Objectives

- Divide and conquer algorithms
  - Counting inversions
  - Closest pairs of points

Review

- What is a recurrence relation?
- How can you compute D&C running times?

Know Your Recurrence Relations

What algorithm has this recurrence relation?
What is that algorithm's running time?

Recurrence	Algorithm	Running Time
T(n) = T(n/2) + O(1)		
T(n) = T(n-1) + O(1)		
T(n) = 2 T(n/2) + O(1)		
T(n) = T(n-1) + O(n)		
T(n) = 2 T(n/2) + O(n)		

Know Your Recurrence Relations

What algorithm has this recurrence relation?
What is that algorithm's running time?

Recurrence	Algorithm	Running Time
T(n) = T(n/2) + O(1)	Binary Search	O(log n)
T(n) = T(n-1) + O(1)	Sequential/Linear Search	O(n)
T(n) = 2 T(n/2) + O(1)	Binary Tree Traversal	O(n)
T(n) = T(n-1) + O(n)	Selection Sort	$O(n^2)$
T(n) = 2 T(n/2) + O(n)	Merge Sort	O(n log n)

COUNTING INVERSIONS

Comparing Rankings

- To determine similarity of rankings, need a metric
- Similarity metric: number of inversions between two rankings
  - My rank: 1, 2, …, n
  - Your rank: $a_1, a_2, \ldots, a_n$
  - Movies i and j *inverted* if $i < j$ but $a_i > a_j$

Movies

	A	B	C	D	E
Me	1	2	3	4	5
You	1	3	4	2	5

Inversions:

3-2, 4-2

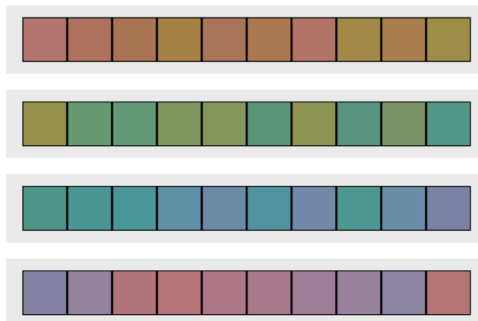
https://www.xrite.com/hue-test

Color Comparison Test

Instructions

1. The first and last color chips are fixed.
2. Drag and drop the colors in each row to arrange them by hue color
3. For best results complete all four color tests
4. Click 'Score My Test' at any time to review results

What's My Color IQ?



Score My Test

Comparing Rankings

- To determine similarity of rankings, need a metric
- Similarity metric: number of inversions between two rankings
  - My rank: 1, 2, …, n
  - Your rank: $a_1, a_2, \ldots, a_n$
  - Movies i and j *inverted* if $i < j$ but $a_i > a_j$

Naïve/Brute force solution?

Movies

	A	B	C	D	E
Me	1	2	3	4	5
You	1	3	4	2	5

Inversions:

3-2, 4-2

Brute Force Solution

- Look at every pair (i,j) and determine if they are an inversion
- Requires $\Theta(n^2)$ time
  - Note: Already an efficient algorithm but try to improve upon runtime

Towards a Better Solution…
- Can't look at each inversion individually

Counting Inversions: Divide-and-Conquer

Towards a solution…

Assume number represents where item *should* be in the list, i.e., where it is in someone else's list

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

Should be at position 5

Towards a solution

Counting Inversions: Divide-and-Conquer

- Divide: separate list into two pieces
- Conquer: recursively count inversions in each half
- Combine: count inversions where a_i and a_j are in different halves, and return sum of three quantities

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

Divide: O(1)

1	5	4	8	10	2

6	9	12	11	3	7

Conquer: 2T(n / 2)

5 blue-blue inversions

8 green-green inversions

Combine: $\Theta(n^2)$

9 blue-green inversions
5-3, 4-3, 8-6, 8-3, 8-7, 10-6, 10-9, 10-3, 10-7

What would make figuring out blue-green inversions easier?

Total = 5 + 8 + 9 = 22

Counting Inversions: Combine

Combine: count blue-green inversions

- Assume each half is sorted
- Count inversions where a_i and a_j are in different halves
- Merge two sorted halves into sorted whole

to maintain sorted invariant

3	7	10	14	18	19

2	11	16	17	23	25

- What does sorting do for us?
- What is our algorithm for counting the inversions and merging?

