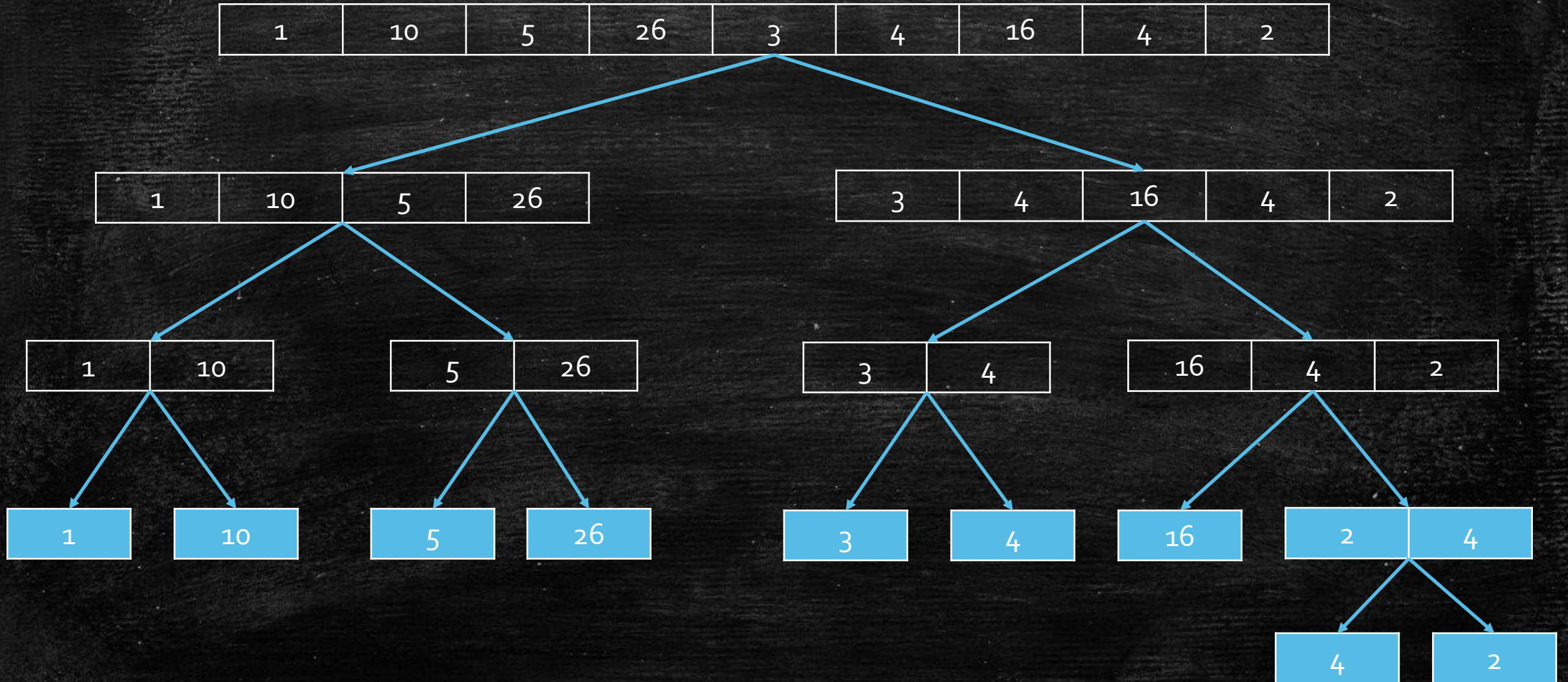


# Merge Sort



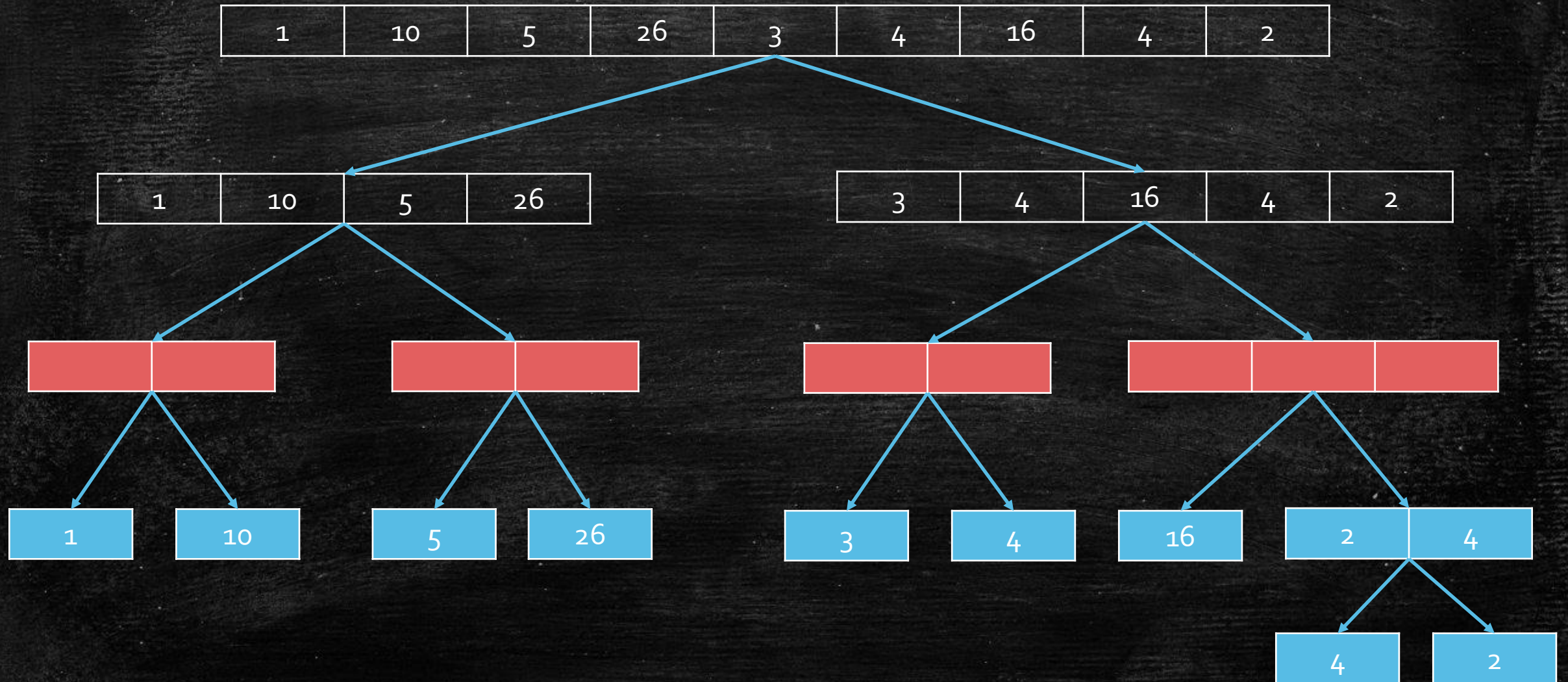


# Merge Sort



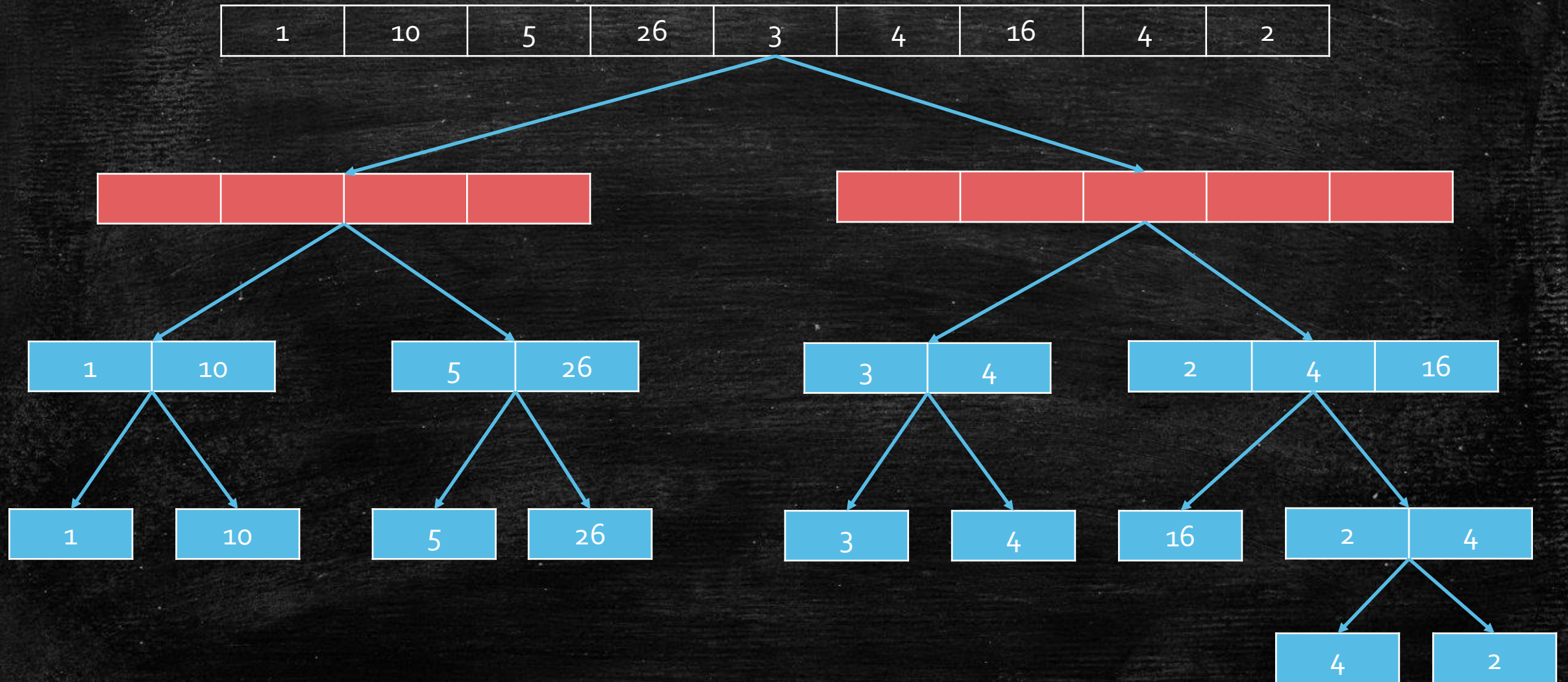


# Merge Sort



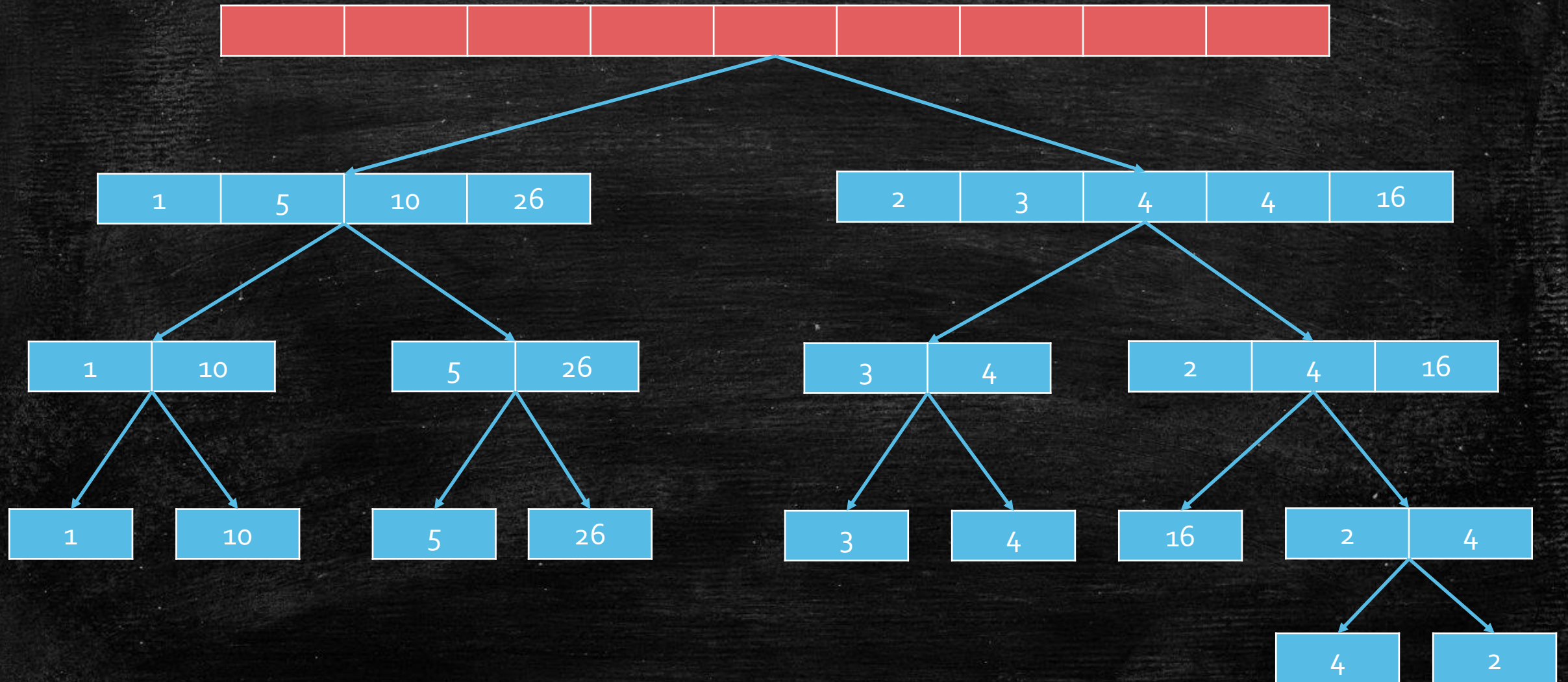


# Merge Sort



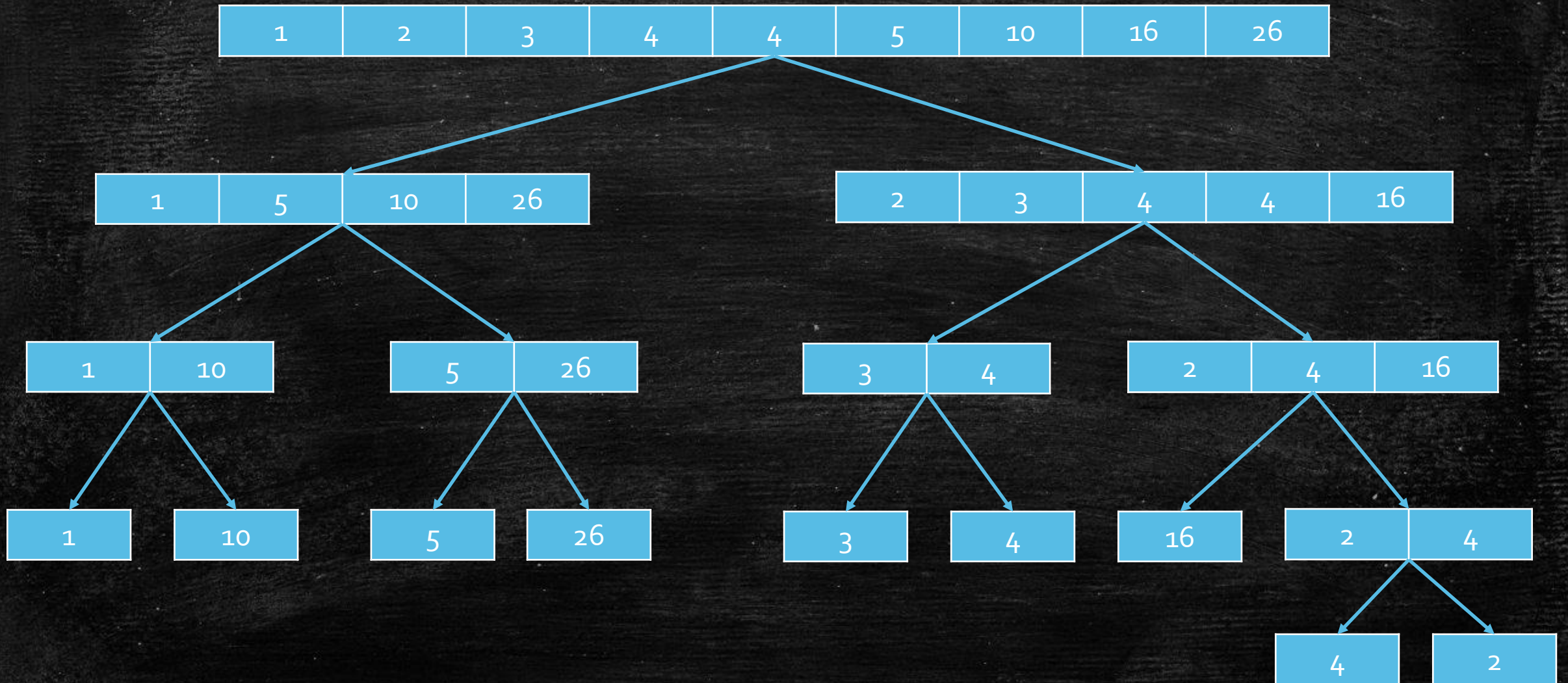


# Merge Sort





# Merge Sort





What is the next?

---



# The remaining questions

---

- How to merge two sorted lists?
- How fast we can make it?



# Merge two sorted lists

---

- **Input:** two sorted lists  $A = a_1, a_2, \dots, a_n$ ,  $B = b_1, b_2, \dots, b_m$
- **Output:** a sorted list  $C$
- Straightforward Plan
  - Insert each  $b$  into  $A$  iteratively.
  - Time:
    - $m$  integer to insert.
    - $n$  times for each insertion.
  - Totally:  $O(mn)$
  - Merge Sort:
    - $T(n) = 2T\left(\frac{n}{2}\right) + O(n^2/4)$

Do we really need so many comparisons?



