CSE 331:

Algorithms & Complexity

"Counting Inversions"

Prof. Charlie Anne Carlson (She/Her)

Lecture 24

Wednesday October 29th, 2025

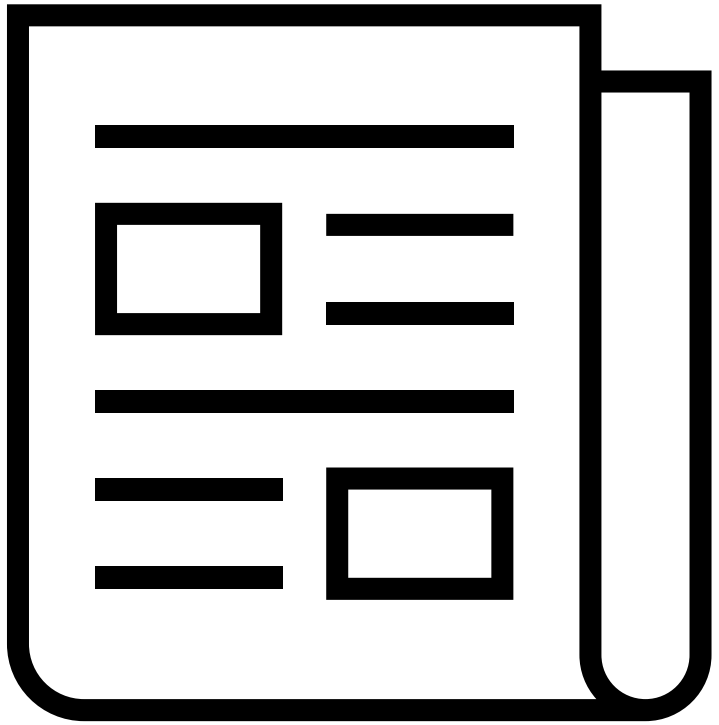
University at Buffalo®

Schedule

1. Course Updates
2. Counting Inversions

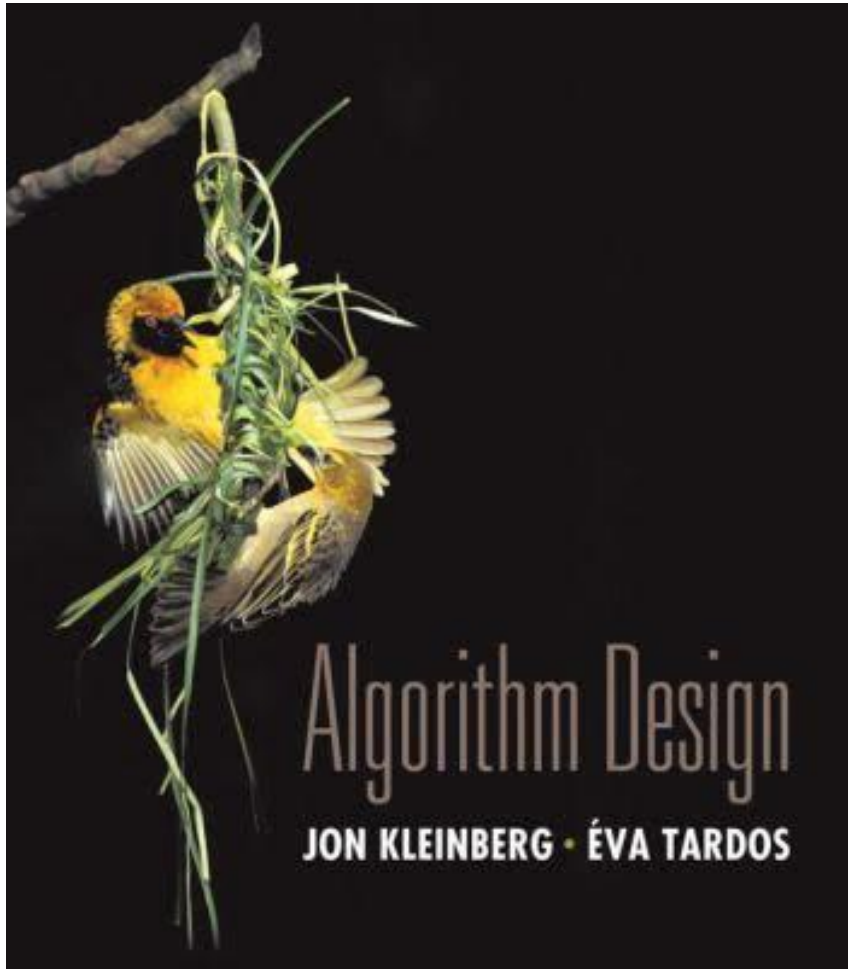
Course Updates

- Post Midterm Grades <-piazza
- HW4 Grades Out
- HW4 Solutions Out
- HW 6 Out Soon
- Group Project
  - Code Problems Oct 31st
  - Reflections November 3rd



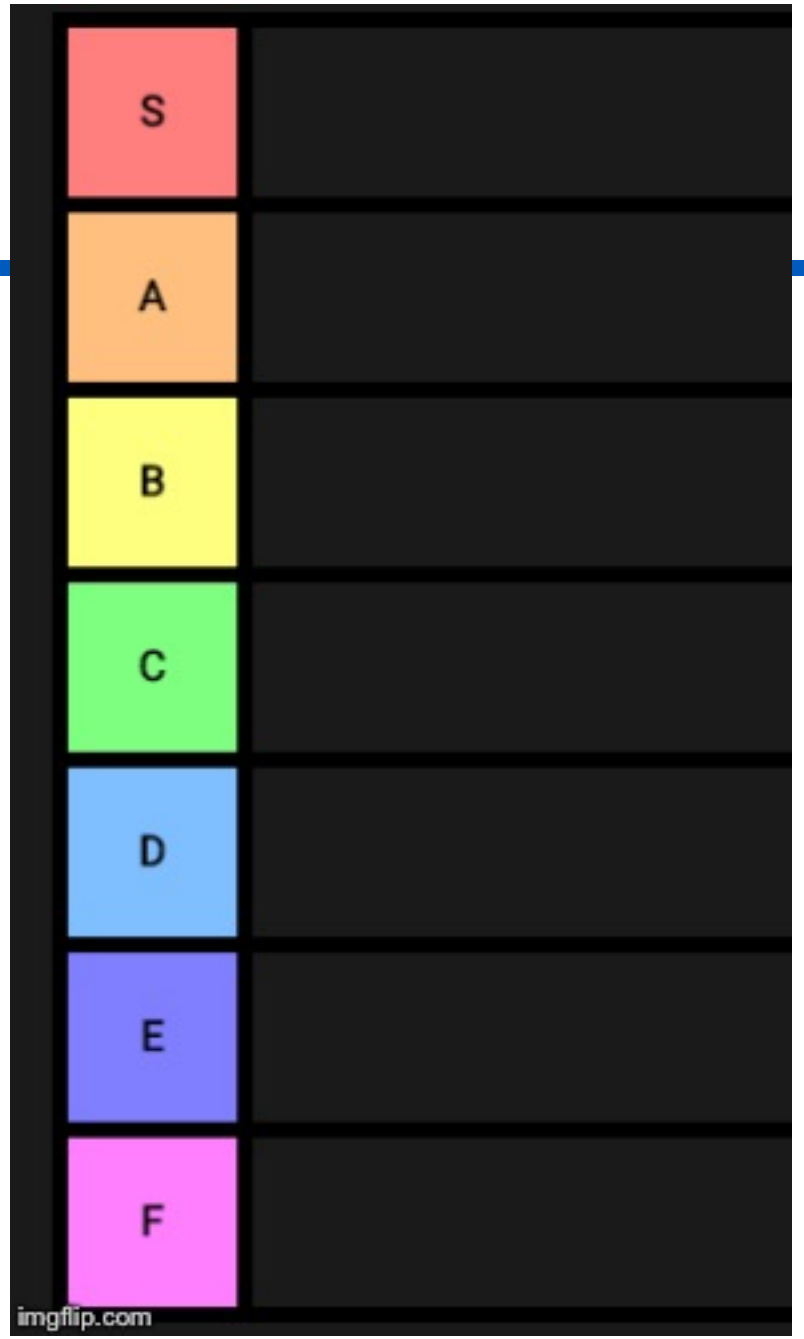
Reading

- You should have read:
  - Finished KT 5.1
  - Finished KT 5.2
  - Finished KT 5.3
- Before Next Class:
  - Start KT 5.5
  - Start KT 5.4



Rankings

- Fix a set of objects and consider ordering them by some known preference.
- E.g. Given the following "animals," ***rank*** your preference for having them as pets:
  - Cat
  - Dog
  - Snake
  - Rock



Rankings

- Twp people may have very different preferences for pets.
- **Question**: How can you measure the difference between two people's preferences/ranks?

Charlie	Not-Charlie
Cat	Dog
Dog	Rock
Snake	Cat
Rock	Snake

Rankings

- Twp people may have very different preferences for pets.
- **Answer**: You might create a ***measure*** that is 0 when the preferences match and increases the more different they are.

Charlie	Not-Charlie
Cat	Dog
Dog	Rock
Snake	Cat
Rock	Snake

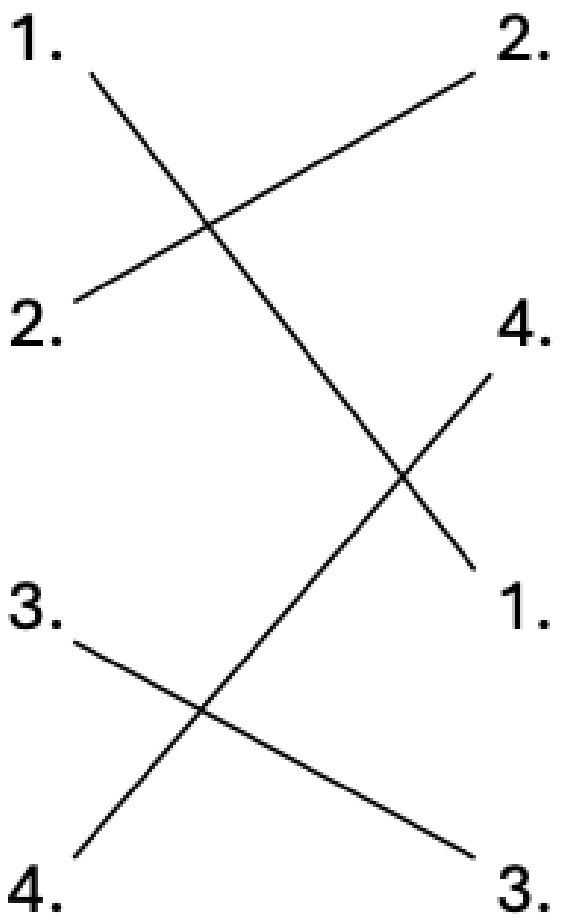
Rankings

- Twp people may have very different preferences for pets.
- **Answer**: You might create a ***measure*** that is 0 when the preferences match and increases the more different they are.
- Let's fix Charlie's preference as the "ordering".

Charlie	Not-Charlie
1. Cat	2. Dog
2. Dog	4. Rock
3. Snake	1. Cat
4. Rock	3. Snake

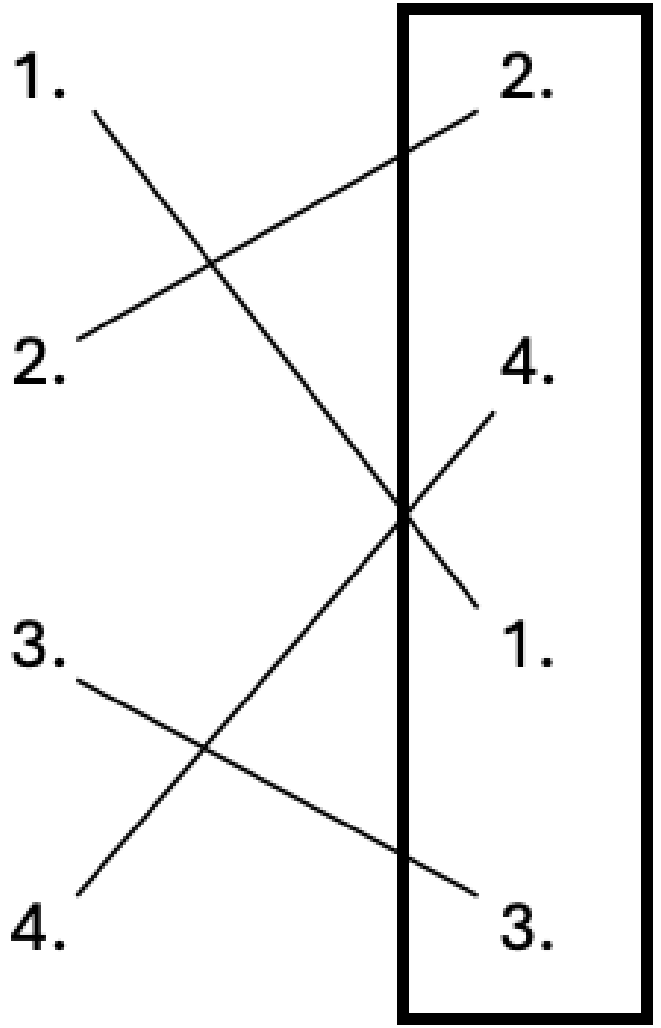
Inversions

- A natural measure for measuring difference is counting ***inversions*.**
- **Definition:** Given a list of numbers $a_1, \ldots, a_n$, we say two indices $\mathrm{i} < \mathrm{j}$ form an inversion if $a_i > a_j$.