Note: these lists are only subsets of the original list

Counting Inversions: Combine

Combine: count blue-green inversions

- Assume each half is sorted
- Count inversions where a_i and a_j are in different halves
- Merge two sorted halves into sorted whole

to maintain sorted invariant

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

Count: O(n)

13 blue-green inversions: 6 + 3 + 2 + 2 + 0 + 0

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

Merge: O(n)

We'll run through an example in a bit…

Merge and Count

```
Merge-and-Count(A,B):
    i=0
    j=0
    inversions = 0
    output = []
    while i < A.size and j < B.size:
       output.append( min(A[i], B[j]) )
       if B[j] < A[i]:
          inversions += A.size - i
       update i or j
    Append the remainder of the non-exhausted list to
    the output
    return inversions and output
```

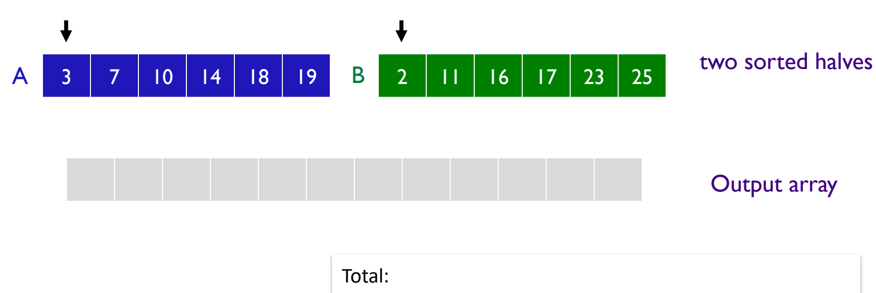
Merge and Count

Precondition: A and B are sorted

```
Merge-and-Count(A,B):
    i=0 (front of list A)
    j=0 (front of list B)
    inversions = 0
    output = []
    while A not empty and B not empty:
       output.append( min(A[i], B[j]) )
       if B[j] < A[i]:
          inversions += A.size - i (remaining elements in A)
       update i or j (whichever had smaller element)
    Append the remainder of the non-exhausted list to
    the output
    return inversions and output
```

