Divide and Conquer: Counting Inversions

Rank Analysis

- Collaborative filtering
  - matches your preference (books, music, movies, restaurants) with that of others
  - finds people with similar tastes
  - recommends new things to you based on purchases of these people
- Meta-search tools
  - same query to many search engines
  - synthesize result by looking for similarities of resulting rankings
- The basis: compare the **similarity of two rankings**

What's similar?

Given numbers 1 to n (the things) rank these according to your preference

- You get some permutation of 1..n
- Compare to someone else's permutation

Extreme similarity

- somebody else's ranking is exactly the same

Extreme dissimilarity

- somebody else's ranking is exactly the opposite

In the middle:

- count the number of out of place rankings

Simplify it

Count the number of **inversions** of a ranking

- $r_1, r_2, \ldots, r_n$
- count the number of out of order pairs
  - $i<j \quad r_i>r_j$
- eg: 2 1 4 3 5
- 2 inversions: (2,1) (4,3)

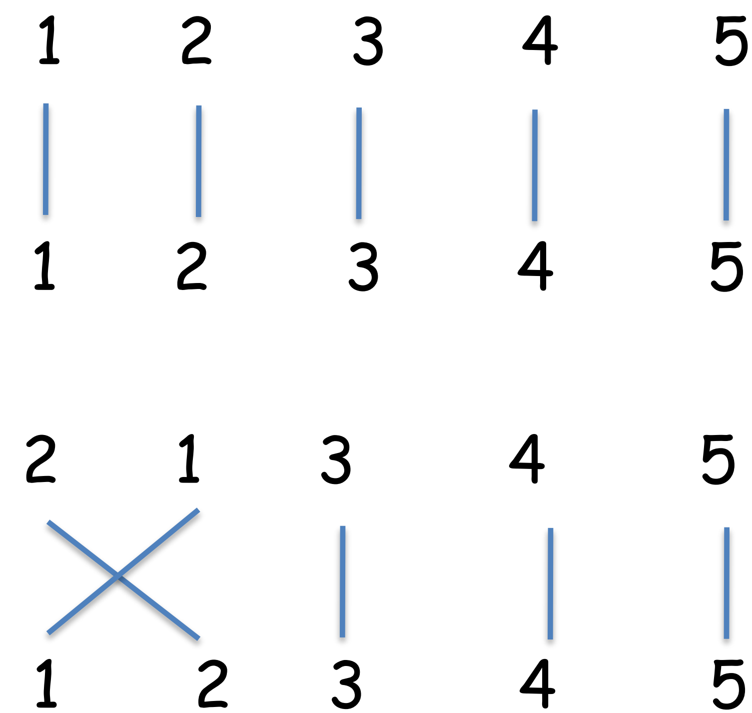
Why is this synonymous with comparing two different rankings?

Because we can re-number, such that one of the rankings becomes 1,2,...,n

Visualizing inversions

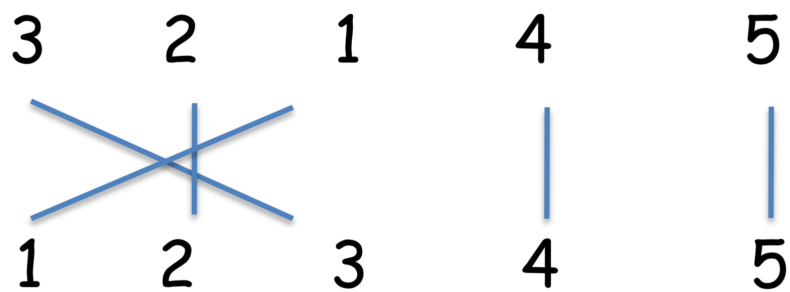
zero inversions

one inversion



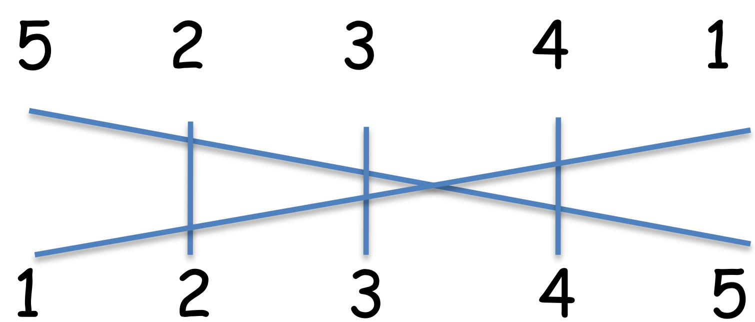
Visualizing inversions

how many?



enumerate them

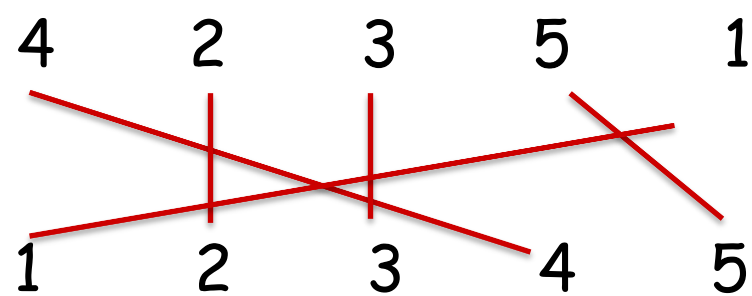
how many?