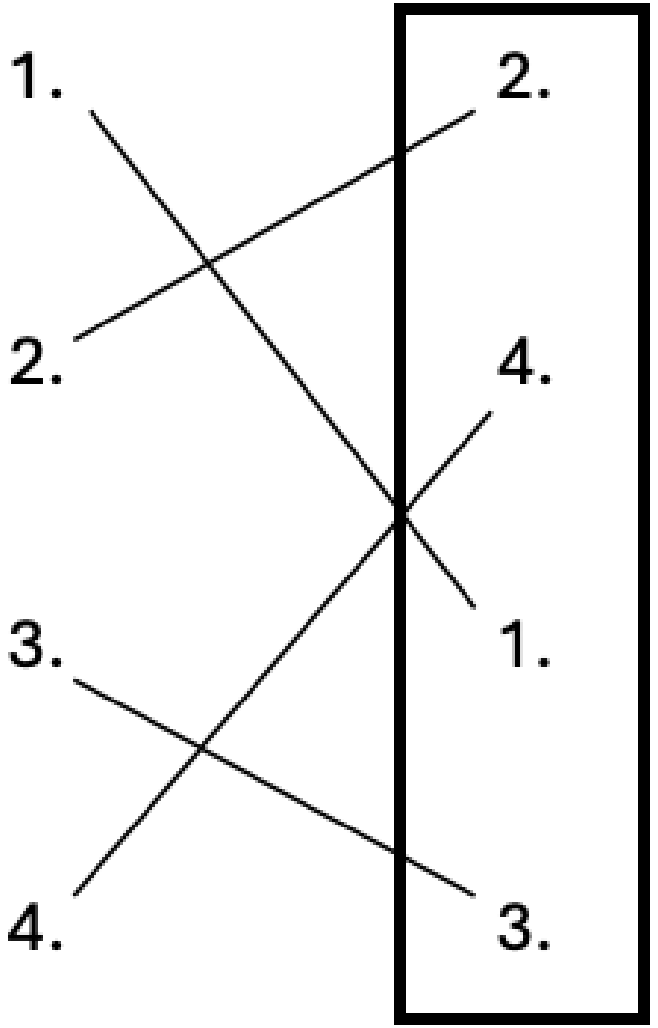
Inversions

- **Definition:** Given a list of numbers $a_1, \ldots, a_n$, we say two indices $\mathrm{i} < \mathrm{j}$ form an inversion if $a_i > a_j$.
- Let's ignore Charlie for a second and just look at the numbers on the right:
  - [2, 4, 1, 3]



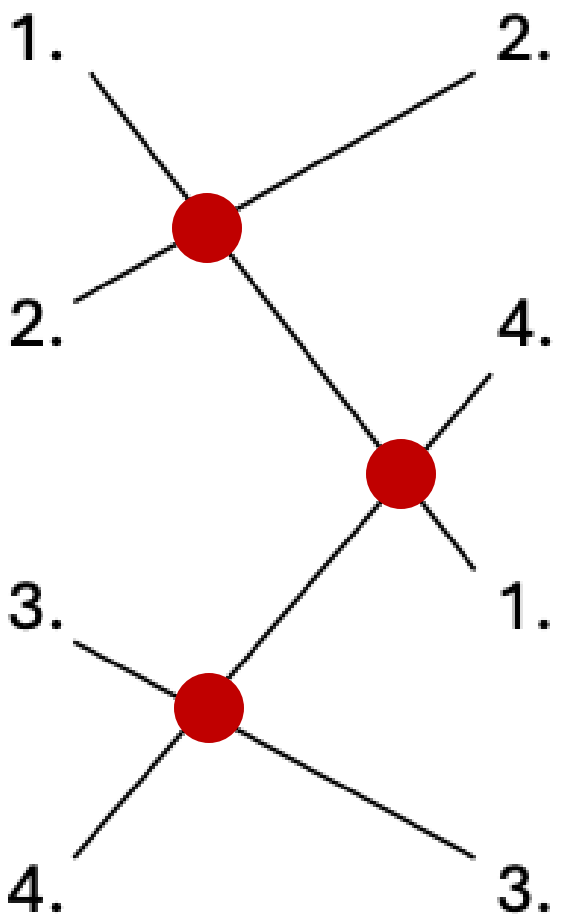
Inversions

- **Definition:** Given a list of numbers $a_1, \ldots, a_n$, we say two indices i $<$ j form an inversion if $a_i > a_j$.
- Let's ignore Charlie for a second and just look at the numbers on the right:
  - [2, 4, 1, 3]
  - (2,1), (4,1), and (4,3) are all inversions.



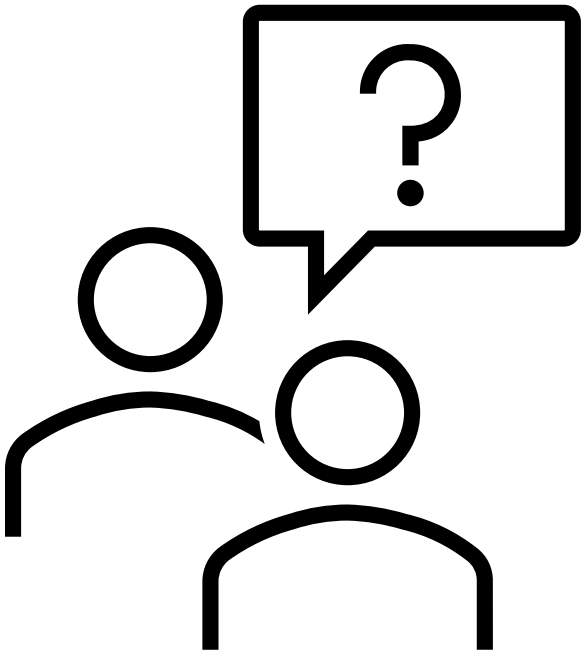
Inversions

- Let’s ignore Charlie for a second and just look at the numbers on the right:
  - [2, 4, 1, 3]
  - (2,1), (4,1), and (4,3) are all inversions.
- Observe that if you look at the lines, an inversion is marked by a crossing.



Problem: Finding Inversions

- **Input**: List of elements A
- **Goal**: Find all the inversions in A



Problem: Finding Inversions

- **Input**: List of elements A
- **Goal**: Find all the inversions in A

```
Input: List A
Inversions = []
For i in [n]:
  For j in [i+1..n]:
    if A[i] > A[j]:
      inversions.append((i,j))
Return Inversions
```

Problem: Finding Inversions

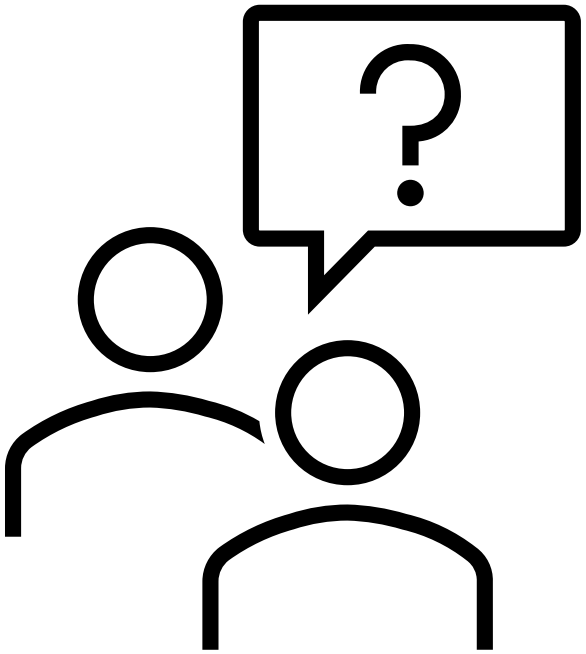
- **Input**: List of elements A
- **Goal**: Find all the inversions in A

```
Input: List A
Inversions = []
For i in [n]:
  For j in [i+1..n]:
    if A[i] > A[j]:
      inversions.append((i,j))
Return Inversions
```

This will take n^2 time to compute the list of inversions!

Problem: Finding Inversions

- **Input**: List of elements A
- **Goal**: Find number of inversions in A



Recursive Algorithm

- The same algorithm works, just return the size of the final inversions list.
- However, we can use recursion to do better.
- Let's make a few observations about the problem before we decide how to use recursion.

Simple Case

- How many inversions can we have if our list is empty?
- How many inversions can we have if our list is size 1?
- How many inversions can we have if our list is size 2?
  - How can you solve this small case?

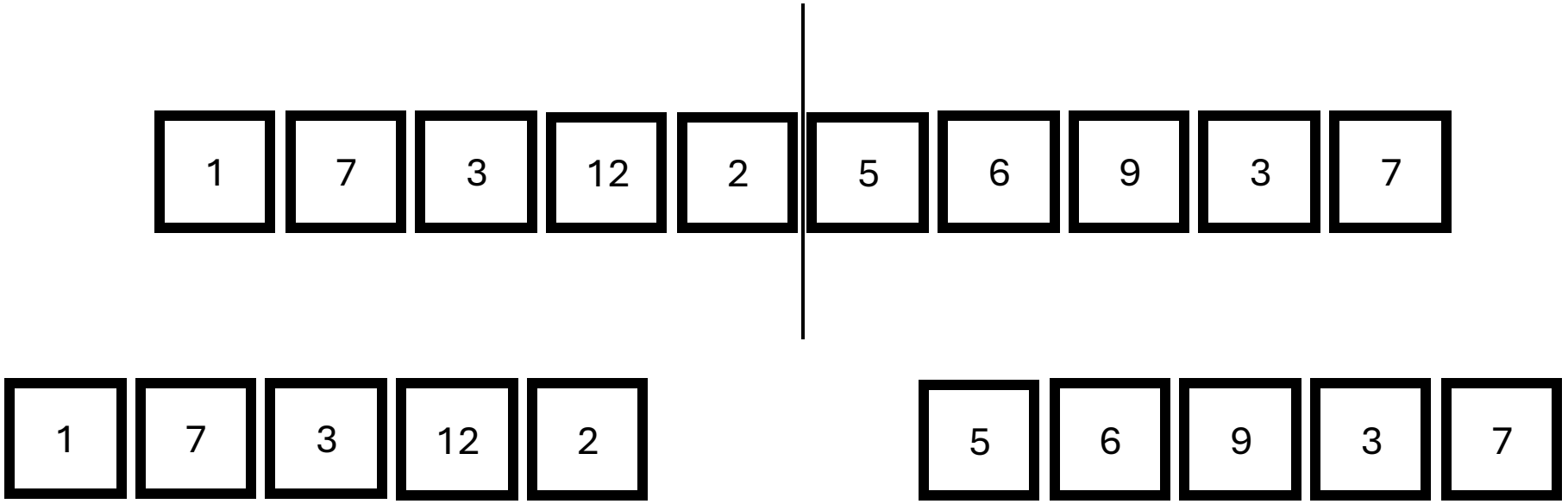
[]

[2]

[3,1]

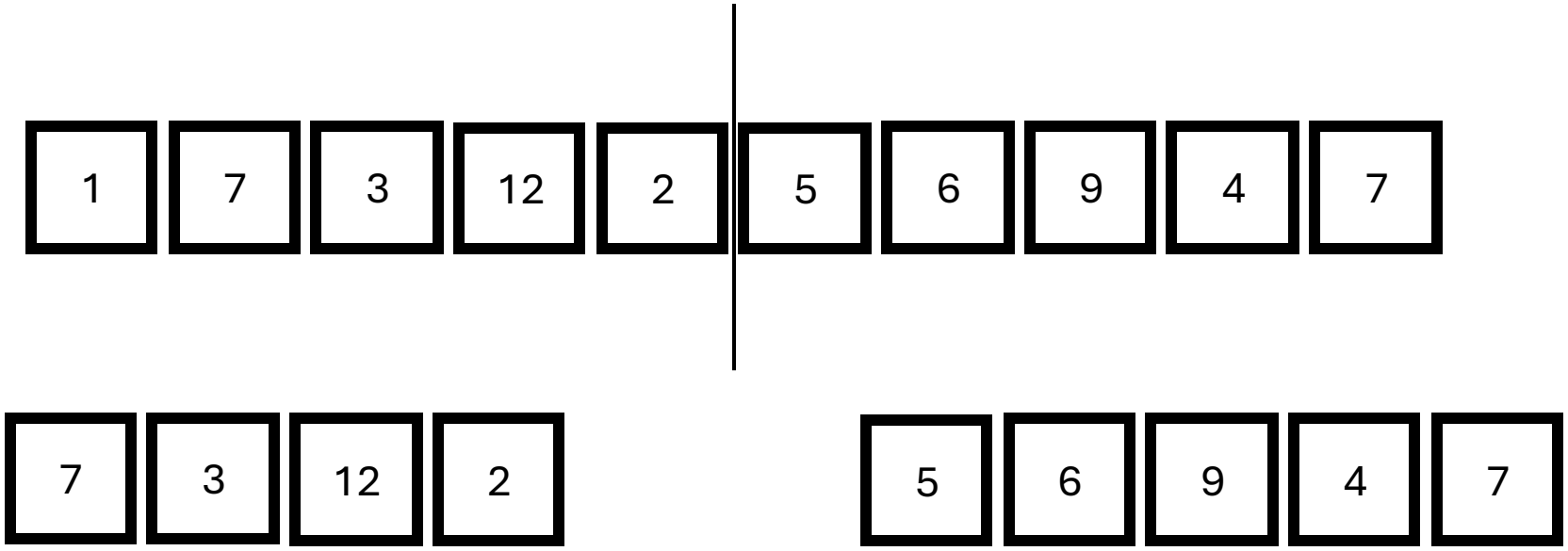
Divide & Conquer

- If we aren't in the simple/base case, then we want to break the problem into smaller problems and recurse.