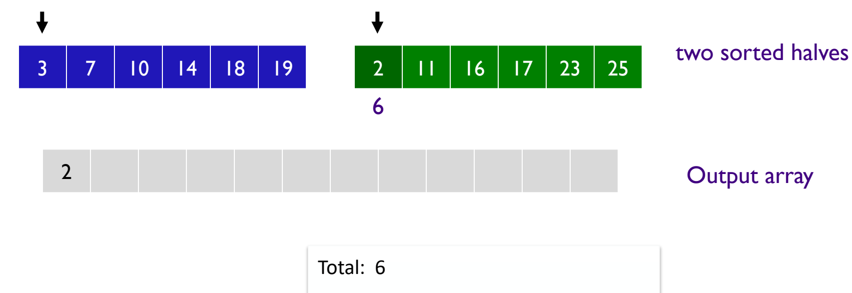

Merge and Count Step

- 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



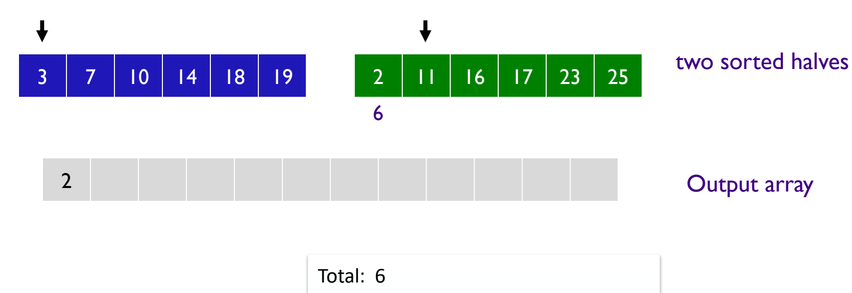
Merge and Count

- 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



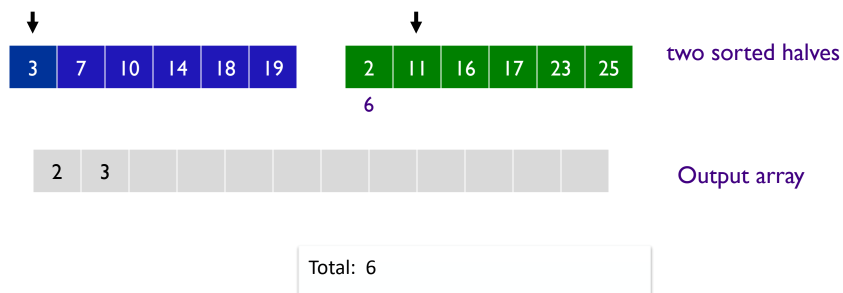
Merge and Count

- 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

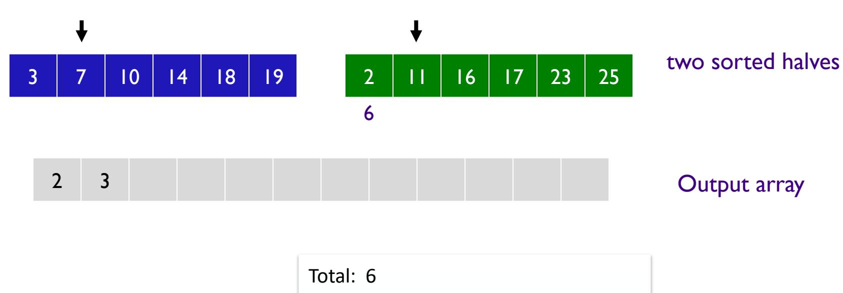
- Given two sorted halves, count number of inversions where a_i and a_j are in different halves
- Combine two sorted halves into sorted whole



March 11, 2019 CSCI211 - Sprenkle 19

Merge and Count

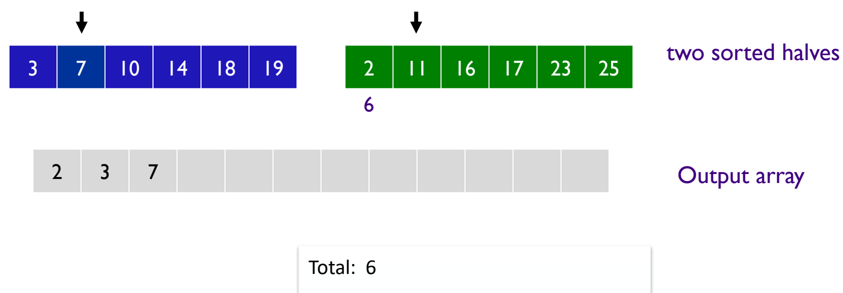
- Given two sorted halves, count number of inversions where a_i and a_j are in different halves
- Combine two sorted halves into sorted whole