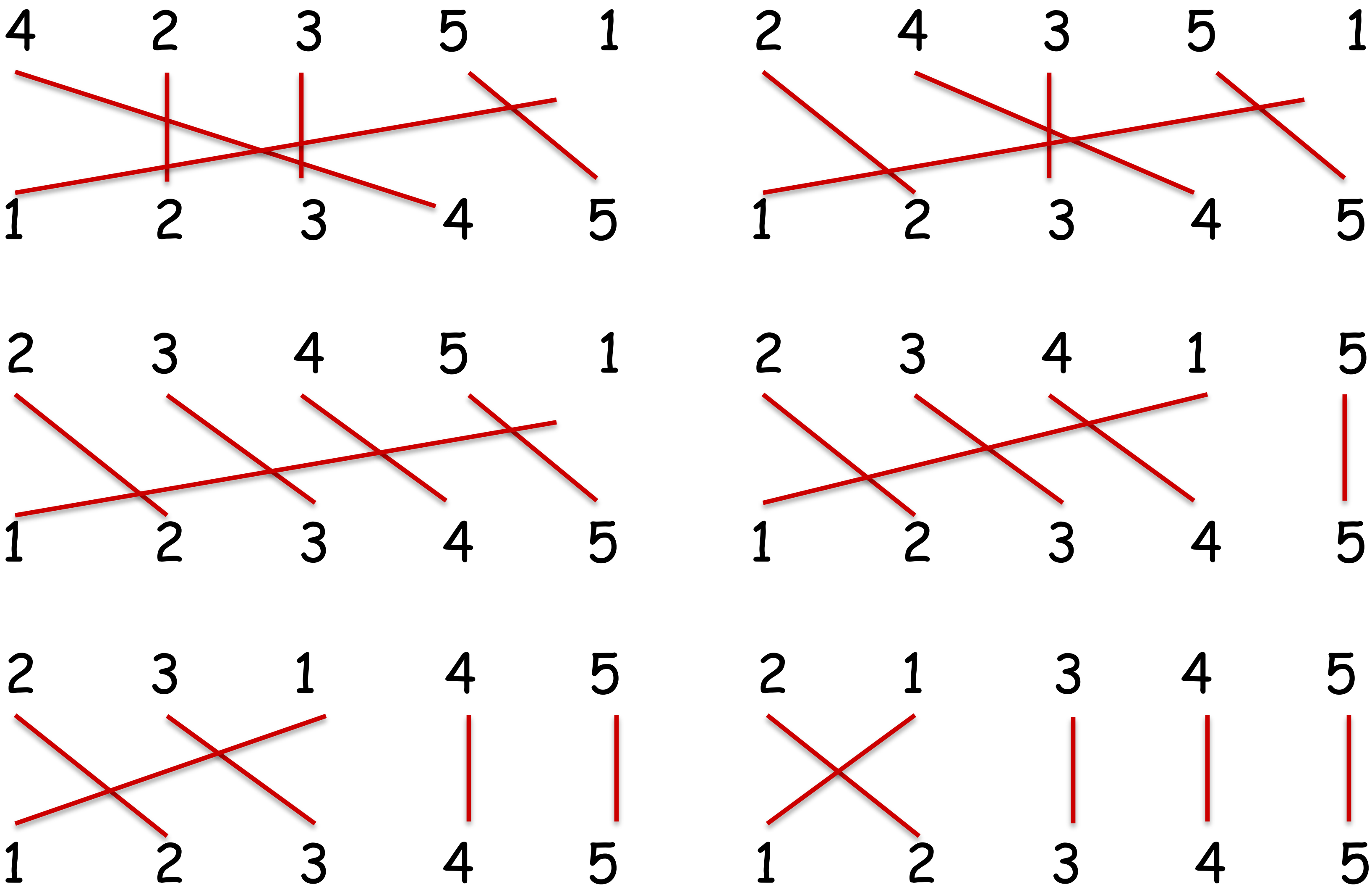
Sort

Does Bubble sort count inversions?
Selection sort?
Insertion sort?
These are $O(n^2)$

Do these sorts on:
and see what happens



Do it to it


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

Can we do better?

Notice: there are potentially $n*(n-1)/2$ inversions

Bubble and insertion sort count each individual inversion

To do better we must not count each individual inversion

Think of merge sort

- in merge sort we do not swap all elements that are out of order with each other, we make larger distance "swaps"
- if we can merge sort and keep track of the number of inversions we may get an $O(n \log n)$ algorithm

Eg: [4 2 3 5 1]