Merging

- If we aren't in the simple/base case, then we want to break the problem into smaller problems and recurse.



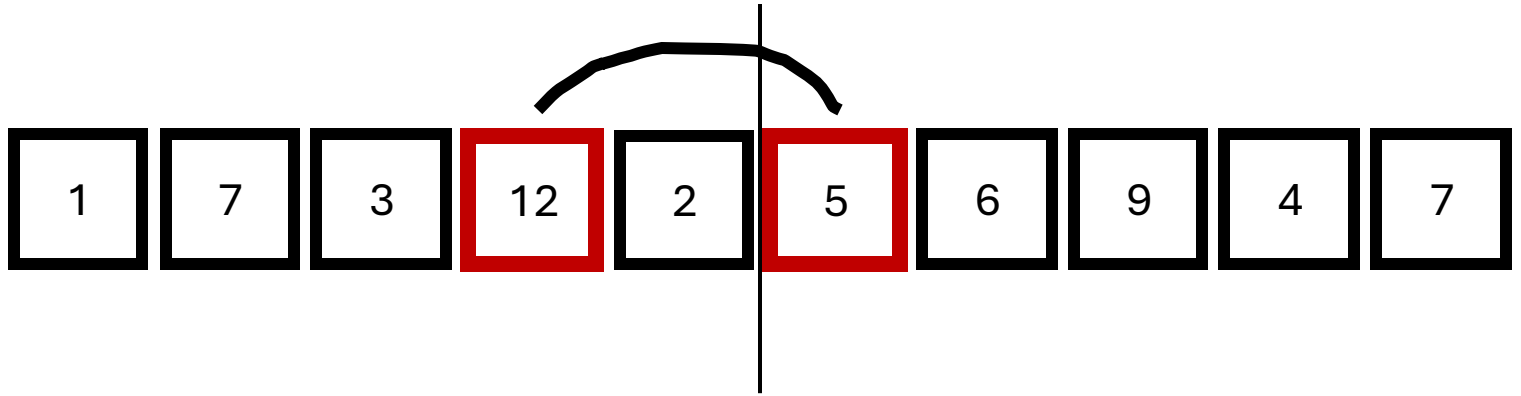
There are 3 inversions: (7,3), (3,2), and (12, 2)

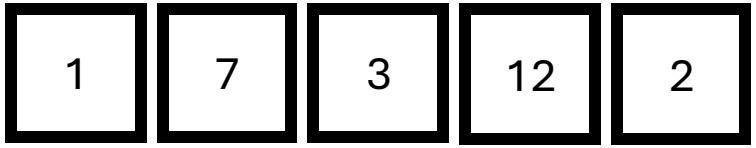
There is 4 inversions: (5,4), (6,4), (9,4), and (9,7)

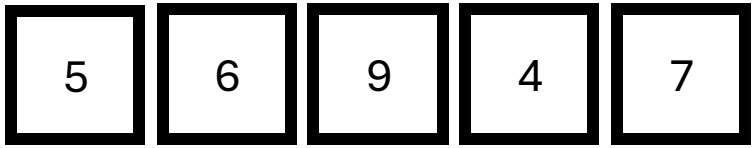
Question: Are these all the inversions?

Merging

- If we aren't in the simple/base case, then we want to break the problem into smaller problems and recurse.







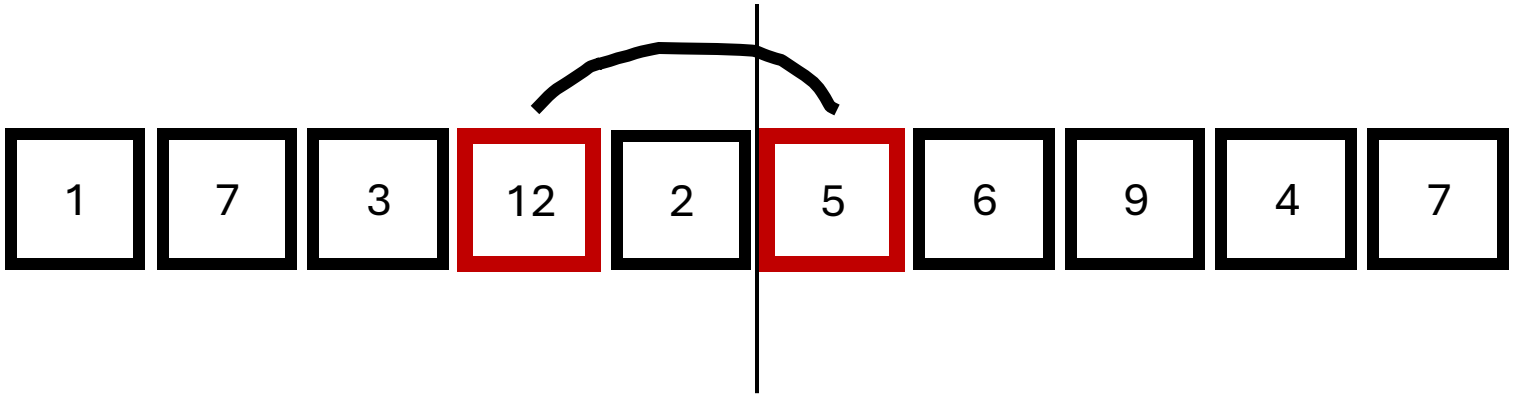
There are 3 inversions: (7,3), (3,2), and (12, 2)

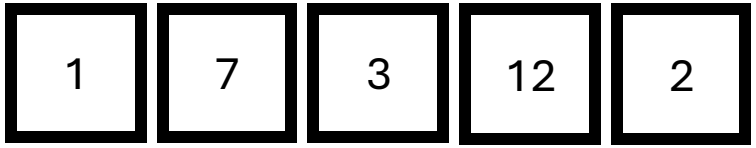
There is 4 inversions: (5,4), (6,4), (9,4), and (9,7)

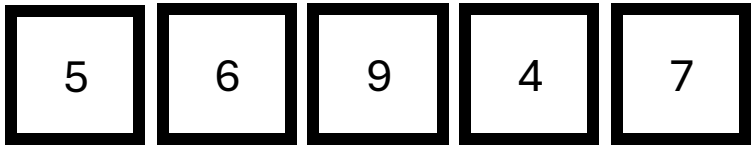
Answer: No!

More Merging

- Once we have the solution to our subproblems, we have to find a way to combine them together.





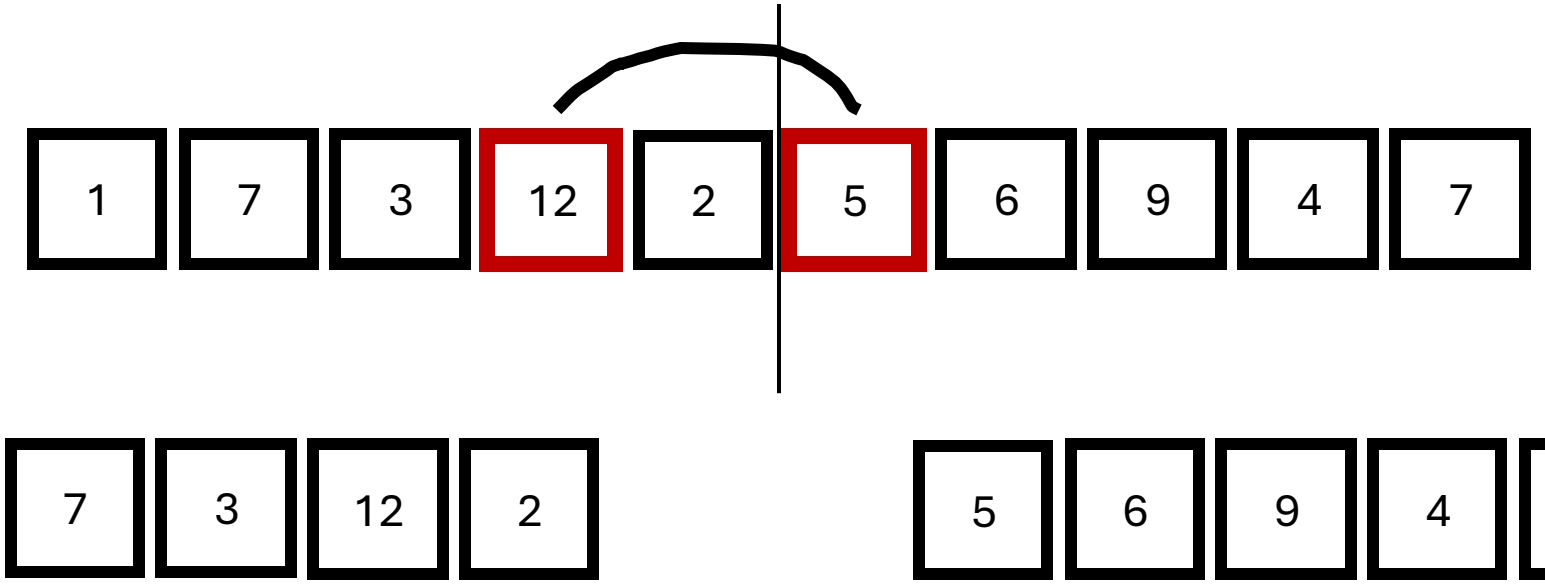


There are 3 inversions: (7,3), (3,2), and (12, 2)

There is 4 inversions: (5,4), (6,4), (9,4), and (9,7)

Naive Idea

- **Question**: How long could it take to check all of this "spanning" inversions?

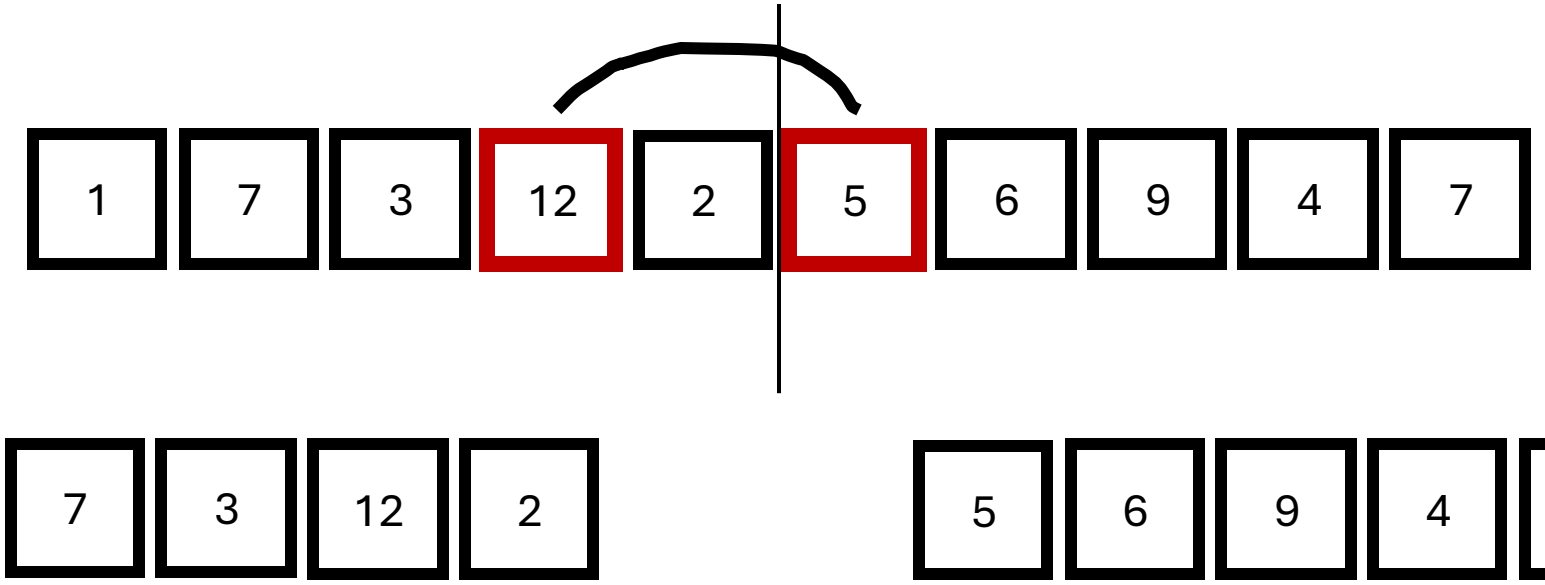


There are 3 inversions: (7,3), (3,2), and (12, 2)

There is 4 inversions: (5,4), (6,4), (9,4), and (9,7)

Naive Idea

- **Answer**: If we assume that n is even then it would take (n/2)^2 comparisons.