# Merge two sorted lists

---

- **Input:** two sorted lists  $A = a_1, a_2, \dots, a_n$ ,  $B = b_1, b_2, \dots, b_m$
- **Output:** a sorted list  $C$
- Smarter Plan
  - Maintain 2 pointers  $i = 1, j = 1$
  - Repeat
    - Append  $\min\{a_i, b_j\}$  to  $C$
    - If  $a_i$  is smaller, then move  $i$  to  $i + 1$ ; If  $b_j$  is smaller, then move  $j$  to  $j + 1$ .
    - Break if  $i > n$  or  $j > m$
  - Append the remainder of the non-empty list to  $C$



# Example

---

1	5	10	26
---	---	----	----



2	3	4	4	16
---	---	---	---	----



--	--	--	--	--	--	--	--	--

- Plan
  - Maintain 2 pointers  $i = 1, j = 1$
  - Repeat
    - Append  $\min\{a_i, b_j\}$  to  $C$
    - If  $a_i$  is smaller, then move  $i$  to  $i + 1$ ; If  $b_j$  is smaller, then move  $j$  to  $j + 1$ .
    - Break if  $i > n$  or  $j > m$
  - Append the remainder of the non-empty list to  $C$



# Example

1	5	10	26
---	---	----	----



2	3	4	4	16
---	---	---	---	----



1								
---	--	--	--	--	--	--	--	--

- Plan
  - Maintain 2 pointers  $i = 1, j = 1$
  - Repeat
    - Append  $\min\{a_i, b_j\}$  to  $C$
    - If  $a_i$  is smaller, then move  $i$  to  $i + 1$ ; If  $b_j$  is smaller, then move  $j$  to  $j + 1$ .
    - Break if  $i > n$  or  $j > m$
  - Append the remainder of the non-empty list to  $C$



# Example

---

1	5	10	26
---	---	----	----



2	3	4	4	16
---	---	---	---	----



1	2							
---	---	--	--	--	--	--	--	--

- Plan
  - Maintain 2 pointers  $i = 1, j = 1$
  - Repeat
    - Append  $\min\{a_i, b_j\}$  to  $C$
    - If  $a_i$  is smaller, then move  $i$  to  $i + 1$ ; If  $b_j$  is smaller, then move  $j$  to  $j + 1$ .
    - Break if  $i > n$  or  $j > m$
  - Append the remainder of the non-empty list to  $C$



# Example

---

1	5	10	26
---	---	----	----



2	3	4	4	16
---	---	---	---	----



1	2	3						
---	---	---	--	--	--	--	--	--

- Plan
  - Maintain 2 pointers  $i = 1, j = 1$
  - Repeat
    - Append  $\min\{a_i, b_j\}$  to  $C$
    - If  $a_i$  is smaller, then move  $i$  to  $i + 1$ ; If  $b_j$  is smaller, then move  $j$  to  $j + 1$ .
    - Break if  $i > n$  or  $j > m$
  - Append the remainder of the non-empty list to  $C$



# Example

---

1	5	10	26
---	---	----	----



2	3	4	4	16
---	---	---	---	----



1	2	3	4					
---	---	---	---	--	--	--	--	--

- Plan
  - Maintain 2 pointers  $i = 1, j = 1$
  - Repeat
    - Append  $\min\{a_i, b_j\}$  to  $C$
    - If  $a_i$  is smaller, then move  $i$  to  $i + 1$ ; If  $b_j$  is smaller, then move  $j$  to  $j + 1$ .
    - Break if  $i > n$  or  $j > m$
  - Append the remainder of the non-empty list to  $C$



# Example

---

1	5	10	26
---	---	----	----



2	3	4	4	16
---	---	---	---	----



1	2	3	4	4				
---	---	---	---	---	--	--	--	--

- Plan
  - Maintain 2 pointers  $i = 1, j = 1$
  - Repeat
    - Append  $\min\{a_i, b_j\}$  to  $C$
    - If  $a_i$  is smaller, then move  $i$  to  $i + 1$ ; If  $b_j$  is smaller, then move  $j$  to  $j + 1$ .
    - Break if  $i > n$  or  $j > m$
  - Append the remainder of the non-empty list to  $C$



# Example

---

1	5	10	26
---	---	----	----



2	3	4	4	16
---	---	---	---	----



1	2	3	4	4	5			
---	---	---	---	---	---	--	--	--

- Plan
  - Maintain 2 pointers  $i = 1, j = 1$
  - Repeat
    - Append  $\min\{a_i, b_j\}$  to  $C$
    - If  $a_i$  is smaller, then move  $i$  to  $i + 1$ ; If  $b_j$  is smaller, then move  $j$  to  $j + 1$ .
    - Break if  $i > n$  or  $j > m$
  - Append the remainder of the non-empty list to  $C$ .



# Example

1	5	10	26
---	---	----	----



2	3	4	4	16
---	---	---	---	----



1	2	3	4	4	5	10		
---	---	---	---	---	---	----	--	--

- Plan
  - Maintain 2 pointers  $i = 1, j = 1$
  - Repeat
    - Append  $\min\{a_i, b_j\}$  to  $C$
    - If  $a_i$  is smaller, then move  $i$  to  $i + 1$ ; If  $b_j$  is smaller, then move  $j$  to  $j + 1$ .
    - Break if  $i > n$  or  $j > m$
  - Append the remainder of the non-empty list to  $C$



# Example

1	5	10	26
---	---	----	----



2	3	4	4	16
---	---	---	---	----



1	2	3	4	4	5	10	16	
---	---	---	---	---	---	----	----	--

- Plan
  - Maintain 2 pointers  $i = 1, j = 1$
  - Repeat
    - Append  $\min\{a_i, b_j\}$  to  $C$
    - If  $a_i$  is smaller, then move  $i$  to  $i + 1$ ; If  $b_j$  is smaller, then move  $j$  to  $j + 1$ .
    - Break if  $i > n$  or  $j > m$
  - Append the reminder of the non-empty list to  $C$



# Example

1	5	10	26
---	---	----	----

2	3	4	4	16
---	---	---	---	----



1	2	3	4	4	5	10	16	26
---	---	---	---	---	---	----	----	----

- Plan
  - Maintain 2 pointers  $i = 1, j = 1$
  - Repeat
    - Append  $\min\{a_i, b_j\}$  to  $C$
    - If  $a_i$  is smaller, then move  $i$  to  $i + 1$ ; If  $b_j$  is smaller, then move  $j$  to  $j + 1$ .
    - Break if  $i > n$  or  $j > m$
  - Append the remainder of the non-empty list to  $C$ .



# Is it correct?

---

Prove by induction.



# Is Merge Sort correct?

---

Prove by induction.



# How fast is the algorithm?

---

- Plan

- Maintain 2 pointers  $i = 1, j = 1$
- Repeat
  - Append  $\min\{a_i, b_j\}$  to  $C$
  - If  $a_i$  is smaller, then move  $i$  to  $i + 1$ ; If  $b_j$  is smaller, then move  $j$  to  $j + 1$ .
  - Break if  $i > n$  or  $j > m$
- Append the remainder of the non-empty list to  $C$

- Analysis 1

- $i$  at most move  $n$  times.
- $j$  at most move  $m$  times for one  $i$  move.
- $O(nm)$ ?



# How fast is the algorithm?

---

- Analysis 1

- $i$  at most move  $n$  times.
- $j$  at most move  $m$  times for one  $i$  movement.
- $O(nm)$ ?

- Analysis 2

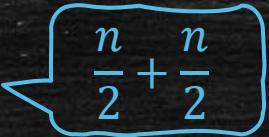
- How many time it takes to append one number to  $C$ ?
- 1 number  $\rightarrow$  1 comparison!
- Output  $m + n$  numbers  $\rightarrow m + n$  comparisons!
- A **linear time** algorithm!
- $O(n + m)$

Charging Argument!



# Finally...

---

- What is the running time of merge sort?
  - Equip the linear time combining into merge sort.
  - Assume merge sort runs  $T(n)$  time to sort a size  $n$  list.
  - What is  $T(n)$ ?
  - $T(n) = 2T\left(\frac{n}{2}\right) + O(n)$  
- I believe you know it is  $O(n \log n)$ ! But, why?



$$T(n) = T\left(\frac{n}{2}\right) + O(n)$$

$$T(n) = T\left(\frac{n}{2}\right) + O(n^2)$$

An efficient tool to  
calculate these expressions

---

$$T(n) = 4T\left(\frac{n}{5}\right) + O(n^3)$$

$$T(n) = 3T\left(\frac{n}{2}\right) + O(n^4)$$



# Master Theorem

---

- Master Theorem

- If  $T(n) = aT\left(\frac{n}{b}\right) + O(n^d)$

- $T(n) = \begin{cases} O(n^d) & a < b^d \\ O(n^{\log_b a}) & a > b^d \\ O(n^d \log n) & a = b^d \end{cases}$



How to understand it?

---



# Understand the parameters

---

- Master Theorem

- If  $T(n) = aT\left(\frac{n}{b}\right) + O(n^d)$

Divide into  $a$   
problems

- $T(n) = \begin{cases} O(n^d) & a < b^d \\ O(n^{\log_b a}) & a > b^d \\ O(n^d \log n) & a = b^d \end{cases}$



# Understand the parameters

- Master Theorem

- If  $T(n) = aT\left(\frac{n}{b}\right) + O(n^d)$

Divide into  $a$  problems

- $T(n) = \begin{cases} O(n^d) & a < b^d \\ O(n^{\log_b a}) & a > b^d \\ O(n^d \log n) & a = b^d \end{cases}$

Subproblem  
size:  $n/b$



# Understand the parameters

- Master Theorem

- If  $T(n) = aT\left(\frac{n}{b}\right) + O(n^d)$

Combining  
cost:  $O(n^d)$

Divide into  $a$   
problems

- $T(n) = \begin{cases} O(n^d) & a < b^d \\ O(n^{\log_b a}) & a > b^d \\ O(n^d \log n) & a = b^d \end{cases}$

Subproblem  
size:  $n/b$



# Running time of merge sort

---

- Recall

- Merge sort divides the problem to **two  $n/2$ -size** problems.
- Merging two  $n/2$ -size sorted lists takes  **$O(n)$**  times.
- $T(n) = 2T\left(\frac{n}{2}\right) + O(n)$