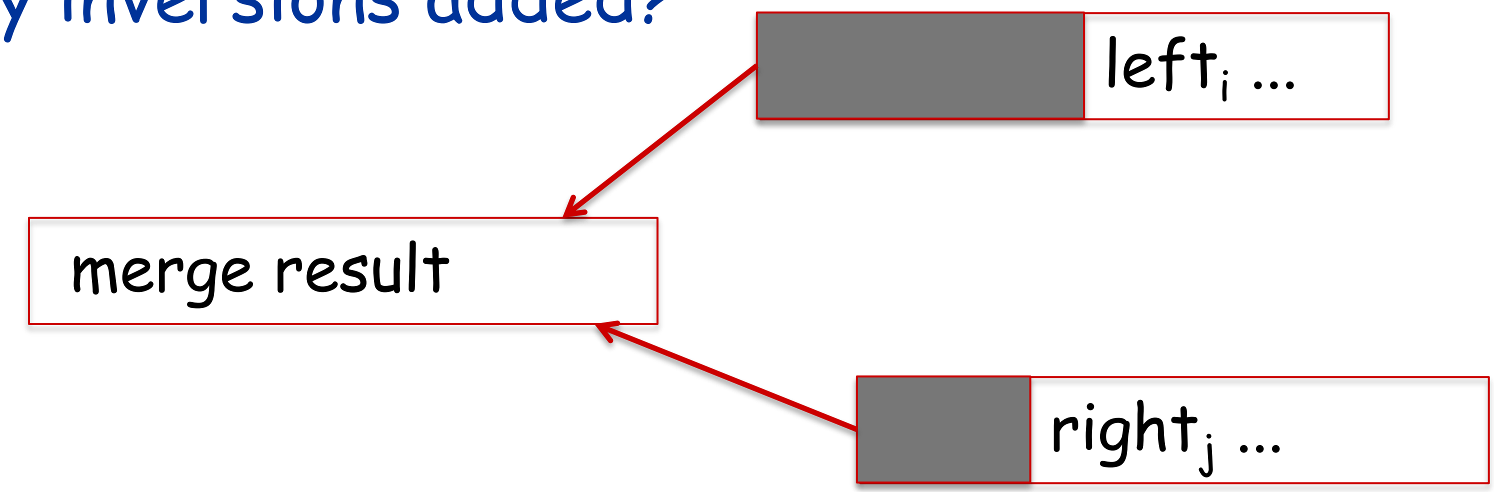
sort [4 2 3 5 1]

- sort LEFT: [4 2 3]
  - sort left: [4 2] → [2 4]:1 inversion
  - sort right: [3]
  - merge(left,right) → [2 3 4] 1 inversions (3 jumps over 4)
- sort RIGHT: [5 1] → [1 5] 1 inversion
- merge(LEFT,RIGHT) →[1 2 3 4 5]
  3 inversions (1 jumps over 2,3 & 4)

Total inversions: 1+1+1+3=6 (go check the visualization)

The algorithm

While merging in merge sort keep track of the number of inversions.
When merging an element from left: no inversions added
When merging an element from right: how many inversions added?



As many elements as are remaining in left,
because the element from the right jumps over them

Counting Inversions: Algorithm

```
Sort-and-Count(L)
   if list L has one element
      return 0 and the list L
   divide the list into two halves A and B
   (rA, A) ← Sort-and-Count(A)
   (rB, B) ← Sort-and-Count(B)
   (r , R) ← Merge-and-Count(A, B)
   return r = rA + rB + r and the sorted list R
```

```
Merge-and-Count(L,R)
  count = 0
  while L and R not empty:
    append smallest of Li and  Rj  to result
    if Rj smallest
       add number of elements remaining in L to count
  if one list empty
    append the other one to result
  return count, result
```

Running time

Just like merge sort, the sort and count algorithm running time satisfies:

$$T(n) = 2\,T(n / 2) + cn$$

Running time is therefore $O(n \log n)$

Repeated substitution

Claim. If T(n) satisfies this recurrence, then T(n) = cn $\log_2$ n.

$$\mathrm{T}(n) = \begin{cases} c & \text{if } n = 1 \\ \underbrace{2T(n/2)}_{\text{sorting both halves}} + \underbrace{cn}_{\text{merging}} & \text{otherwise} \end{cases}$$

For n > 1:

$$\begin{aligned} T(n) &= 2T(n/2) && + \ cn \\ &= 4T(n/4) && + \ cn + 2n/2 \\ &= 8T(n/8) && + cn + cn + 4cn/4 \\ &\cdots \\ &= 2^{\log_2 n} T(1) && + \underbrace{cn \ + \cdots + \ cn}_{\log_2 n} \\ &= O(n \log_2 n) \end{aligned}$$