March 11, 2019 CSCI211 - Sprenkle 20

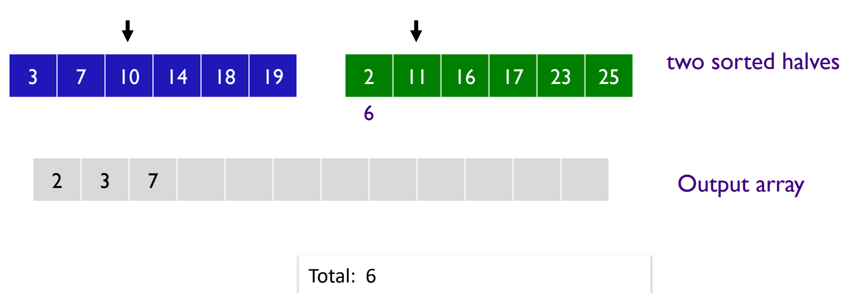
Merge and Count

- 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



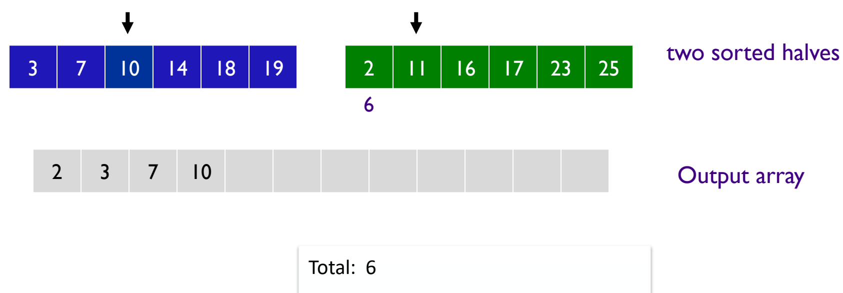
Merge and Count

- 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



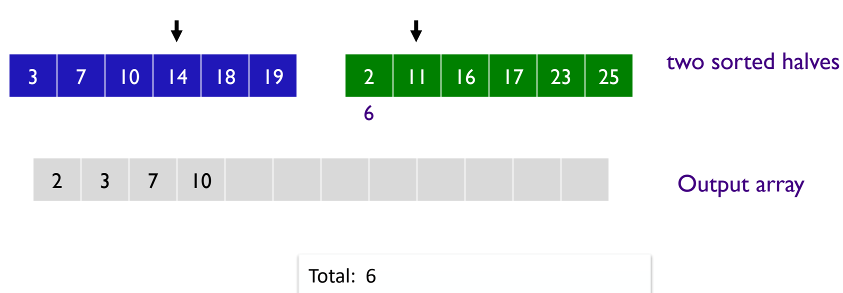
Merge and Count

- 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



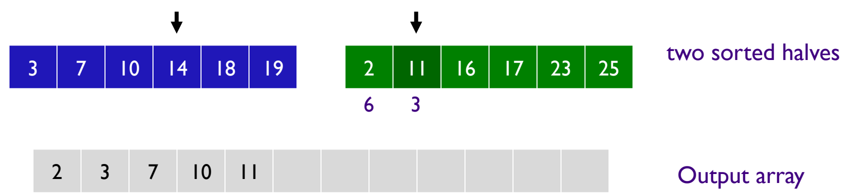
Merge and Count

- 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

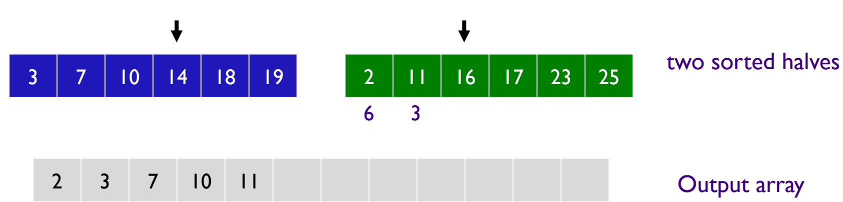
- Given two sorted halves,
  count number of inversions
  where a_i and a_j are in different halves
- Combine two sorted halves into sorted whole



Total: 6 + 3

Merge and Count

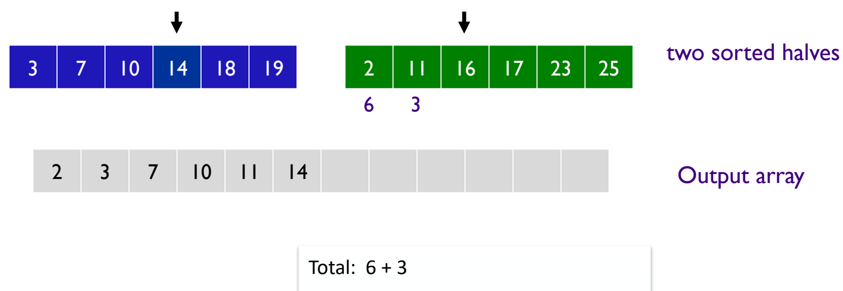
- Given two sorted halves,
  count number of inversions
  where a_i and a_j are in different halves
- Combine two sorted halves into sorted whole