- Use Master Theorem

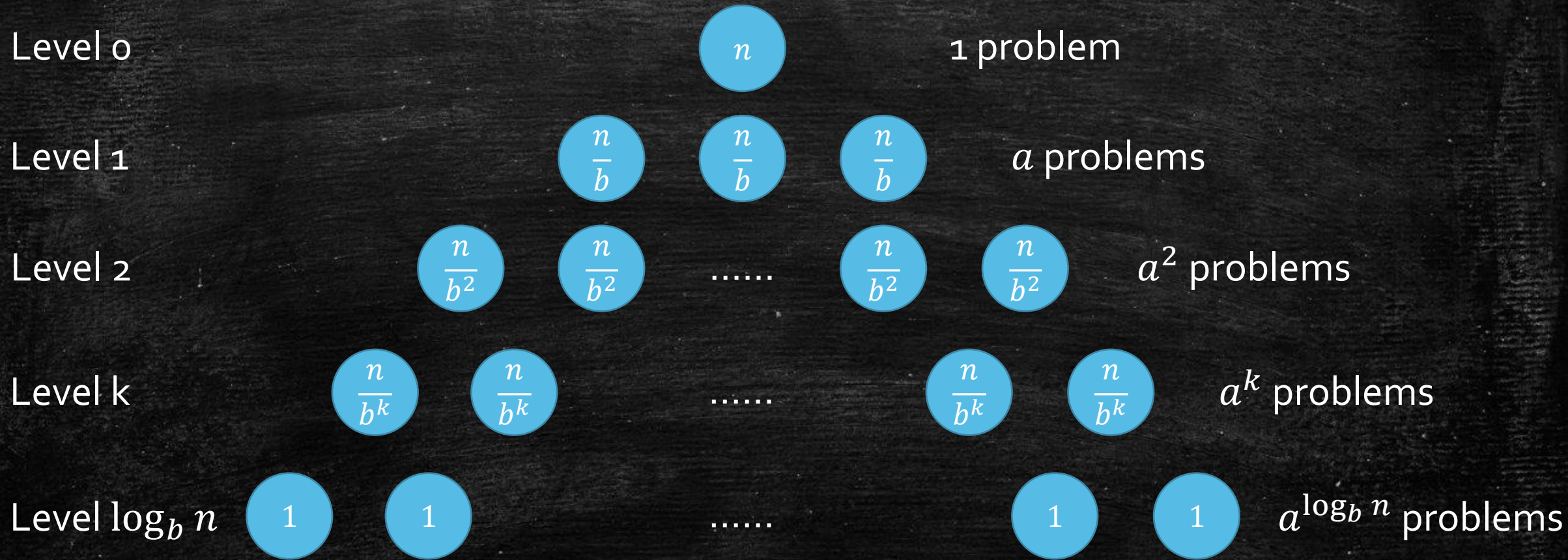
- If  $T(n) = aT\left(\frac{n}{b}\right) + O(n^d)$ , then  $T(n) = \begin{cases} O(n^d) & a < b^d \\ O(n^{\log_b a}) & a > b^d \\ O(n^d \log n) & a = b^d \end{cases}$

- $a = 2, b = 2, d = 1$

- $T(n) = O(n \log n)$



# Understand the formula & proof



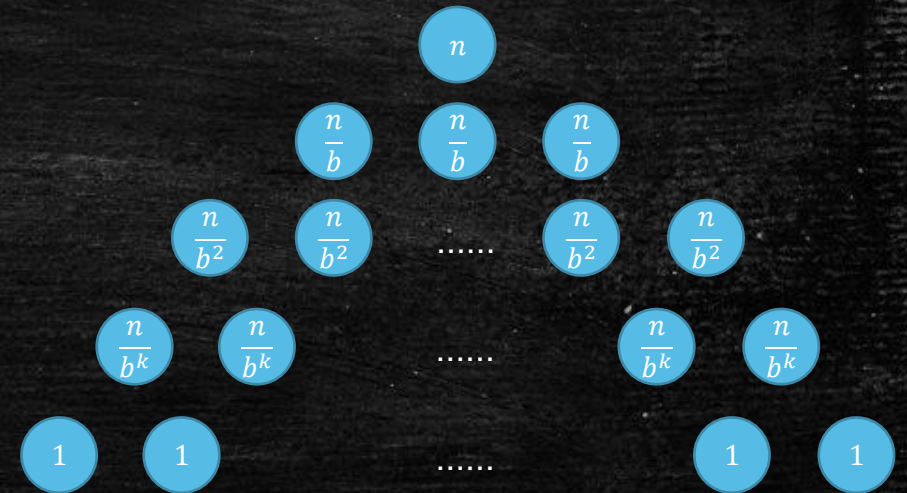


# Understand the formula & proof

- The running time of solving all size-1 problem is
  - $a^{\log_b n} \cdot O(1) = O(n^{\log_b a})$
- The total running time is
  - $O(n^d) + a \cdot O\left(\left(\frac{n}{b}\right)^d\right) + \dots + a^k \cdot O\left(\left(\frac{n}{b^k}\right)^d\right) + \dots + a^{\log_b n} \cdot O(1)$
- Simplification
  - $O(n^d) \cdot \left(1 + \frac{a}{b^d} + \dots + \left(\frac{a}{b^d}\right)^k + \dots + \left(\frac{a}{b^d}\right)^{\log_b n}\right)$

$$= O(n^d) \cdot \left(1 + \frac{a}{b^d} + \dots + \left(\frac{a}{b^d}\right)^k + \dots + \left(\frac{a}{b^d}\right)^{\log_b n}\right)$$

$$n = b^{\log_b n}$$





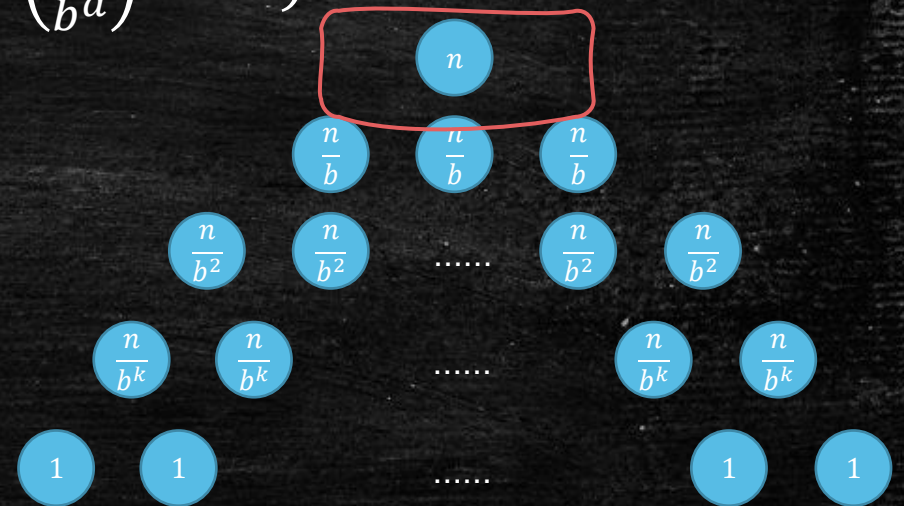
## Case 1: $a < b^d$

$$T(n) = \begin{cases} O(n^d) & a < b^d \\ O(n^{\log_b a}) & a > b^d \\ O(n^d \log n) & a = b^d \end{cases}$$

$$T(n) = O(n^d) \cdot \left(1 + \frac{a}{b^d} + \dots + \left(\frac{a}{b^d}\right)^k + \dots + \left(\frac{a}{b^d}\right)^{\log_b n}\right)$$

$$a < b^d \rightarrow \frac{a}{b^d} < 1$$

$$T(n) = O(n^d)$$





## Case 2: $a > b^d$

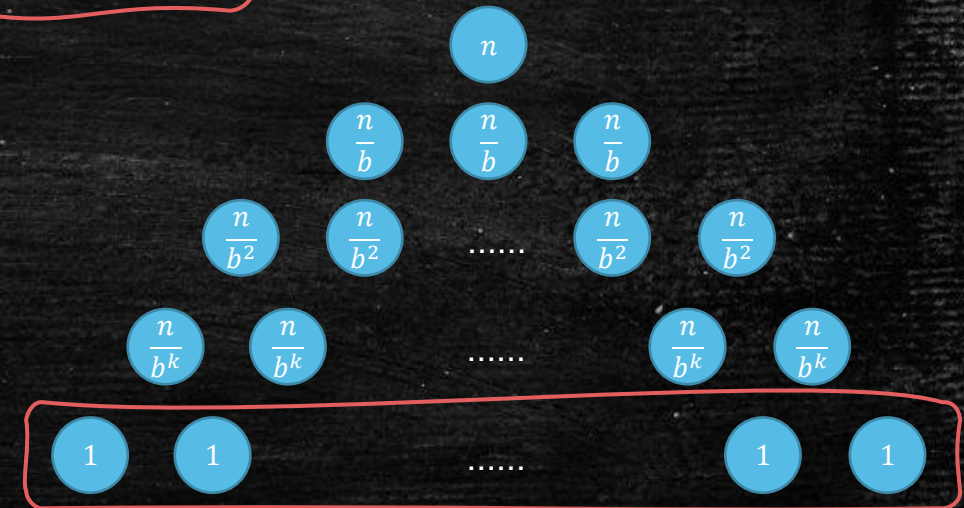
- $T(n) = \begin{cases} O(n^d) & a < b^d \\ O(n^{\log_b a}) & a > b^d \\ O(n^d \log n) & a = b^d \end{cases}$