mergesort: Recurrence Analysis

$$f(n) = a \cdot f(n/b) + cn^d$$

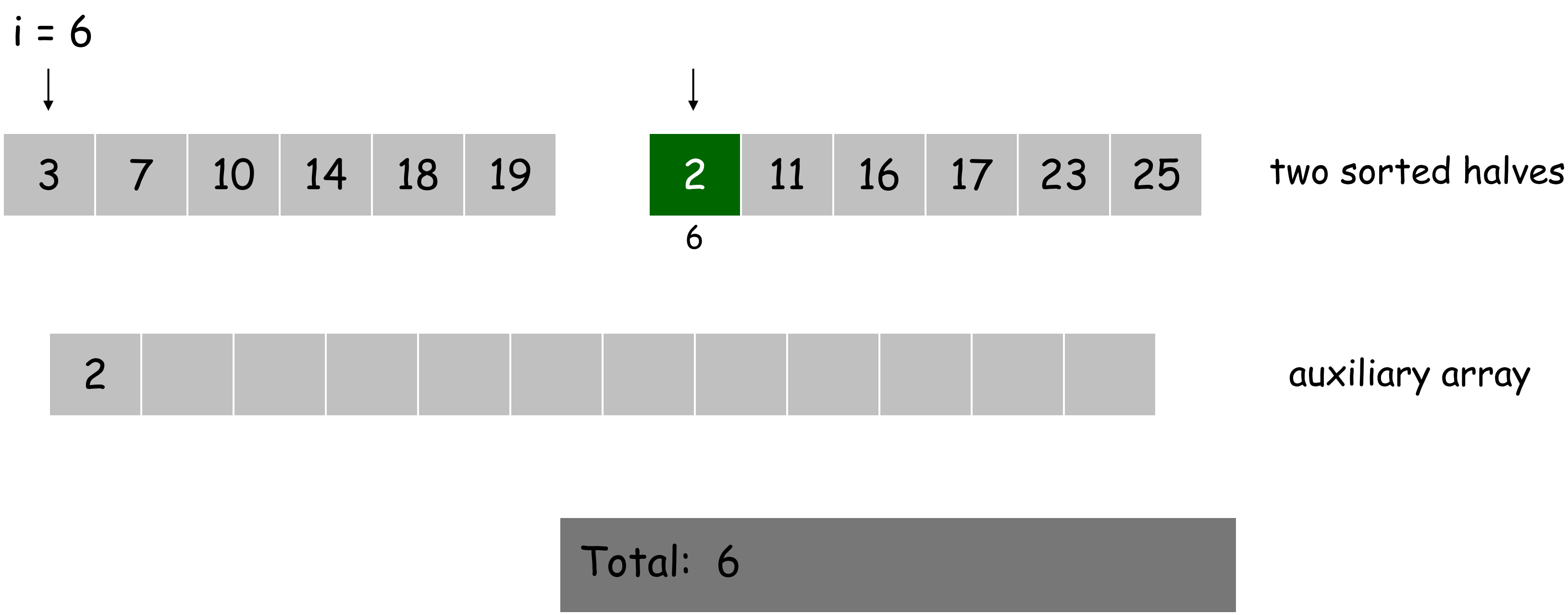
a =
b =
d =
O(?)

$$f(n) = \begin{cases} O\left(n^d\right) & \text{if } a < b^d \\ O\left(n^d \log n\right) & \text{if } a = b^d \\ O\left(n^{\log_b a}\right) & \text{if } a > b^d \end{cases}$$

Merge and Count

Merge and count step.

- Given two sorted halves, count number of inversions where a_i and a_j are in different halves.
- Combine two sorted halves into sorted whole.



Merge and Count

Merge and count step.

- Given two sorted halves, count number of inversions where a_i and a_j are in different halves.
- Combine two sorted halves into sorted whole.

i = 6

3	7	10	14	18	19		2	11	16	17	23	25
							6					

two sorted halves

2											

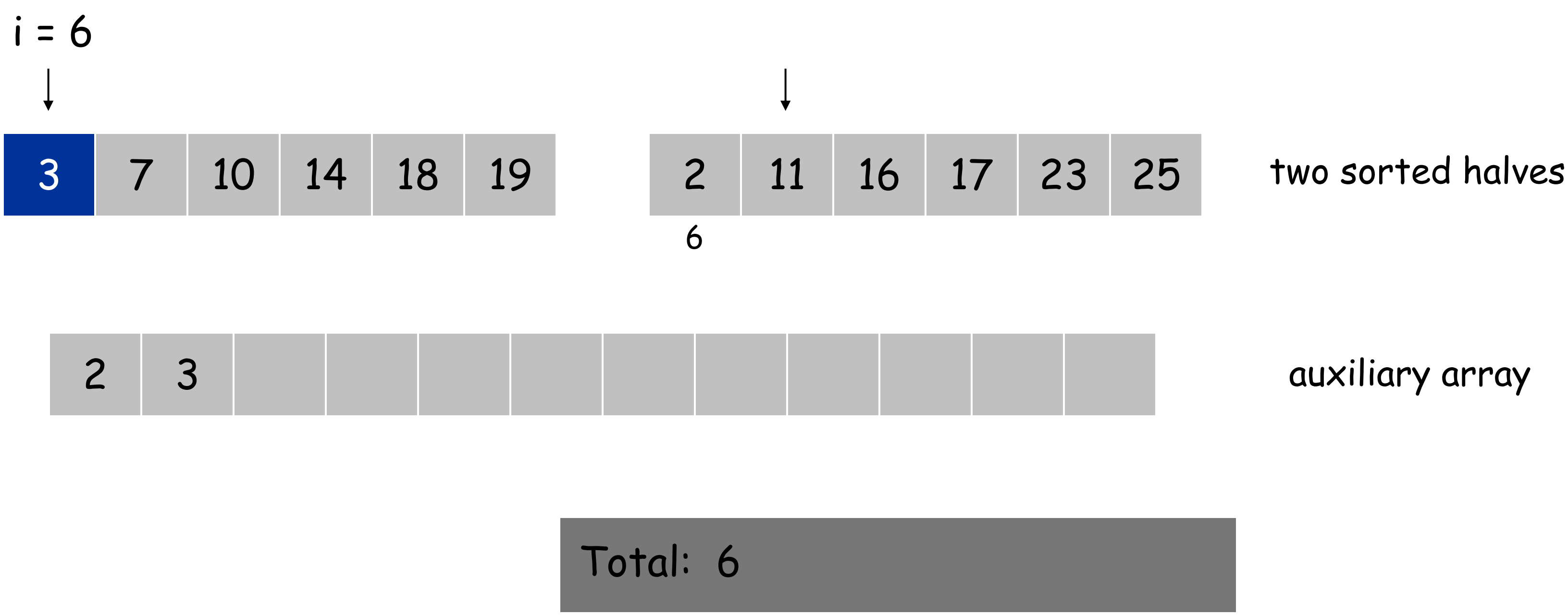
auxiliary array

Total: 6

Merge and Count

Merge and count step.