Total: 6 + 3

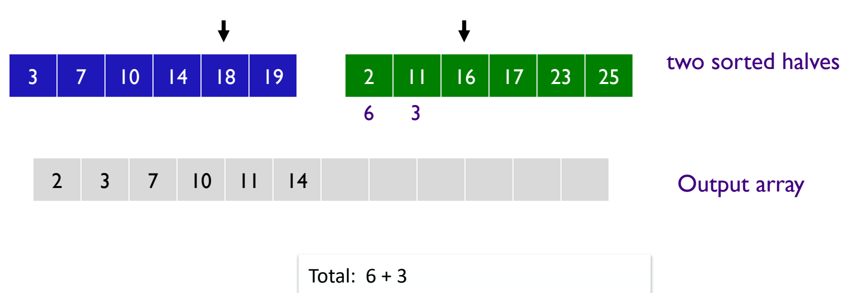
Merge and Count

- 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



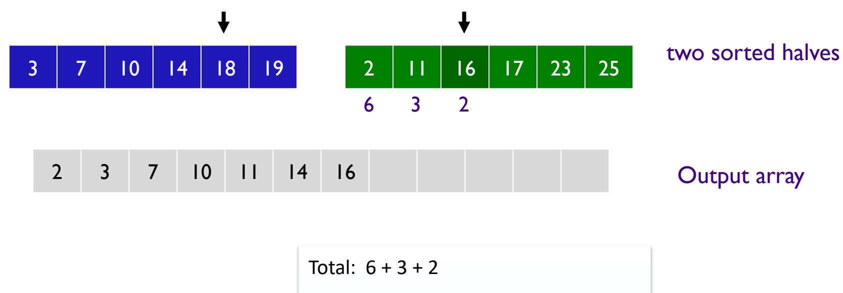
Merge and Count

- 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



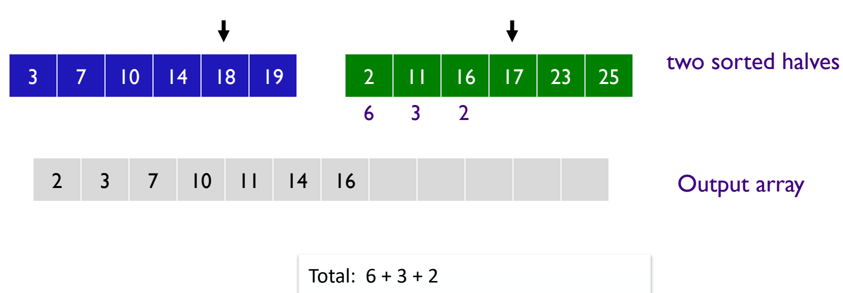
Merge and Count

- 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



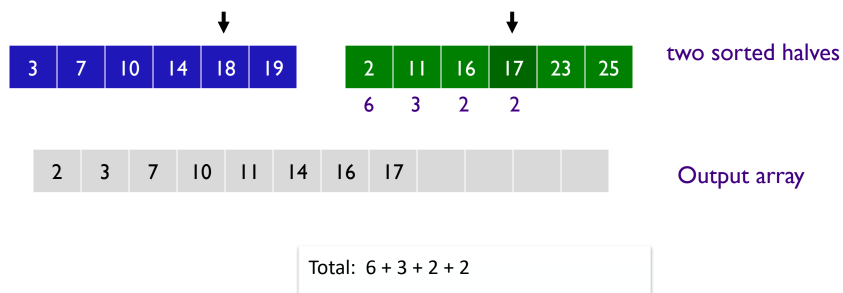
Merge and Count

- 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



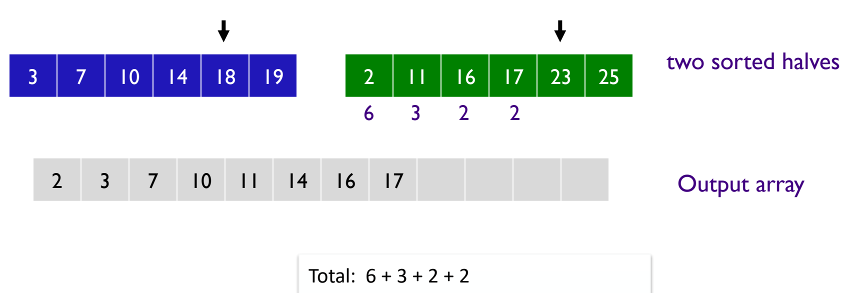
Merge and Count

- 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

- 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

- 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

two sorted halves

3	7	10	14	18	19

2	11	16	17	23	25
6	3	2	2		

Output array

2	3	7	10	11	14	16	17	18			

Total: 6 + 3 + 2 + 2

Merge and Count

- 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

two sorted halves

3	7	10	14	18	19

2	11	16	17	23	25
6	3	2	2		

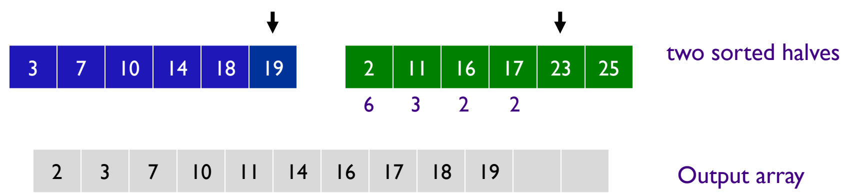
Output array

2	3	7	10	11	14	16	17	18			

Total: 6 + 3 + 2 + 2

Merge and Count

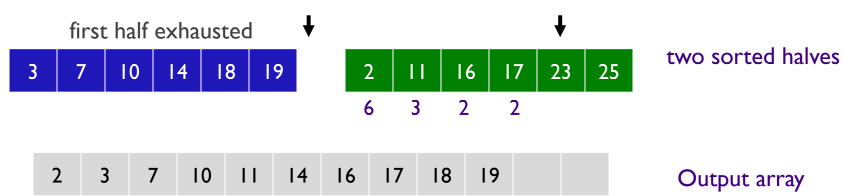
- Given two sorted halves, count number of inversions where a_i and a_j are in different halves
- Combine two sorted halves into sorted whole



Total: 6 + 3 + 2 + 2

Merge and Count

- 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 + 2 + 2

Merge and Count

- 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

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	

Output array

Total: 6 + 3 + 2 + 2 + 0

Merge and Count

- 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

3	7	10	14	18	19

2	11	16	17	23	25
6	3	2	2	0	

two sorted halves

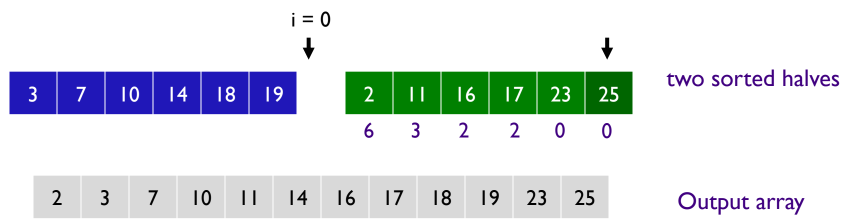