- $T(n) = O(n^d) \cdot (1 + \frac{a}{b^d} + \dots + (\frac{a}{b^d})^k + \dots + (\frac{a}{b^d})^{\log_b n})$

- $a > b^d \rightarrow \frac{a}{b^d} > 1$

- The last term dominates the sum

- $T(n) = O\left(n^d \left(\frac{a}{b^d}\right)^{\log_b n}\right) = O\left(n^d \frac{a^{\log_b n}}{n^d}\right)$   
 $= O(a^{\log_b n}) = O(n^{\log_b a})$





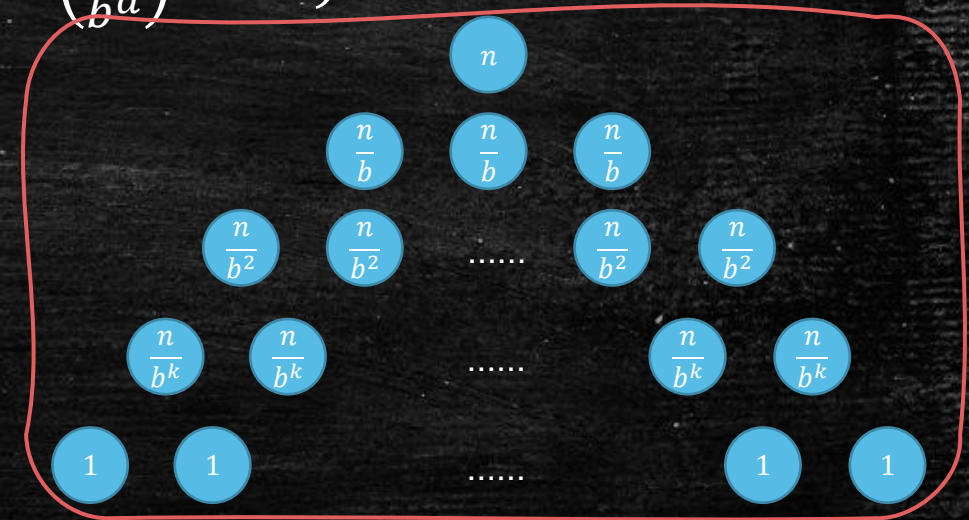
### Case 3: $a = b^d$

$$T(n) = \begin{cases} O(n^d) & a < b^d \\ O(n^{\log_b a}) & a > b^d \\ O(n^d \log n) & a = b^d \end{cases}$$

$$T(n) = O(n^d) \cdot \left(1 + \frac{a}{b^d} + \dots + \left(\frac{a}{b^d}\right)^k + \dots + \left(\frac{a}{b^d}\right)^{\log_b n}\right)$$

$$a = b^d \rightarrow \frac{a}{b^d} = 1$$

$$T(n) = O(n^d \log_b n)$$





# Divide and Conquer

---

Counting Inversions



# Counting Inversions

---

- **Input:** A list of  $n$  integers
  - $x_1, x_2, x_3, \dots, x_n$
- **Output:** number of **inversions**
- Application
  - You rank  $n$  songs.
  - Music site consults database to find people with **similar** tastes.
  - What is **similar**?
    - My rank:  $1, 2, 3, 4, 5, \dots, n$
    - Your rank:  $x_1, x_2, x_3, \dots, x_n$
    - Songs  $i, j$  are **inverted** if  $i < j$  but  $x_i > x_j$ .
    - Similar metric: number of inversions.



# Counting Inversions vs Merge Sort

---



# Counting Inversions

---

- **Input:** A list of  $n$  integers
  - $x_1, x_2, x_3, \dots, x_n$
- **Output:** number of **inversions**
- Plan 1: Brute-force
  - $O(n^2)$



# Counting Inversions

---

- **Input:** A list of  $n$  integers
  - $x_1, x_2, x_3, \dots, x_n$
- **Output:** number of **inversions**
- Plan 2: Divide and Conquer (Merge Sort Style)
  - **Divide:** Dive the input into two subsets:
    - $x_1, x_2, \dots, x_{n/2}; x_{n/2+1}, x_{n/2+2}, \dots, x_n$
  - **Recurse:** count inversions in the two subsets.
    - Let  $c_1, c_2$  be the two numbers.
  - **Combine:** Return the total number of inversions.
    - Count the inversions across subsets, to be  $c_3$ .
    - Return  $c_1 + c_2 + c_3$ .