- Given two sorted halves, count number of inversions where a_i and a_j are in different halves.
- Combine two sorted halves into sorted whole.



Merge and Count

Merge and count step.

- Given two sorted halves, count number of inversions where a_i and a_j are in different halves.
- Combine two sorted halves into sorted whole.

i = 5

3	7	10	14	18	19		2	11	16	17	23	25	two sorted halves
							6						

2	3											auxiliary array

Total: 6

Merge and Count

Merge and count step.

- Given two sorted halves, count number of inversions where a_i and a_j are in different halves.
- Combine two sorted halves into sorted whole.

i = 5

3	7	10	14	18	19

2	11	16	17	23	25
6					

two sorted halves

2	3	7									

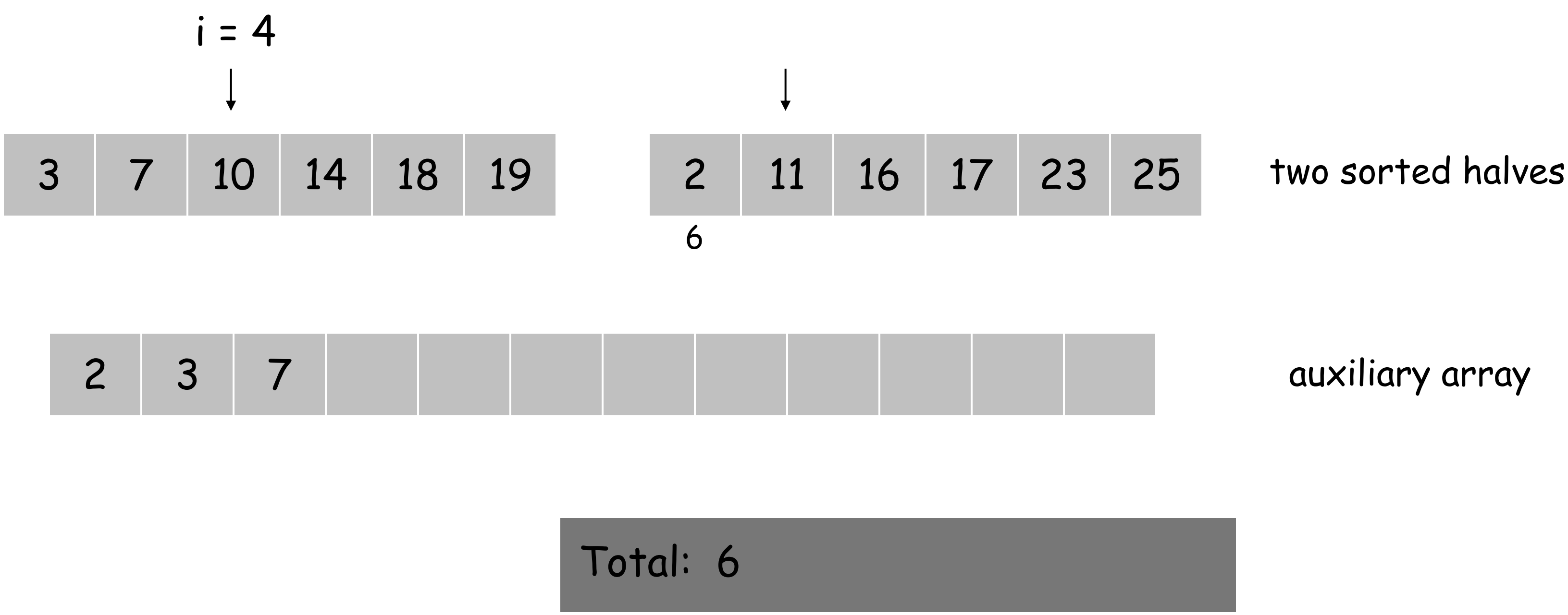
auxiliary array

Total: 6

Merge and Count

Merge and count step.