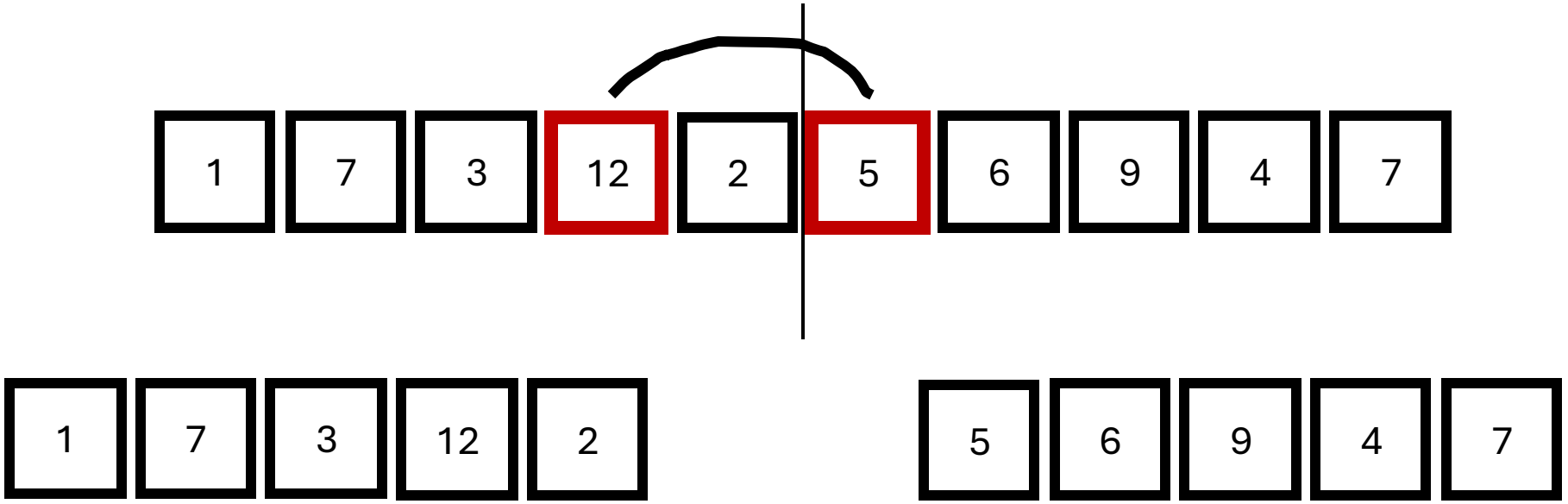


There are 3 inversions: (7,3), (3,2), and (12, 2)

There is 4 inversions: (5,4), (6,4), (9,4), and (9,7)

Easy Case

- **Question**: What if we knew that both lists were sorted?

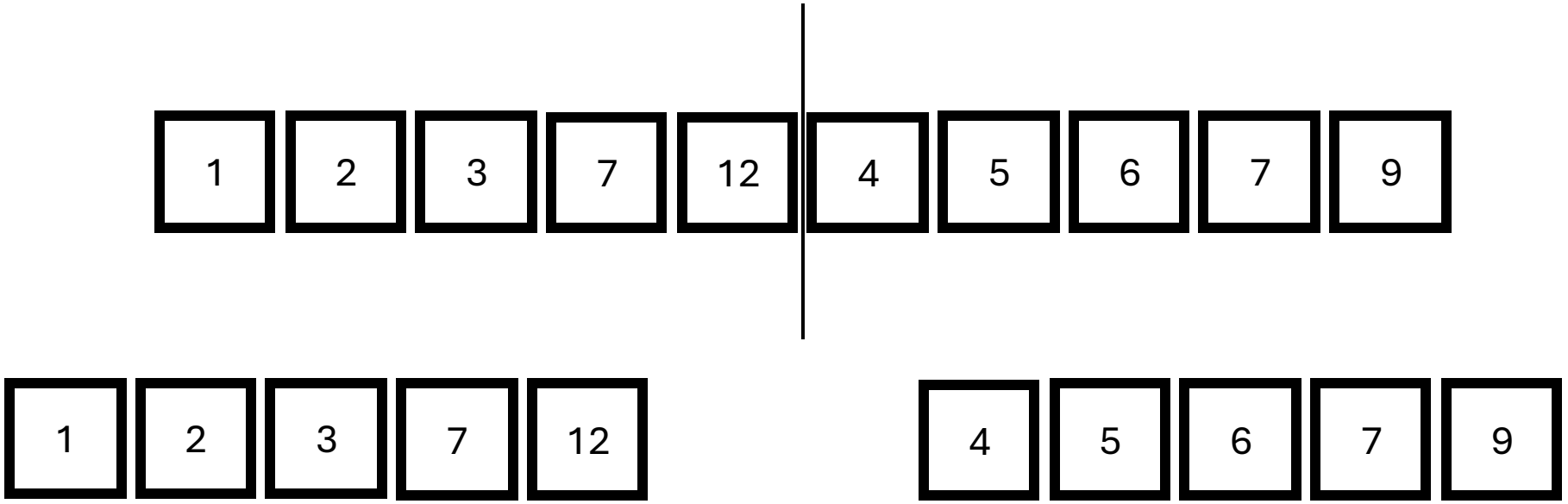


There are 3 inversions: (7,3), (3,2), and (12, 2)

There is 4 inversions: (5,4), (6,4), (9,4), and (9,7)

Easy Case

- **Question**: What if we knew that both sublists were sorted?

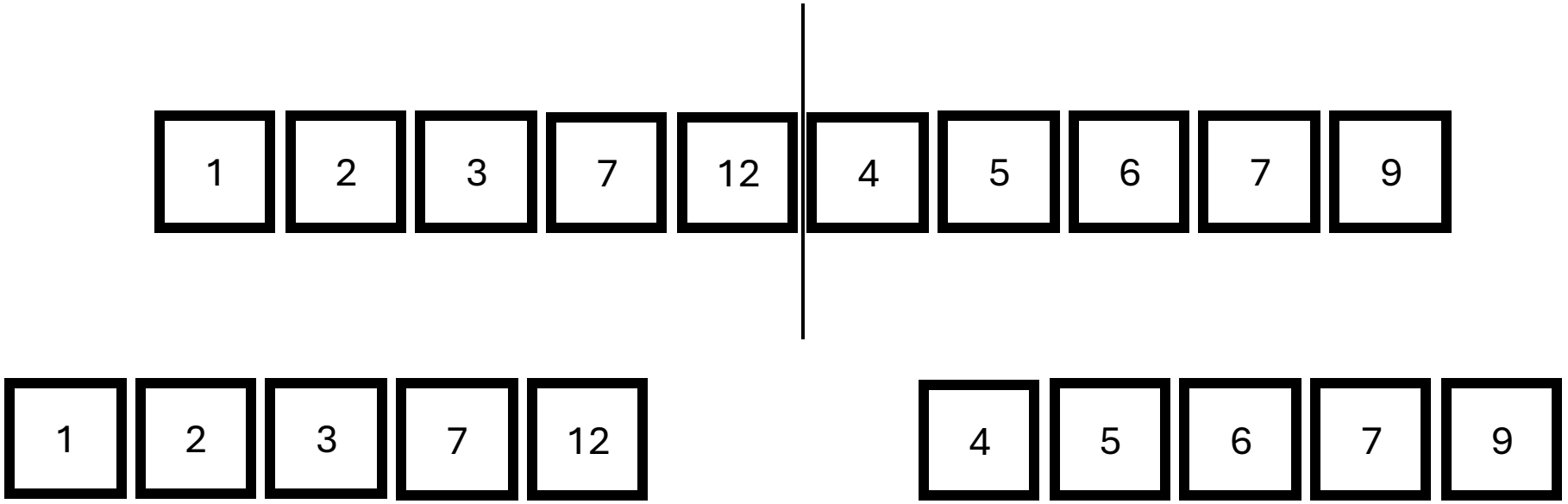


There are 0 inversions.

There is 0 inversions.

Easy Case

- **Answer**: No inversions in each sublist but we can count the spanning inversions easier.

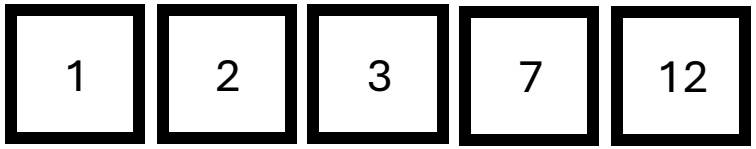


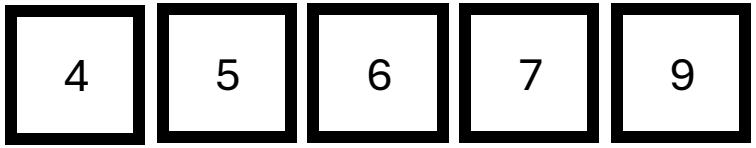
There are 0 inversions.

There is 0 inversions.

Merge & Count

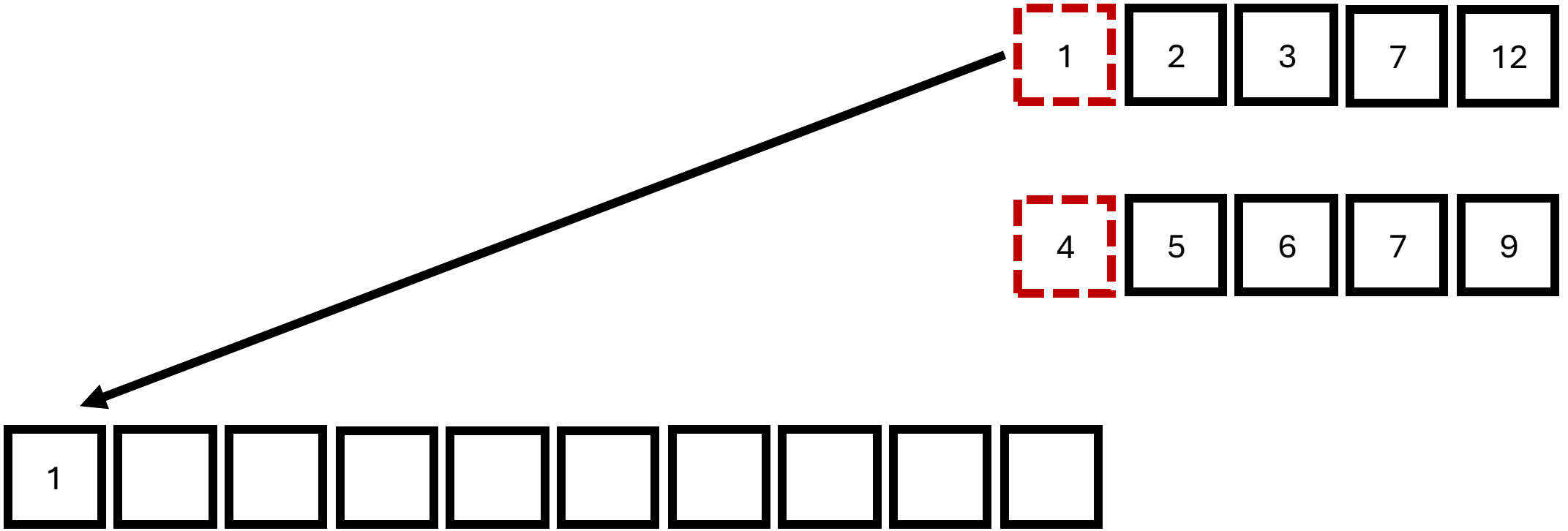
- Let’s do something like our merge step from mergesort but also count the number of inversions.
- Work through both lists and add the smallest element to the merged list.
  - Now, we also compute the number of inversions when we add the smallest element





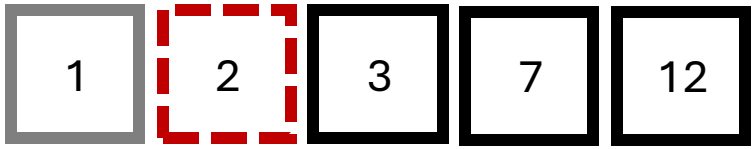
Merge & Count

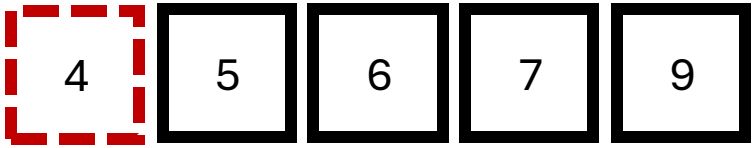
Question: How many inversions involving 1?

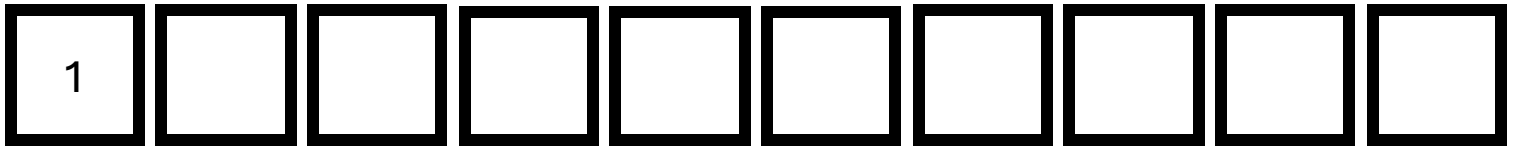


Merge & Count

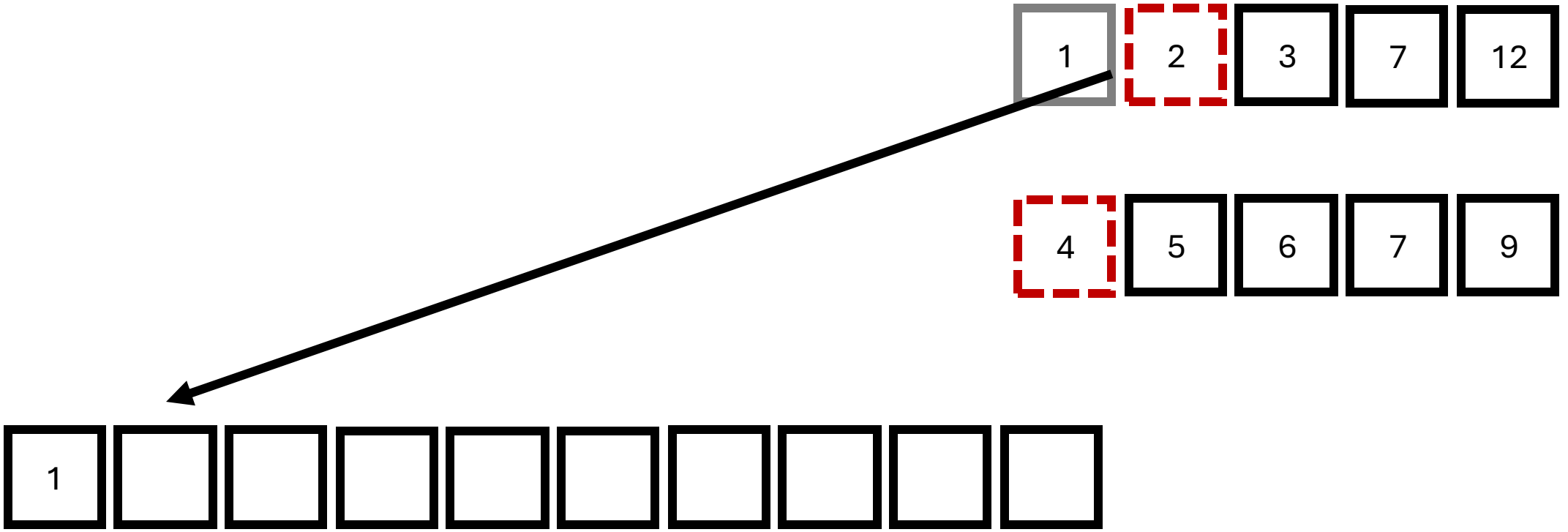
Answer: None because smaller than everything?