# Counting Inversions

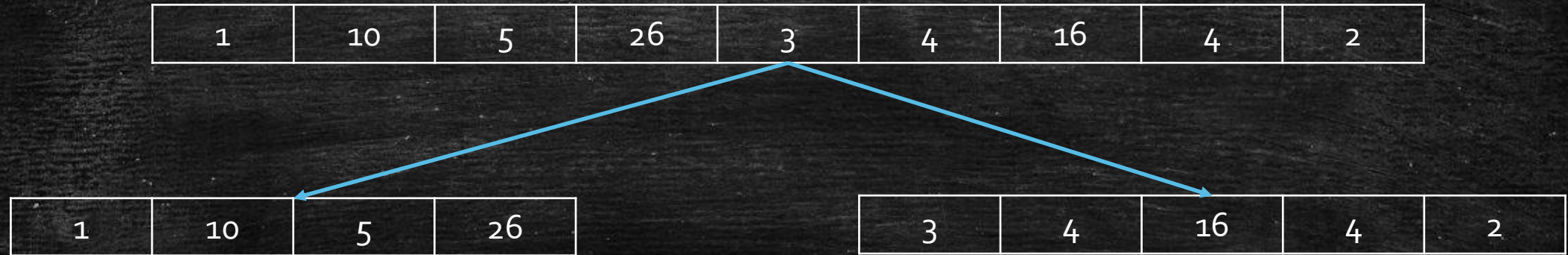
---

1	10	5	26	3	4	16	4	2
---	----	---	----	---	---	----	---	---



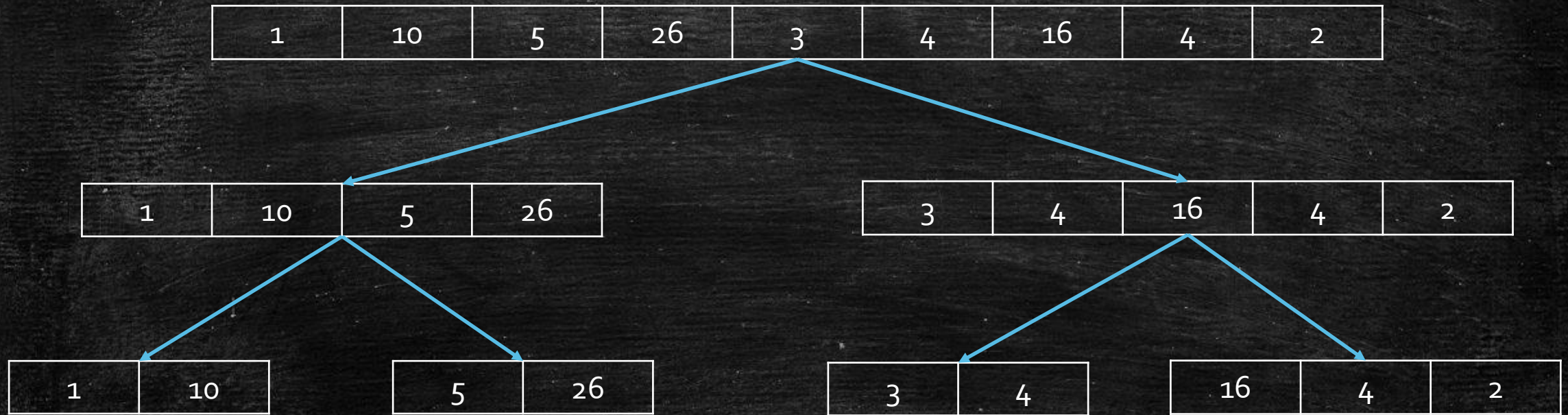
# Counting Inversions

---



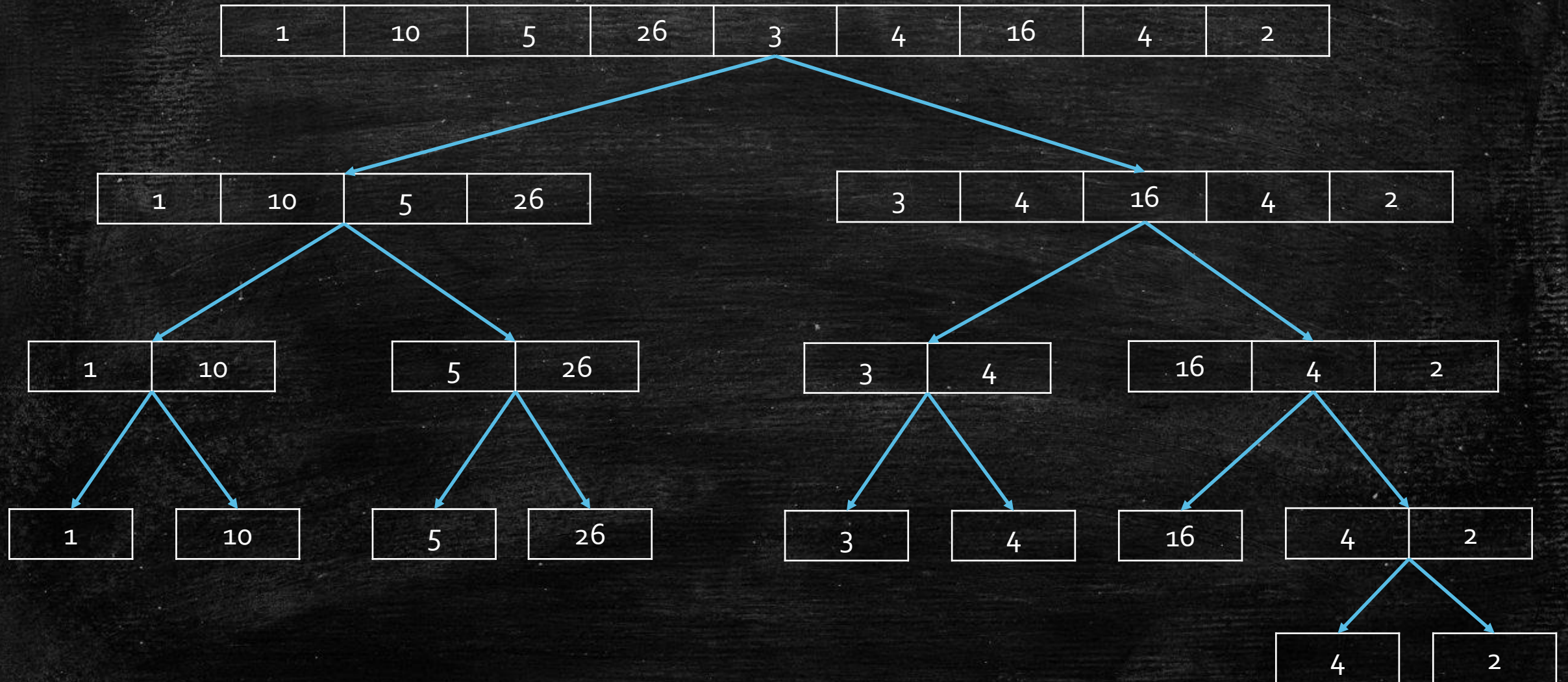


# Counting Inversions



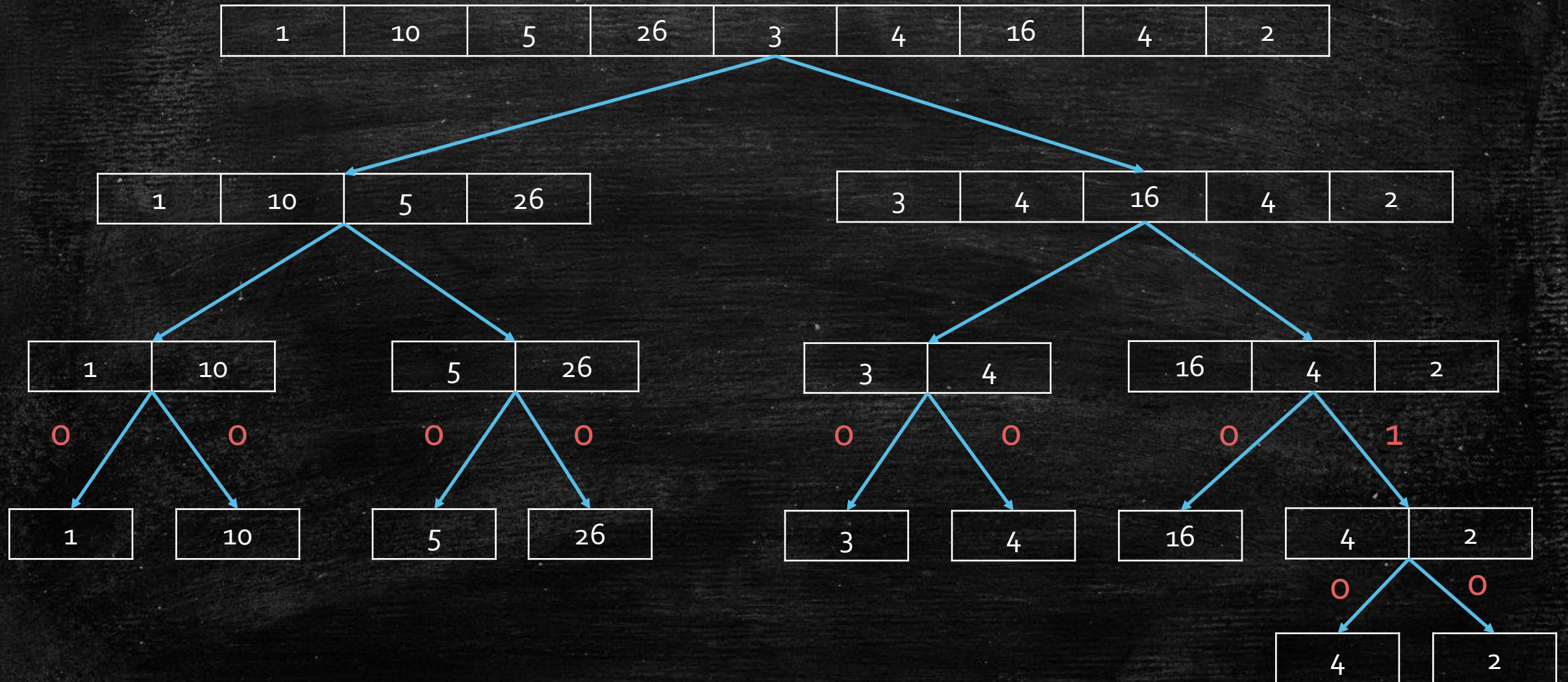


# Counting Inversions



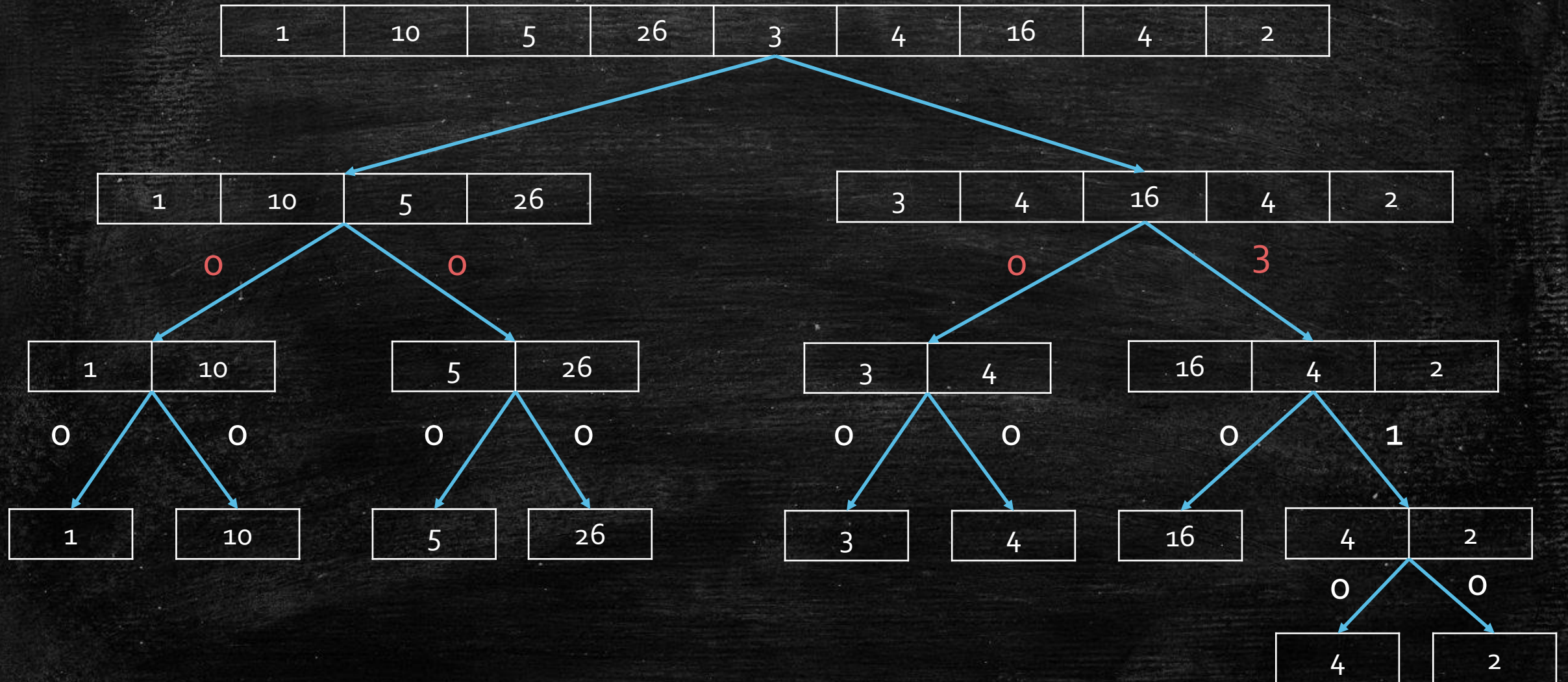


# Counting Inversions





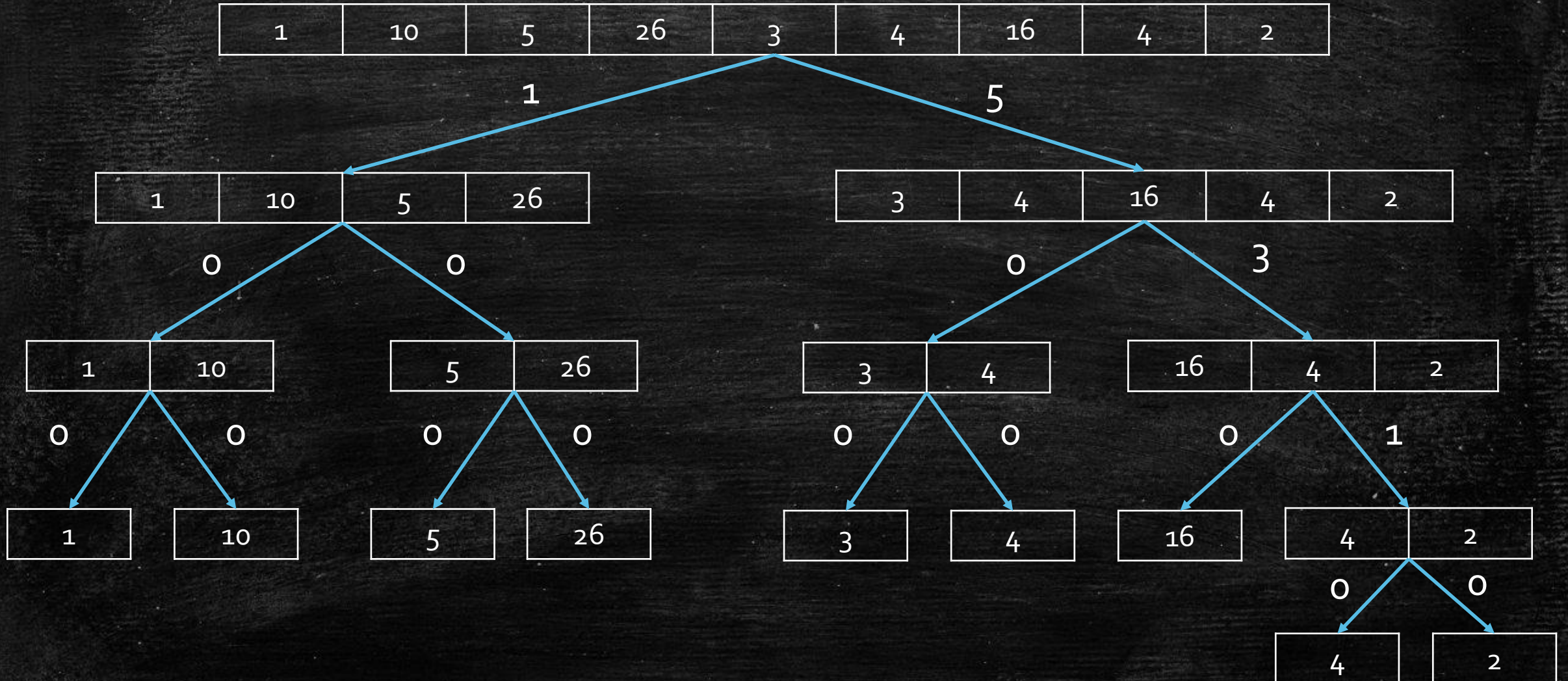
# Counting Inversions





# Counting Inversions

19





# Count inversions across two lists

---

- **Input:** two lists  $A = a_1, a_2, \dots, a_n$ ,  $B = b_1, b_2, \dots, b_m$
- **Output:** number of inversions across two lists
- Plan
  - For each  $a_i$  in  $A$ 
    - Count the number of  $b_j < a_i$  in  $B$ .
    - Add the number into total number of inversions.
  - Return the total number.



# Example

---

1	10	5	26
---	----	---	----



3	4	16	4	2
---	---	----	---	---

Counter = 0

- Plan
  - For each  $a_i$  in  $A$ 
    - Count the number of  $b_j < a_i$  in  $B$ .
    - Add the number into total number of inversions.
  - Return the total number.



# Example

---

1	10	5	26
---	----	---	----



3	4	16	4	2
---	---	----	---	---

Counter = 4

- Plan
  - For each  $a_i$  in  $A$ 
    - Count the number of  $b_j < a_i$  in  $B$ .
    - Add the number into total number of inversions.
  - Return the total number.



# Example

---

1	10	5	26
---	----	---	----



3	4	16	4	2
---	---	----	---	---

Counter = 8

- Plan
  - For each  $a_i$  in  $A$ 
    - Count the number of  $b_j < a_i$  in  $B$ .
    - Add the number into total number of inversions.
  - Return the total number.



# Example

---

1	10	5	26
---	----	---	----



3	4	16	4	2
---	---	----	---	---

Counter = 13

- Plan
  - For each  $a_i$  in  $A$ 
    - Count the number of  $b_j < a_i$  in  $B$ .
    - Add the number into total number of inversions.
  - Return the total number.



# Example

---

1	10	5	26
---	----	---	----

3	4	16	4	2
---	---	----	---	---



Counter = 13

- Plan
  - For each  $a_i$  in  $A$ 
    - Count the number of  $b_j < a_i$  in  $B$ .
    - Add the number into total number of inversions.
  - Return the total number.



# How long it takes?

---