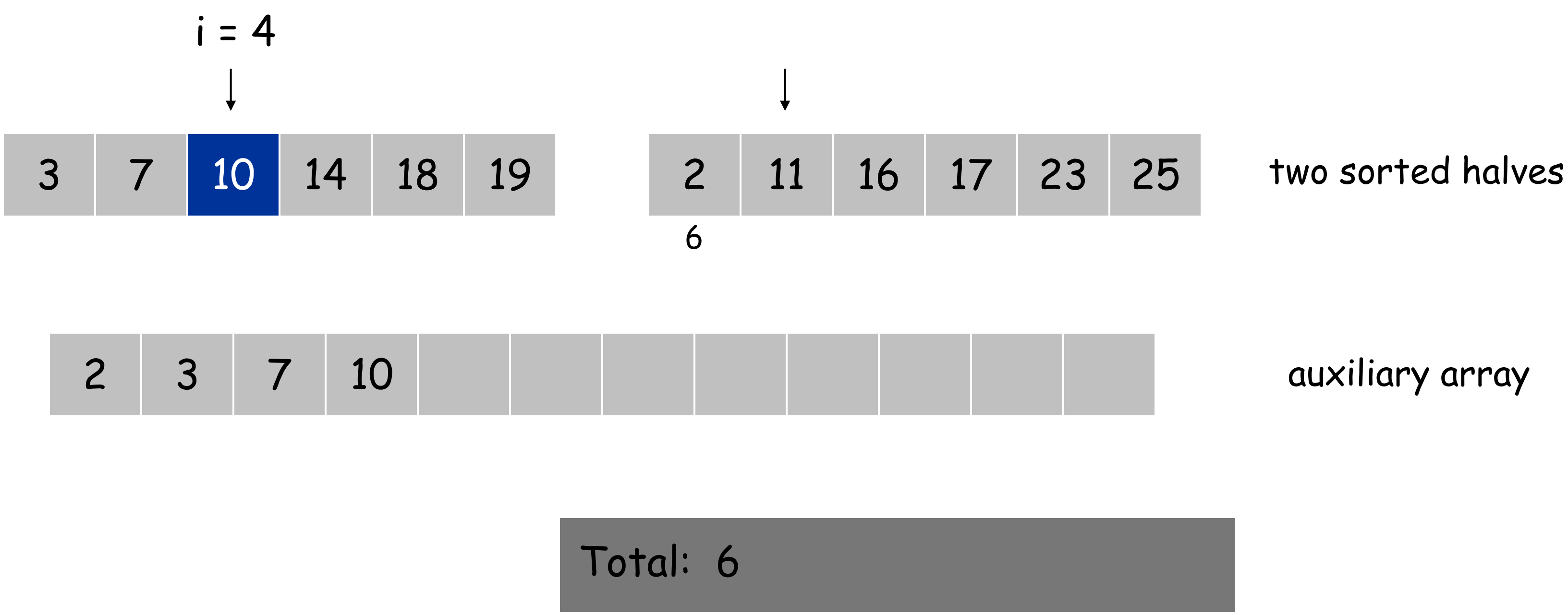
- Given two sorted halves, count number of inversions where a_i and a_j are in different halves.
- Combine two sorted halves into sorted whole.



Merge and Count

Merge and count step.

- Given two sorted halves, count number of inversions where a_i and a_j are in different halves.
- Combine two sorted halves into sorted whole.



Merge and Count

Merge and count step.

- Given two sorted halves, count number of inversions where a_i and a_j are in different halves.
- Combine two sorted halves into sorted whole.

i = 3

3	7	10	14	18	19	2	11	16	17	23	25	two sorted halves
						6						

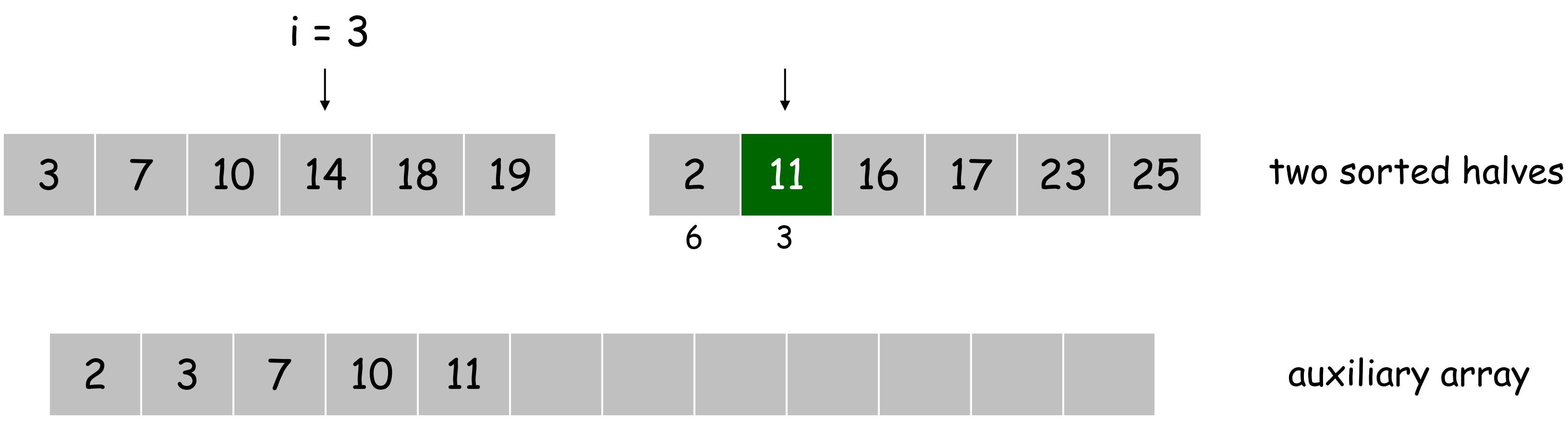
2	3	7	10									auxiliary array

Total: 6

Merge and Count

Merge and count step.

- Given two sorted halves, count number of inversions where a_i and a_j are in different halves.
- Combine two sorted halves into sorted whole.



Total: 6 + 3

Merge and Count

Merge and count step.