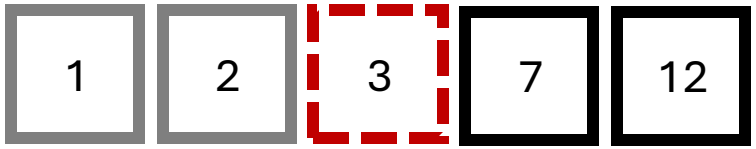


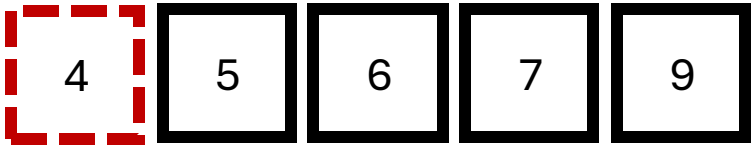


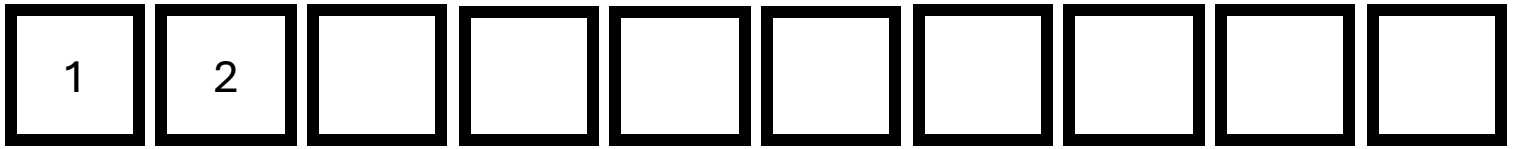
Merge & Count



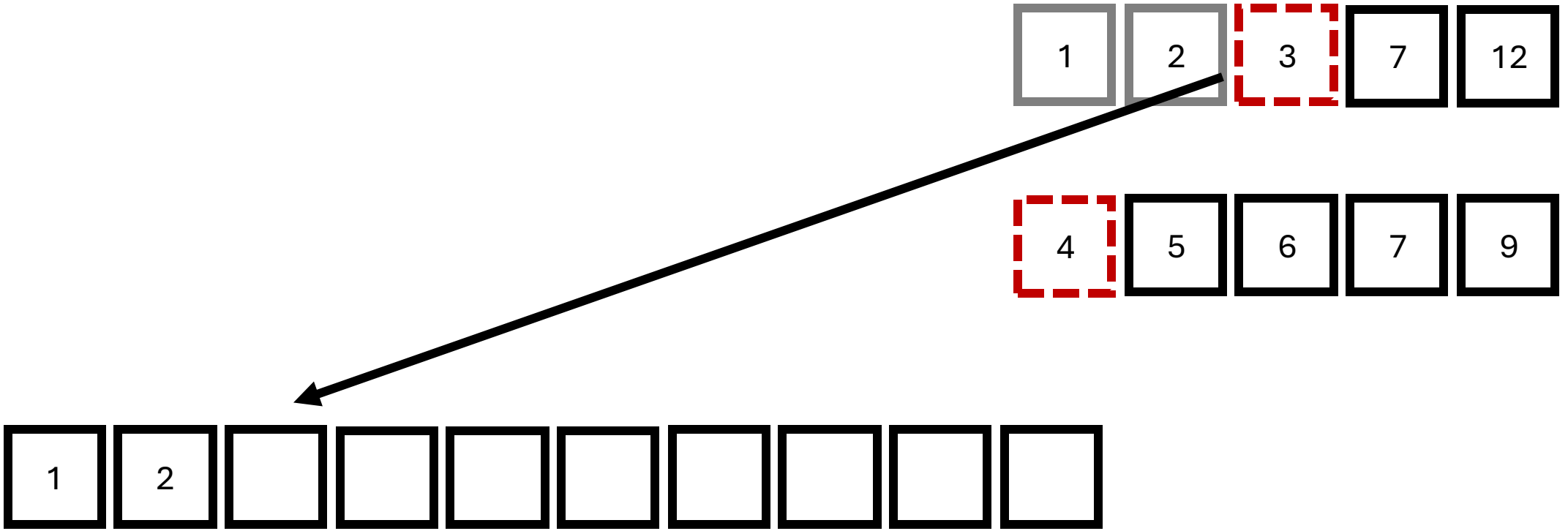
Merge & Count



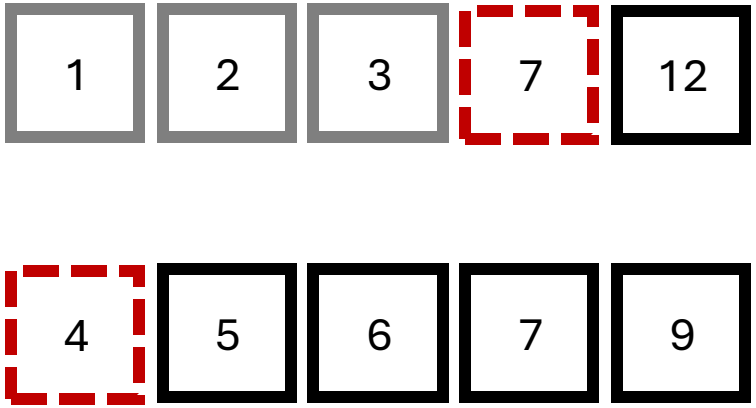


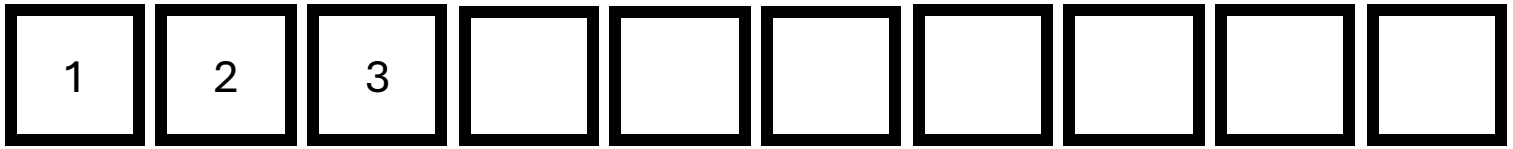


Merge & Count



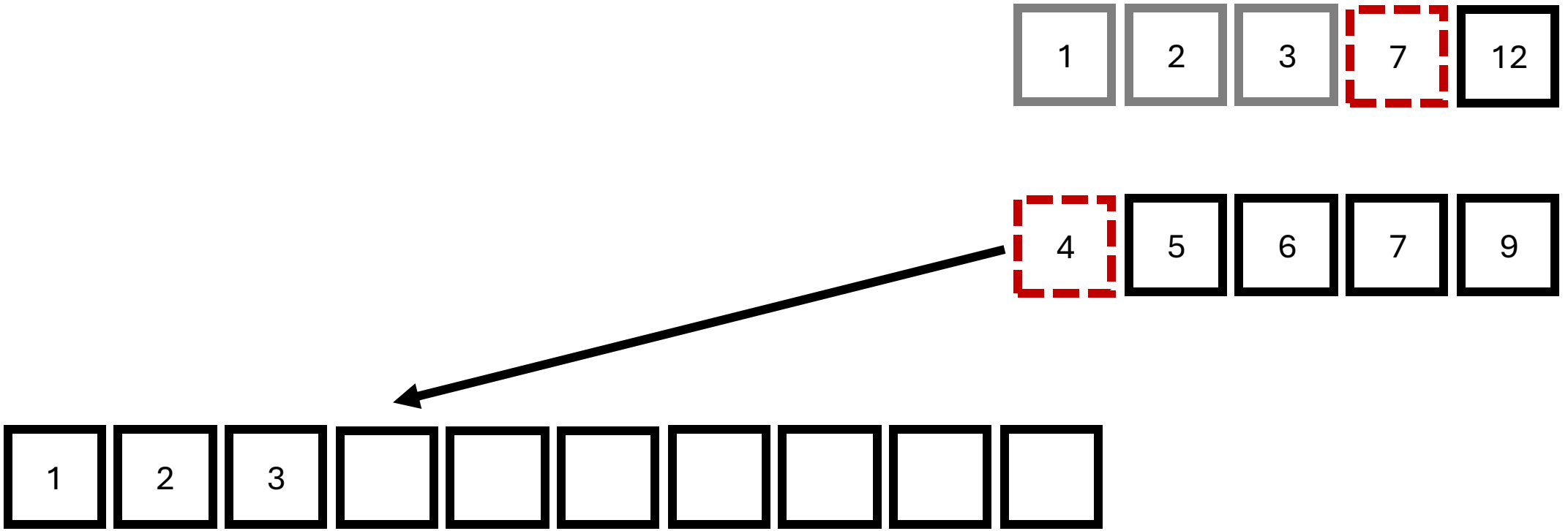
Merge & Count





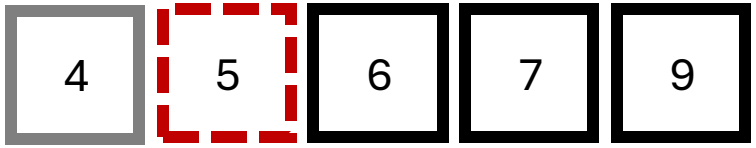
Merge & Count

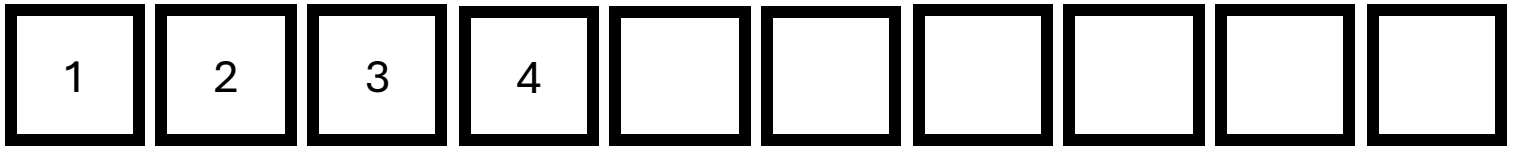
Question: How many inversions involving 4?



Merge & Count

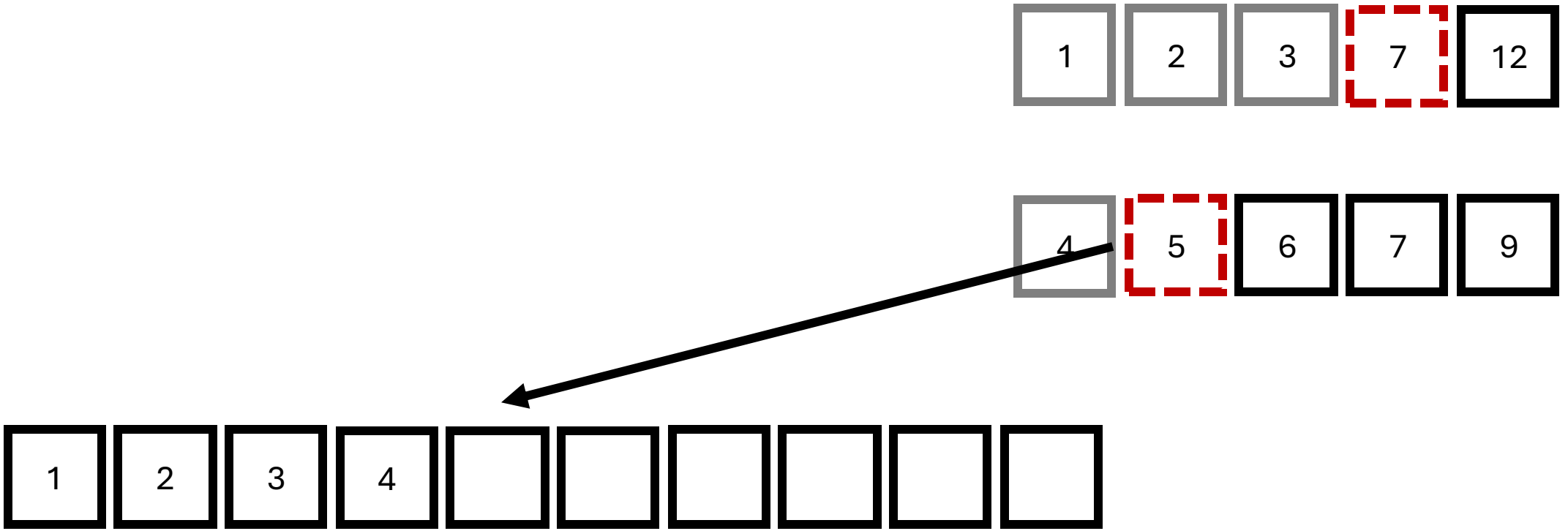
Answer: There are two: (7,4) and (12,4)





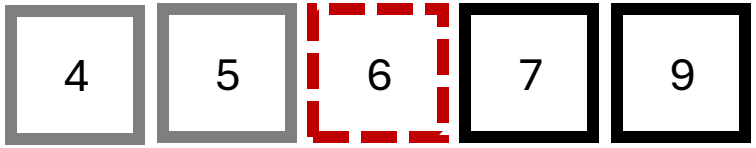
Merge & Count

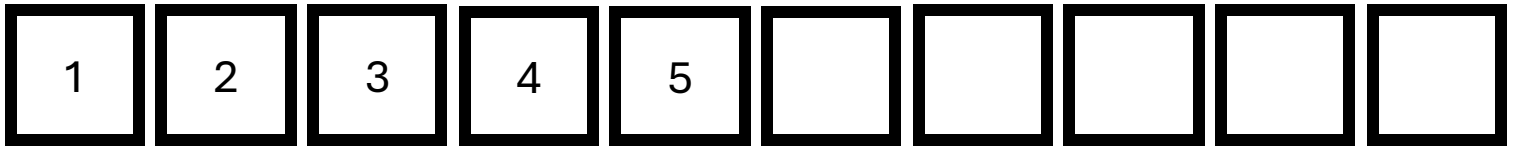
Question: How may inversion involving 5?