- Analysis
  - Each  $a_i$ , scan the whole list  $B$ .
  - It takes  $O(nm)$ .
- How to improve?
  - It become easier when the two lists are sorted!
  - Why not do **merging** and **counting** together!

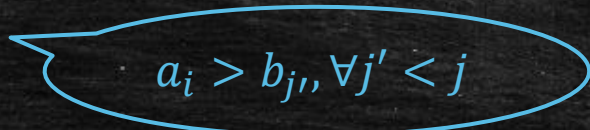


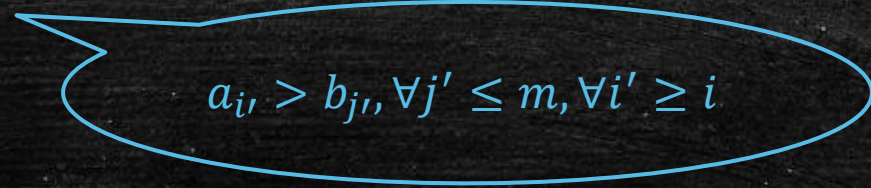
# Merge & Count

---

- Plan

- Maintain 2 pointers  $i = 1, j = 1$ , and a counter  $c = 0$
- Repeat
  - Append  $\min\{a_i, b_j\}$  to  $C$
  - If  $a_i$  is smaller, then move  $i$  to  $i + 1$ ; If  $b_j$  is smaller, then move  $j$  to  $j + 1$ .
  - If we move  $i$  to  $i + 1$ , then  $c = c + j - 1$ .
  - Break if  $i > n$  or  $j > m$
- Append the remainder of the non-empty list to  $C$
- If  $i \leq n$ ,  $c = c + m \cdot (n - i + 1)$


$$a_i > b_{j'}, \forall j' < j$$


$$a_{i'} > b_{j'}, \forall j' \leq m, \forall i' \geq i$$



# Example

---



- Plan
  - Maintain 2 pointers  $i = 1, j = 1$ , and a counter  $c = 0$
  - Repeat
    - Append  $\min\{a_i, b_j\}$  to  $C$
    - If  $a_i$  is smaller, then move  $i$  to  $i + 1$ ; If  $b_j$  is smaller, then move  $j$  to  $j + 1$ .
    - If we move  $i$  to  $i + 1$ , then  $c = c + j - 1$ .
    - Break if  $i > n$  or  $j > m$
  - Append the remainder of the non-empty list to  $C$
  - If  $i \leq n$ ,  $c = c + m \cdot (n - i + 1)$



# Example



- Plan

- Maintain 2 pointers  $i = 1, j = 1$ , and a counter  $c = 0$
- Repeat
  - Append  $\min\{a_i, b_j\}$  to  $C$
  - If  $a_i$  is smaller, then move  $i$  to  $i + 1$ ; If  $b_j$  is smaller, then move  $j$  to  $j + 1$ .
  - If we move  $i$  to  $i + 1$ , then  $c = c + j - 1$ .
  - Break if  $i > n$  or  $j > m$
- Append the remainder of the non-empty list to  $C$
- If  $i \leq n$ ,  $c = c + m \cdot (n - i + 1)$



# Example

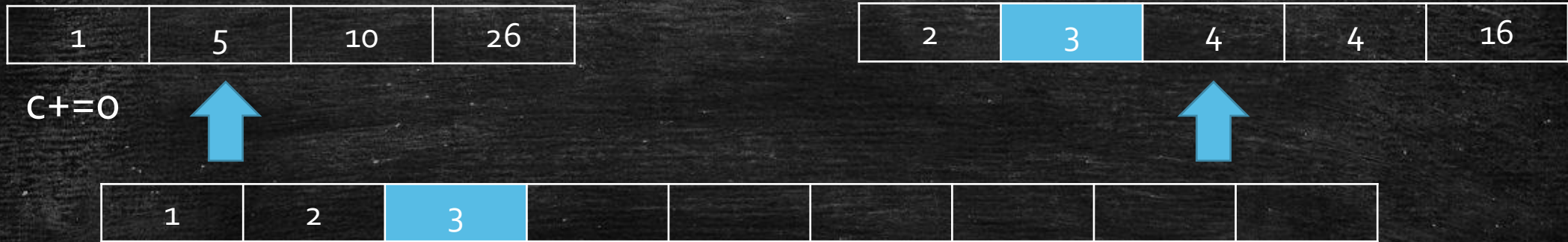


- Plan

- Maintain 2 pointers  $i = 1, j = 1$ , and a counter  $c = 0$
- Repeat
  - Append  $\min\{a_i, b_j\}$  to  $C$
  - If  $a_i$  is smaller, then move  $i$  to  $i + 1$ ; If  $b_j$  is smaller, then move  $j$  to  $j + 1$ .
  - If we move  $i$  to  $i + 1$ , then  $c = c + j - 1$ .
  - Break if  $i > n$  or  $j > m$
- Append the remainder of the non-empty list to  $C$
- If  $i \leq n$ ,  $c = c + m \cdot (n - i + 1)$