- Given two sorted halves, count number of inversions where a_i and a_j are in different halves.
- Combine two sorted halves into sorted whole.

i = 3

two sorted halves

3	7	10	14	18	19

2	11	16	17	23	25
6	3				

auxiliary array

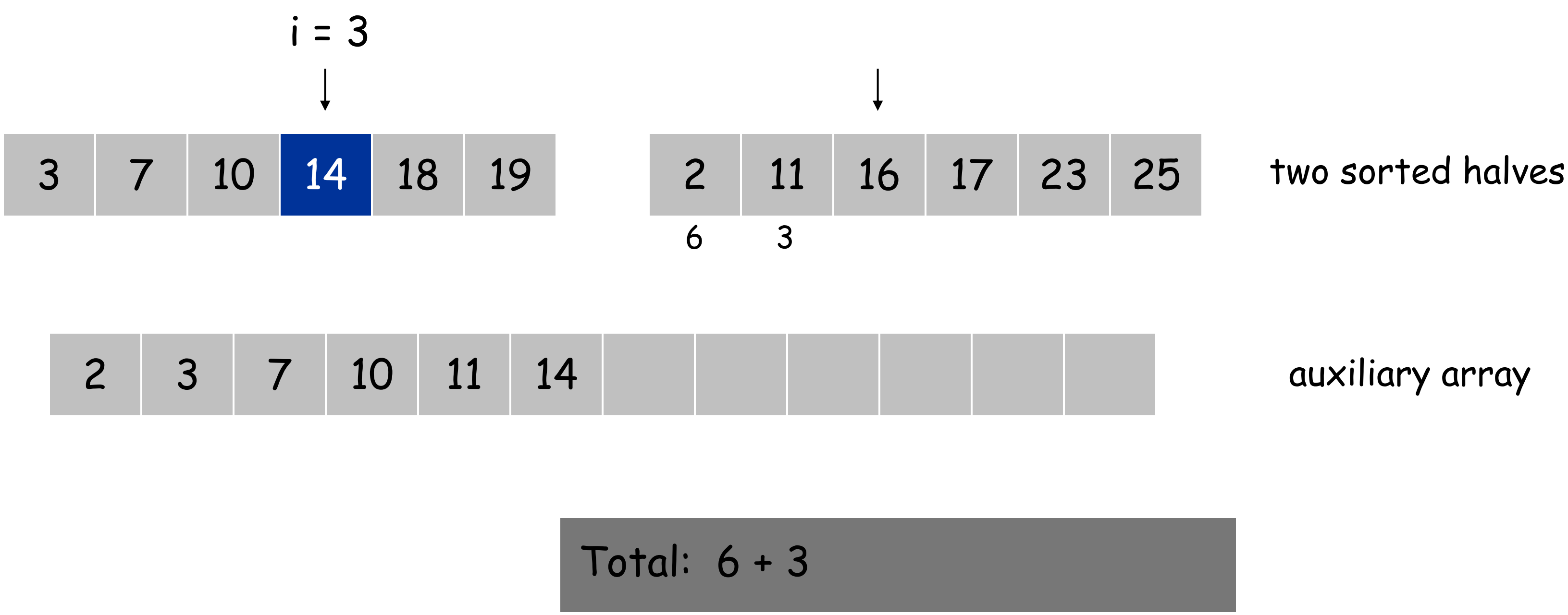
2	3	7	10	11							

Total: 6 + 3

Merge and Count

Merge and count step.

- Given two sorted halves, count number of inversions where a_i and a_j are in different halves.
- Combine two sorted halves into sorted whole.



Merge and Count

Merge and count step.

- Given two sorted halves, count number of inversions where a_i and a_j are in different halves.
- Combine two sorted halves into sorted whole.