Merge & Count

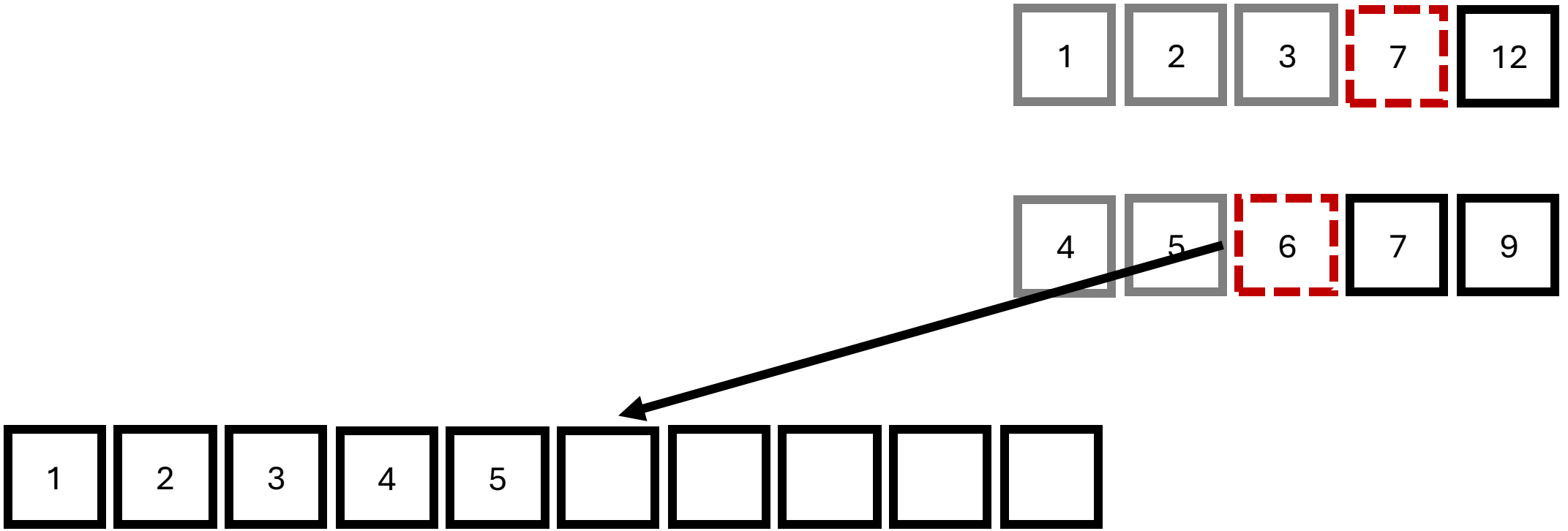
Answer: There are two: (7,5) and (12,5)





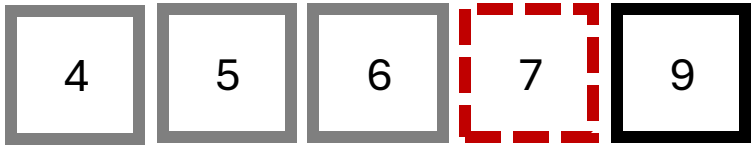
Merge & Count

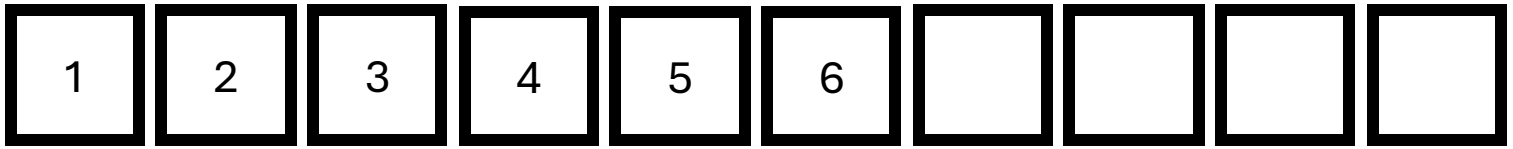
Question: How may inversion involving 6?



Merge & Count

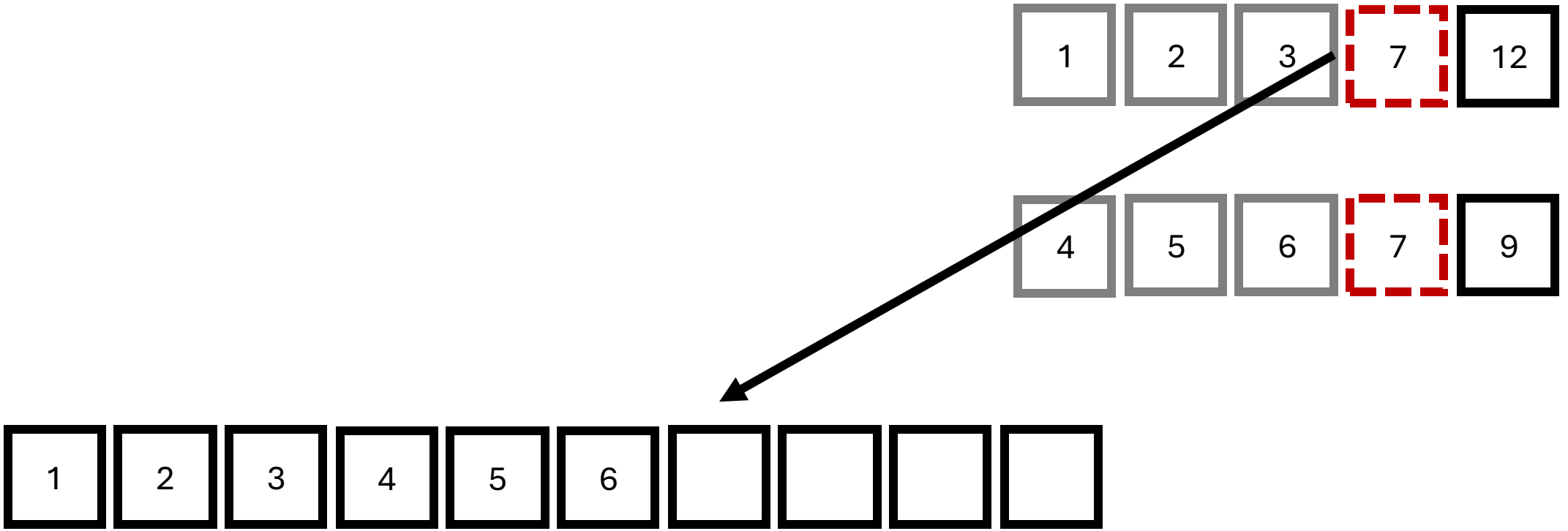
Answer: There are two: (7,6) and (12,6)





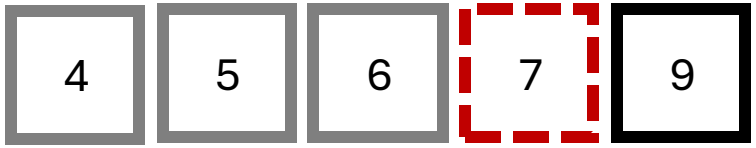
Merge & Count

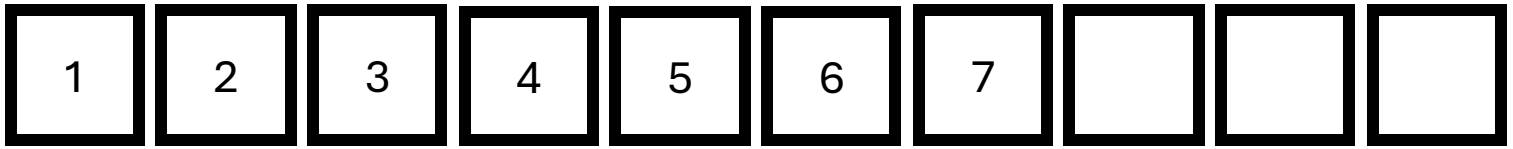
Question: How may inversion involving the top 7?



Merge & Count

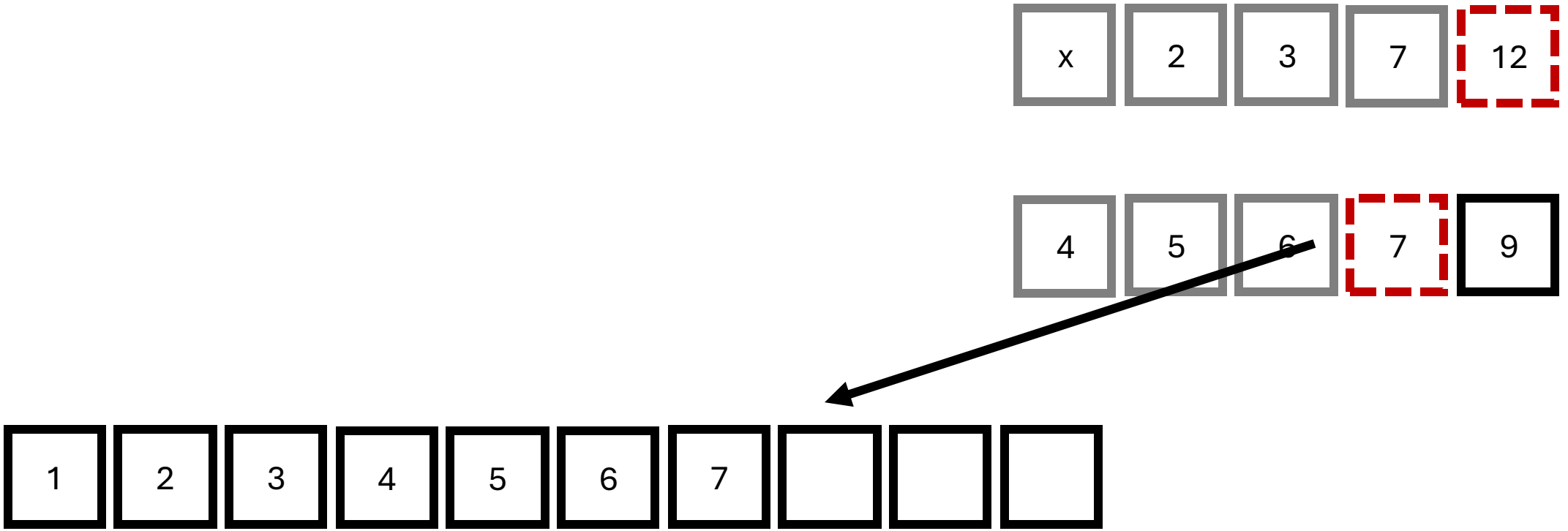
Answer: There were 3, but we already listed them.





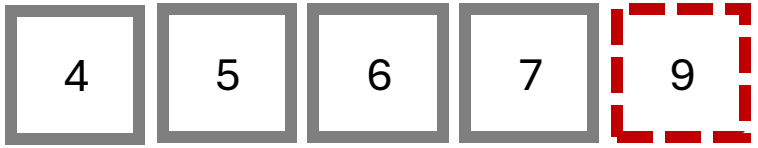
Merge & Count

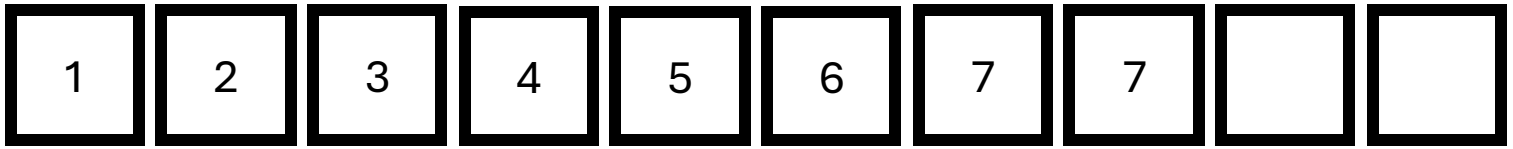
Question: How may inversion involving the bottom 7?