# Example

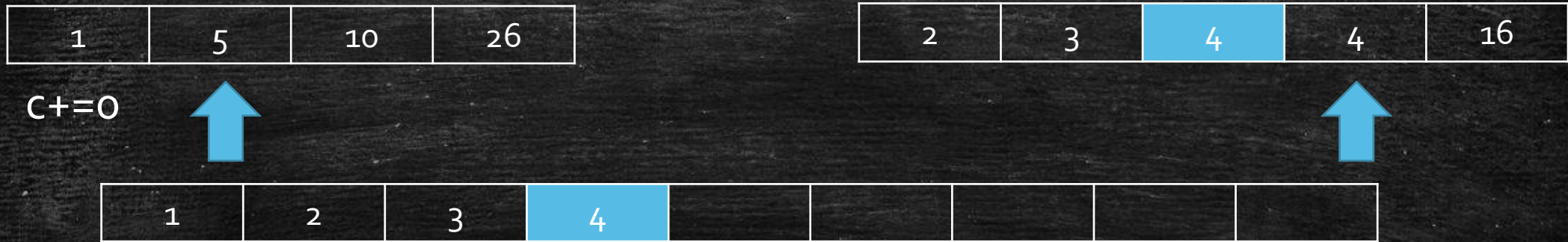


- Plan

- Maintain 2 pointers  $i = 1, j = 1$ , and a counter  $c = 0$
- Repeat
  - Append  $\min\{a_i, b_j\}$  to  $C$
  - If  $a_i$  is smaller, then move  $i$  to  $i + 1$ ; If  $b_j$  is smaller, then move  $j$  to  $j + 1$ .
  - If we move  $i$  to  $i + 1$ , then  $c = c + j - 1$ .
  - Break if  $i > n$  or  $j > m$
- Append the remainder of the non-empty list to  $C$
- If  $i \leq n$ ,  $c = c + m \cdot (n - i + 1)$



# Example

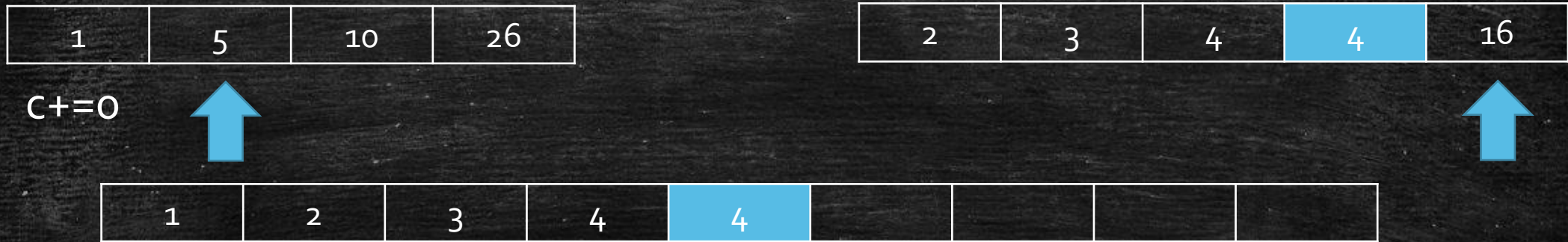


- Plan

- Maintain 2 pointers  $i = 1, j = 1$ , and a counter  $c = 0$
- Repeat
  - Append  $\min\{a_i, b_j\}$  to  $C$
  - If  $a_i$  is smaller, then move  $i$  to  $i + 1$ ; If  $b_j$  is smaller, then move  $j$  to  $j + 1$ .
  - If we move  $i$  to  $i + 1$ , then  $c = c + j - 1$ .
  - Break if  $i > n$  or  $j > m$
- Append the remainder of the non-empty list to  $C$
- If  $i \leq n$ ,  $c = c + m \cdot (n - i + 1)$



# Example

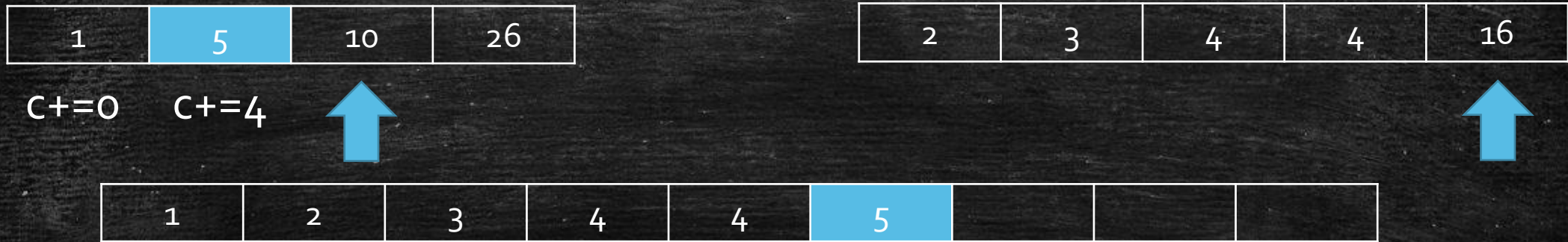


- Plan

- Maintain 2 pointers  $i = 1, j = 1$ , and a counter  $c = 0$
- Repeat
  - Append  $\min\{a_i, b_j\}$  to  $C$
  - If  $a_i$  is smaller, then move  $i$  to  $i + 1$ ; If  $b_j$  is smaller, then move  $j$  to  $j + 1$ .
  - If we move  $i$  to  $i + 1$ , then  $c = c + j - 1$ .
  - Break if  $i > n$  or  $j > m$
- Append the remainder of the non-empty list to  $C$
- If  $i \leq n$ ,  $c = c + m \cdot (n - i + 1)$



# Example



- Plan

- Maintain 2 pointers  $i = 1, j = 1$ , and a counter  $c = 0$
- Repeat
  - Append  $\min\{a_i, b_j\}$  to  $C$
  - If  $a_i$  is smaller, then move  $i$  to  $i + 1$ ; If  $b_j$  is smaller, then move  $j$  to  $j + 1$ .
  - If we move  $i$  to  $i + 1$ , then  $c = c + j - 1$ .
  - Break if  $i > n$  or  $j > m$
- Append the remainder of the non-empty list to  $C$
- If  $i \leq n$ ,  $c = c + m \cdot (n - i + 1)$



# Example

1	5	10	26
---	---	----	----

$c += 0$

$c += 4$

$c += 4$



2	3	4	4	16
---	---	---	---	----



1	2	3	4	4	5	10		
---	---	---	---	---	---	----	--	--

- Plan

- Maintain 2 pointers  $i = 1, j = 1$ , and a counter  $c = 0$
- Repeat
  - Append  $\min\{a_i, b_j\}$  to  $C$
  - If  $a_i$  is smaller, then move  $i$  to  $i + 1$ ; If  $b_j$  is smaller, then move  $j$  to  $j + 1$ .
  - If we move  $i$  to  $i + 1$ , then  $c = c + j - 1$ .
  - Break if  $i > n$  or  $j > m$
- Append the remainder of the non-empty list to  $C$
- If  $i \leq n$ ,  $c = c + m \cdot (n - i + 1)$



# Example

1	5	10	26
---	---	----	----

$c += 0$

$c += 4$

$c += 4$



2	3	4	4	16
---	---	---	---	----



1	2	3	4	4	5	10	16	
---	---	---	---	---	---	----	----	--

- Plan

- Maintain 2 pointers  $i = 1, j = 1$ , and a counter  $c = 0$
- Repeat
  - Append  $\min\{a_i, b_j\}$  to  $C$
  - If  $a_i$  is smaller, then move  $i$  to  $i + 1$ ; If  $b_j$  is smaller, then move  $j$  to  $j + 1$ .
  - If we move  $i$  to  $i + 1$ , then  $c = c + j - 1$ .
  - Break if  $i > n$  or  $j > m$
- Append the remainder of the non-empty list to  $C$
- If  $i \leq n$ ,  $c = c + m \cdot (n - i + 1)$



# Example

1	5	10	26
---	---	----	----

$c += 0$     $c += 4$     $c += 4$     $c += 5$



2	3	4	4	16
---	---	---	---	----



1	2	3	4	4	5	10	16	26
---	---	---	---	---	---	----	----	----

- Plan

- Maintain 2 pointers  $i = 1, j = 1$ , and a counter  $c = 0$
- Repeat
  - Append  $\min\{a_i, b_j\}$  to  $C$
  - If  $a_i$  is smaller, then move  $i$  to  $i + 1$ ; If  $b_j$  is smaller, then move  $j$  to  $j + 1$ .
  - If we move  $i$  to  $i + 1$ , then  $c = c + j - 1$ .
  - Break if  $i > n$  or  $j > m$
- Append the remainder of the non-empty list to  $C$
- If  $i \leq n$ ,  $c = c + m \cdot (n - i + 1)$



Try to prove to yourself the  
correctness.

---



# How fast is it now?

---

- The same as Merging two sorted listed.
- Counting Inversion is as fast as Merge Sort.
- $T(n) = 2T\left(\frac{n}{2}\right) + O(n) = O(n \log n)$



# Today's goal

---

- Learn Insertion Sort and Merge Sort
- Learn how to Count Inversions with Merge Sort
- Learn to prove the correctness of them
- Learn to analyze their running time
  - **Charging Argument**
- Learn the Master Theorem