i = 2

two sorted halves

3	7	10	14	18	19		2	11	16	17	23	25
							6	3				

auxiliary array

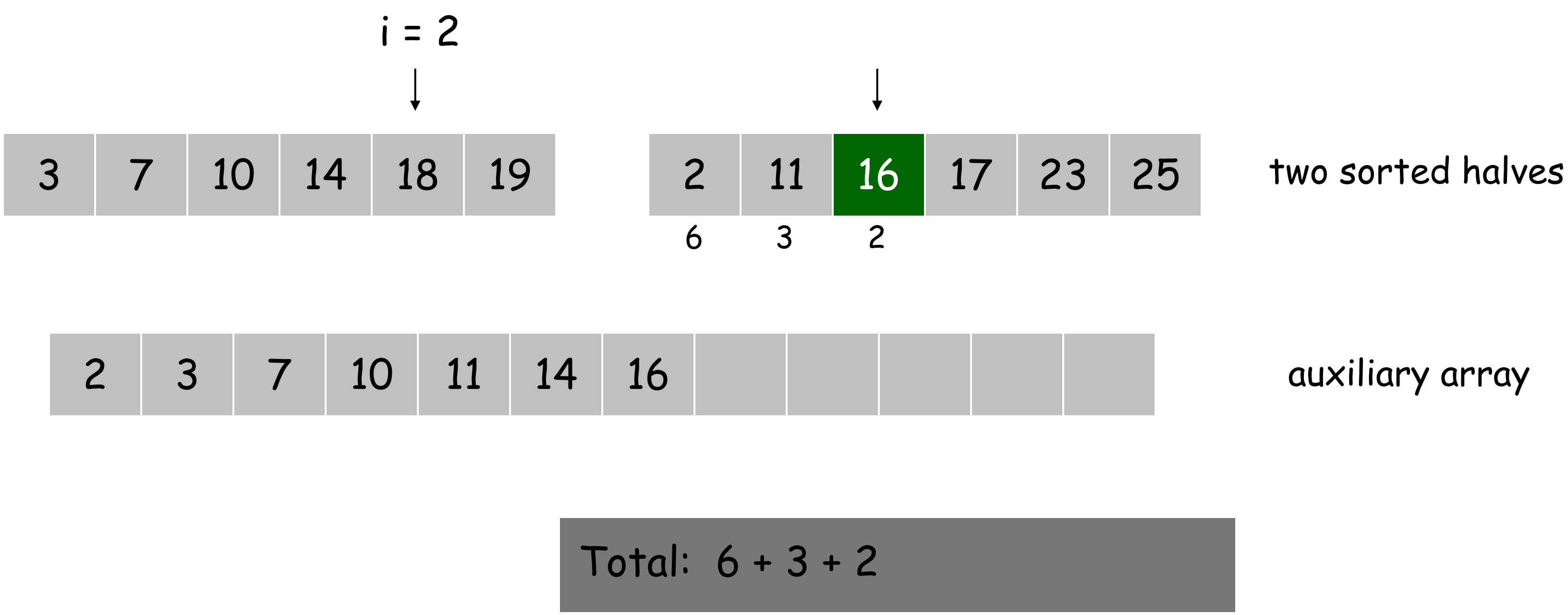
2	3	7	10	11	14						

Total: 6 + 3

Merge and Count

Merge and count step.

- Given two sorted halves, count number of inversions where a_i and a_j are in different halves.
- Combine two sorted halves into sorted whole.



Merge and Count

Merge and count step.

- Given two sorted halves, count number of inversions where a_i and a_j are in different halves.
- Combine two sorted halves into sorted whole.

i = 2

3	7	10	14	18	19	2	11	16	17	23	25	two sorted halves
						6	3	2				

2	3	7	10	11	14	16						auxiliary array

Total: 6 + 3 + 2

Merge and Count

Merge and count step.

- Given two sorted halves, count number of inversions where a_i and a_j are in different halves.
- Combine two sorted halves into sorted whole.

i = 2

two sorted halves

3	7	10	14	18	19

2	11	16	17	23	25
6	3	2	2		

auxiliary array

2	3	7	10	11	14	16	17				

Total: 6 + 3 + 2 + 2

Merge and Count

Merge and count step.

- Given two sorted halves, count number of inversions where a_i and a_j are in different halves.
- Combine two sorted halves into sorted whole.

i = 2

3	7	10	14	18	19	2	11	16	17	23	25	two sorted halves
						6	3	2	2			

2	3	7	10	11	14	16	17					auxiliary array

Total: 6 + 3 + 2 + 2

Merge and Count

Merge and count step.

- Given two sorted halves, count number of inversions where a_i and a_j are in different halves.
- Combine two sorted halves into sorted whole.

i = 2

3	7	10	14	18	19	2	11	16	17	23	25	two sorted halves
						6	3	2	2			

2	3	7	10	11	14	16	17	18				auxiliary array

Total: 6 + 3 + 2 + 2

Merge and Count

Merge and count step.

- Given two sorted halves, count number of inversions where a_i and a_j are in different halves.
- Combine two sorted halves into sorted whole.

i = 1

two sorted halves

3	7	10	14	18	19	2	11	16	17	23	25
						6	3	2	2		

auxiliary array

2	3	7	10	11	14	16	17	18			

Total: 6 + 3 + 2 + 2

Merge and Count

Merge and count step.

- Given two sorted halves, count number of inversions where a_i and a_j are in different halves.
- Combine two sorted halves into sorted whole.

i = 1

3	7	10	14	18	**19**		2	11	16	17	23	25	two sorted halves
							6	3	2	2			

2	3	7	10	11	14	16	17	18	19			auxiliary array

Total: 6 + 3 + 2 + 2

Merge and Count

Merge and count step.

- Given two sorted halves, count number of inversions where a_i and a_j are in different halves.
- Combine two sorted halves into sorted whole.

first half exhausted   i = 0

3	7	10	14	18	19		2	11	16	17	23	25
							6	3	2	2		

two sorted halves

2	3	7	10	11	14	16	17	18	19		

auxiliary array

Total: 6 + 3 + 2 + 2

Merge and Count

Merge and count step.

- Given two sorted halves, count number of inversions where a_i and a_j are in different halves.
- Combine two sorted halves into sorted whole.

i = 0

3	7	10	14	18	19		2	11	16	17	23	25
							6	3	2	2	0	

two sorted halves

2	3	7	10	11	14	16	17	18	19	23	

auxiliary array

Total: 6 + 3 + 2 + 2 + 0

Merge and Count

Merge and count step.

- Given two sorted halves, count number of inversions where a_i and a_j are in different halves.
- Combine two sorted halves into sorted whole.

i = 0

two sorted halves

3	7	10	14	18	19		2	11	16	17	23	25
							6	3	2	2	0	

auxiliary array

2	3	7	10	11	14	16	17	18	19	23	

Total: 6 + 3 + 2 + 2 + 0

Merge and Count

Merge and count step.

- Given two sorted halves, count number of inversions where a_i and a_j are in different halves.
- Combine two sorted halves into sorted whole.

i = 0

two sorted halves

3	7	10	14	18	19	2	11	16	17	23	25
						6	3	2	2	0	0

auxiliary array

2	3	7	10	11	14	16	17	18	19	23	25

Total: 6 + 3 + 2 + 2 + 0 + 0

Merge and Count

Merge and count step.

- Given two sorted halves, count number of inversions where a_i and a_j are in different halves.
- Combine two sorted halves into sorted whole.

i = 0

3	7	10	14	18	19		2	11	16	17	23	25
							6	3	2	2	0	0

two sorted halves

2	3	7	10	11	14	16	17	18	19	23	25

auxiliary array

Total: 6 + 3 + 2 + 2 + 0 + 0 = 13