Merge & Count

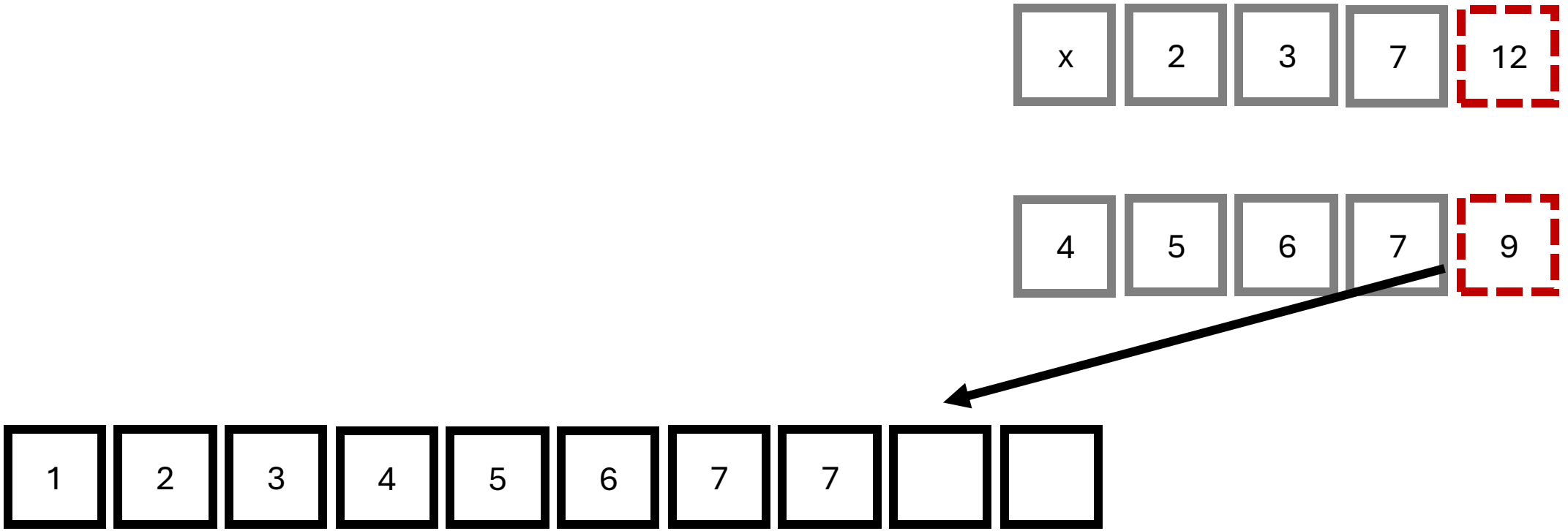
Answer: There is 1: (12,7)





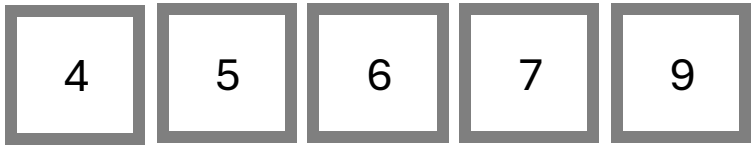
Merge & Count

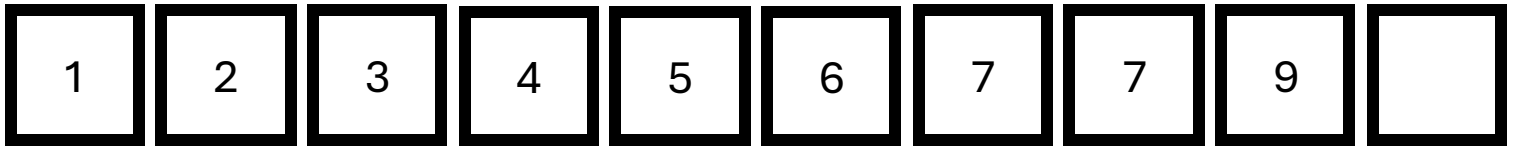
Question: How many involving 9?



Merge & Count

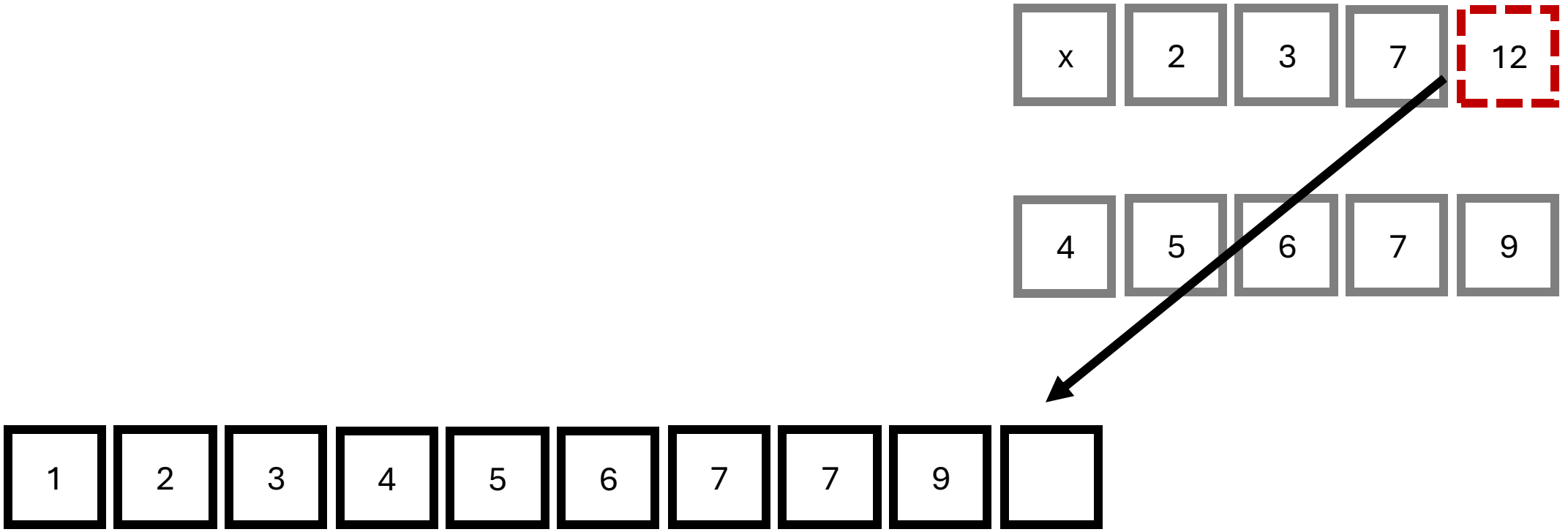
Answer: There is 1: (12,9)





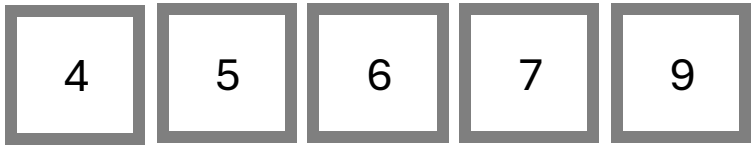
Merge & Count

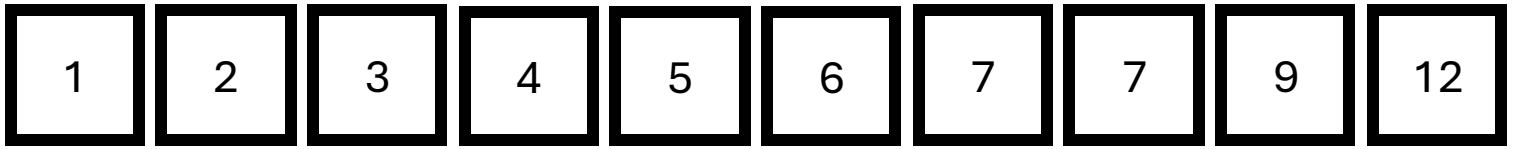
Question: How many involving 12?



Merge & Count

Answer: There were 5 but we listed them already.

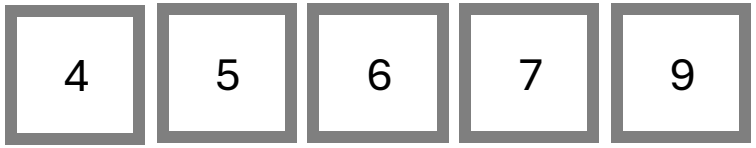


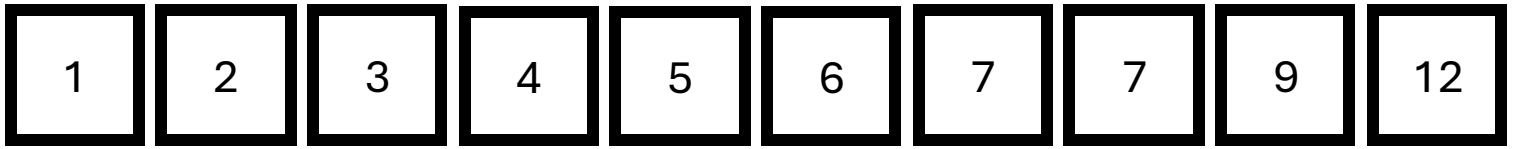


Merge & Count

Question: How would you describe our new merge process?

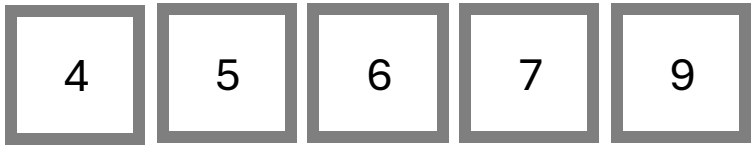
Question: What was the rule for counting inversions?

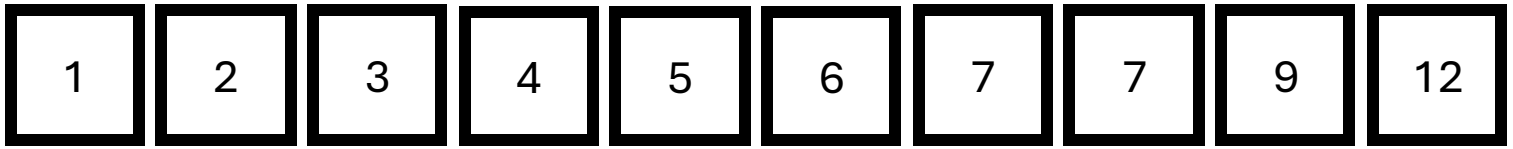




Merge & Count

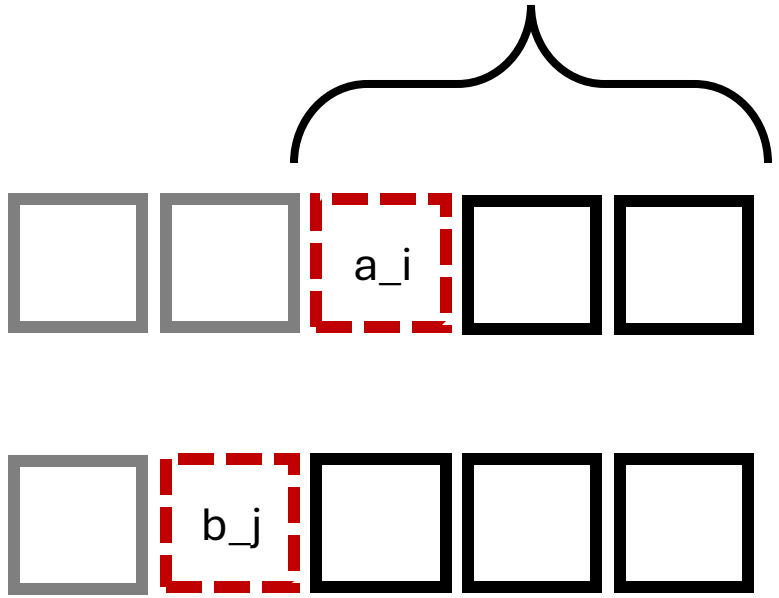
Answer: When we added to our list from the second smaller list, we knew that there was going to be a new inversion for every remaining element in the top list.

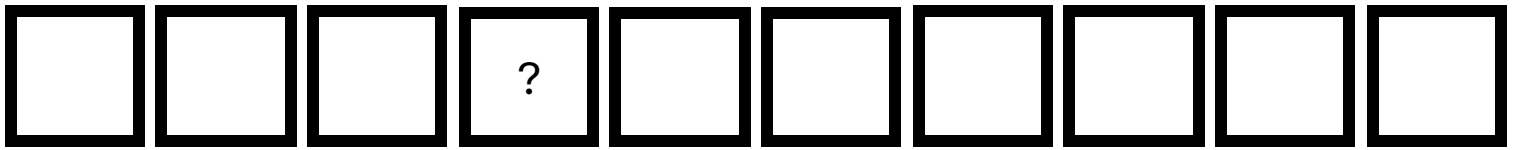




Merge & Count

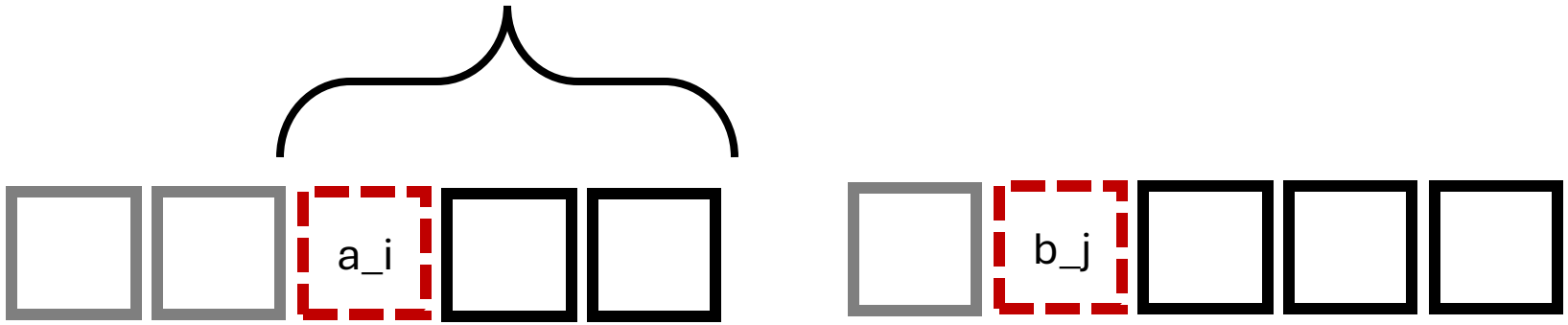
Observation: If b_j < a_i then we know that b_j is going to be in |A|-i inversions.





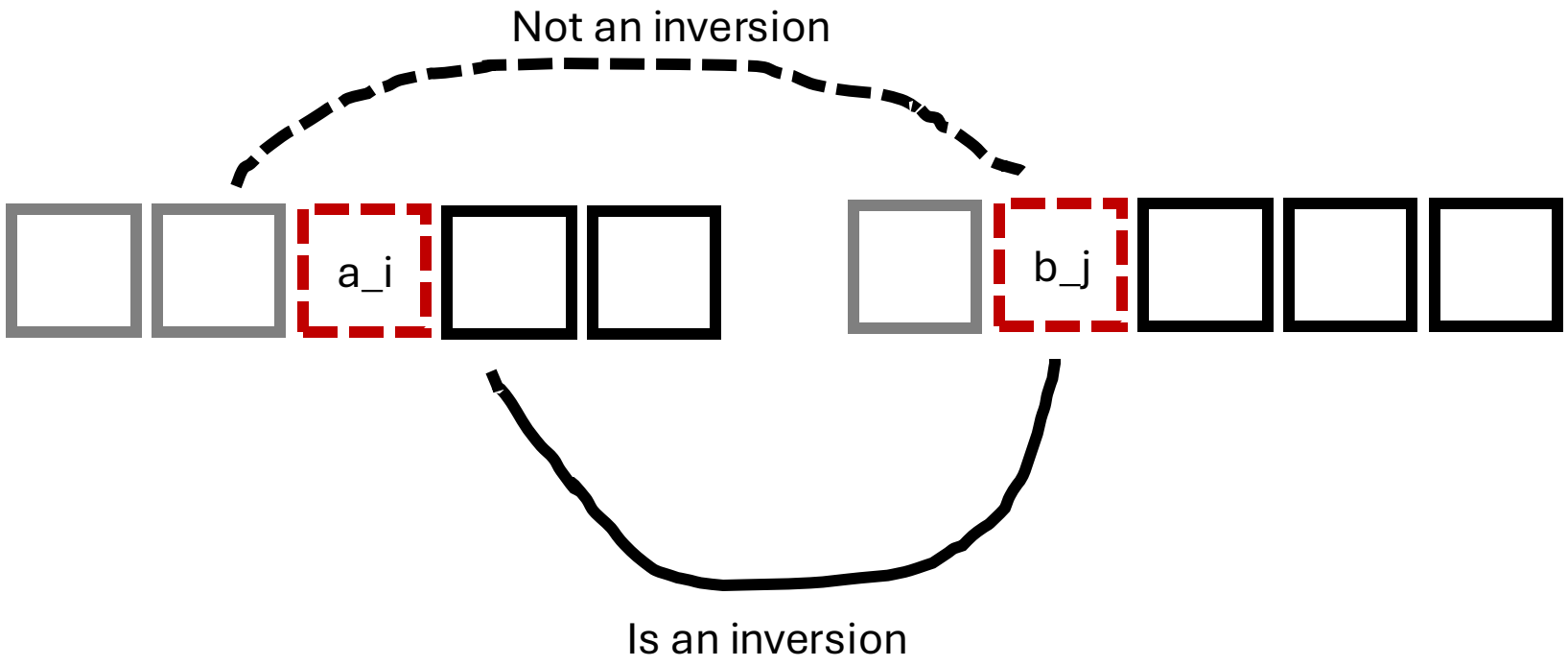
Merge & Count

Observation: If b_j < a_i then we know that b_j is going to be in |A|-i inversions.



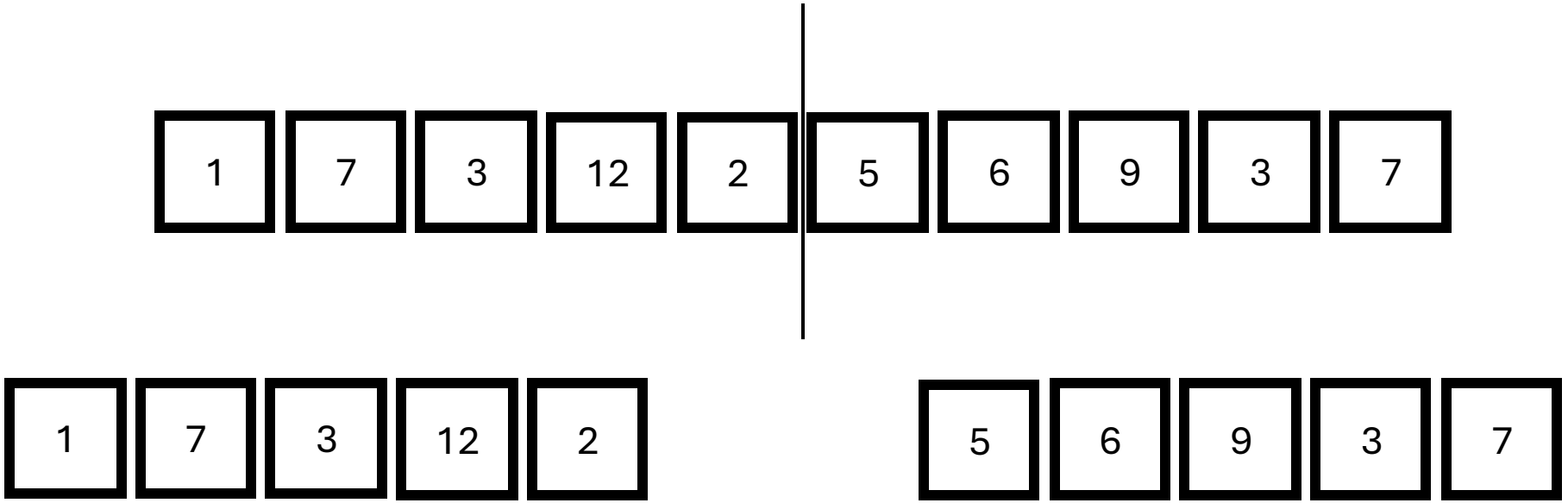
Merge & Count

- **We can see that a_(i-1) must have been smaller**
- **We can also see that a_(i+1) must be bigger.**



Divide & Conquer

- **Question**: How should we use this merge and count algorithm?



Divide & Conquer Algorithm

- We will use the logic of the previous few slides to make a Merge-and-Count(A,B) algorithm that will merge two sorted lists and count the number of “spanning” inversions.

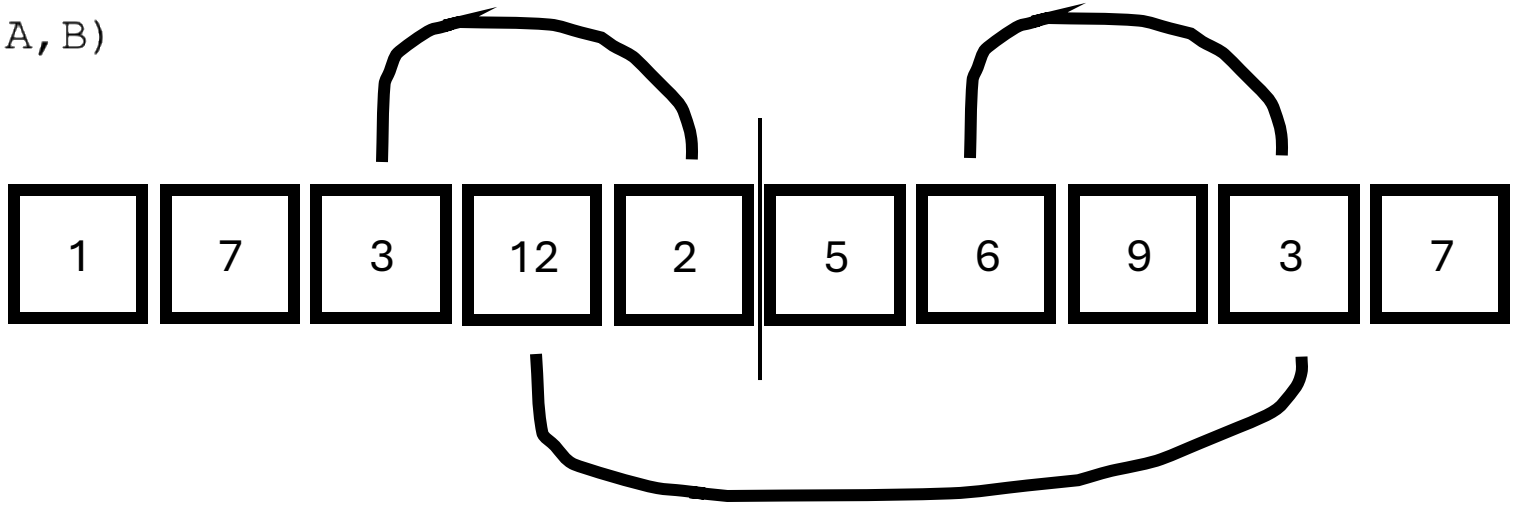
- We will now make a new algorithm called Sort-and-Count(L) that will take a list and return the list sorted and return the number of inversions before being sorted.

Sort-and-Count

1. Input: list L of length n
2. If the list has on element:
3. there are no inversions
4. Else:
5. Divide the list into two halves:
6. A contains first $\lceil n/2 \rceil$ elements
7. B contains second $\lfloor n/2 \rfloor$ elements
8. (r,A) = Sort-and-Count(A)
9. (q,B) = Sort-and-Count(B)
10. (k,L) = Merge-and-Count(A,B)
11. Return (r+q+k,L)

When is each type of inversion counted?

```
1. Input: list L of length n
2.  If the list has on element:
3.   there are no inversions
4.  Else:
5.   Divide the list into two halves:
6.    A contains first ⌈n/2⌉ elements
7.    B contains second ⌊n/2⌋ elements
8.   (r,A) = Sort-and-Count(A)
9.   (q,B) = Sort-and-Count(B)
10.  (k,L) = Merge-and-Count(A,B)
11. Return (r+q+k,L)
```



Sort-and-Count Runtime?

1. Input: list L of length n
2. If the list has on element:
3. there are no inversions
4. Else:
5. Divide the list into two halves:
6. A contains first $\lceil n/2 \rceil$ elements
7. B contains second $\lfloor n/2 \rfloor$ elements
8. (r,A) = Sort-and-Count(A)
9. (q,B) = Sort-and-Count(B)
10. (k,L) = Merge-and-Count(A,B)
11. Return (r+q+k,L)

Sort-and-Count Runtime?

- Observations:
  - Takes O(n) time to divide.
  - Takes 2T(n/2) time to do recursive calls.
  - Takes O(n) time to merge.
  - Takes O(1) time to do base case.

```
1. Input: list L of length n
2.  If the list has on element:
3.   there are no inversions
4.  Else:
5.   Divide the list into two halves:
6.    A contains first ⌈n/2⌉ elements
7.    B contains second ⌊n/2⌋ elements
8.   (r,A) = Sort-and-Count(A)
9.   (q,B) = Sort-and-Count(B)
10.  (k,L) = Merge-and-Count(A,B)
11. Return (r+q+k,L)
```

Sort-and-Count Runtime

- We have the same recurrence we had for mergesort and if we solve it using the methods from before, we get the same runtime of O(nlog(n)).

```
1. Input: list L of length n
2.  If the list has on element:
3.   there are no inversions
4.  Else:
5.   Divide the list into two halves:
6.    A contains first ⌈n/2⌉ elements
7.    B contains second ⌊n/2⌋ elements
8.   (r,A) = Sort-and-Count(A)
9.   (q,B) = Sort-and-Count(B)
10.  (k,L) = Merge-and-Count(A,B)
11. Return (r+q+k,L)
```

Sort-and-Count Runtime?

- **Question**: What would you change to get the list of all inversions?
- **Question**: How would this change the runtime?

```
1. Input: list L of length n
2.  If the list has on element:
3.   there are no inversions
4.  Else:
5.   Divide the list into two halves:
6.    A contains first ⌈n/2⌉ elements
7.    B contains second ⌊n/2⌋ elements
8.   (r,A) = Sort-and-Count(A)
9.   (q,B) = Sort-and-Count(B)
10.  (k,L) = Merge-and-Count(A,B)
11. Return (r+q+k,L)
```

Sort-and-Count Runtime?

- **Answer**: You'd want to change your Sort-and-Count to return list of inversions.
- **Answer**: This would take longer because we do have to list all pairs in some cases.

```
1. Input: list L of length n
2.  If the list has on element:
3.   there are no inversions
4.  Else:
5.   Divide the list into two halves:
6.    A contains first ⌈n/2⌉ elements
7.    B contains second ⌊n/2⌋ elements
8.   (r,A) = Sort-and-Count(A)
9.   (q,B) = Sort-and-Count(B)
10.  (k,L) = Merge-and-Count(A,B)
11.  Return (r+q+k,L)
```