2	3	7	10	11	14	16	17	18	19	23	

Output array

Total: 6 + 3 + 2 + 2 + 0

Merge and Count

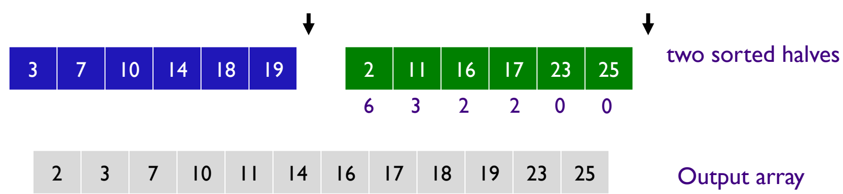
- Given two sorted halves,
  count number of inversions
  where a_i and a_j are in different halves
- Combine two sorted halves into sorted whole



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

Merge and Count

- 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 + 2 + 2 + 0 + 0 = 13

Counting Inversions: Implementation

```
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
   (iA, A) = Sort-and-Count(A)
   (iB, B) = Sort-and-Count(B)
   (i, L) = Merge-and-Count(A, B)

   total_inversions = iA + iB + i
   return total_inversions and the sorted list L
```


Counting Inversions: Implementation

```
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
   (iA, A) = Sort-and-Count(A)
   (iB, B) = Sort-and-Count(B)
   (i, L) = Merge-and-Count(A, B)

   total_inversions = iA + iB + i
   return total_inversions and the sorted list L
```

- Merge-and-Count
  - Pre-condition. A and B are sorted.
  - Post-condition. L is sorted.

Counting Inversions: Implementation

```
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
   (iA, A) = Sort-and-Count(A)
   (iB, B) = Sort-and-Count(B)
   (i, L) = Merge-and-Count(A, B)

   total_inversions = iA + iB + i
   return total_inversions and the sorted list L
```

Recurrence relation?
Runtime of algorithm?

- Merge-and-Count
  - Pre-condition. A and B are sorted.
  - Post-condition. L is sorted.

Analysis

Recurrence Relation:

$T(n) \leq 2T(n/2) + O(n)$

$\rightarrow T(n) \in O(n \log n)$

```
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
   (iA, A) = Sort-and-Count(A)     T(n/2)
   (iB, B) = Sort-and-Count(B)     T(n/2)
   (i, L) = Merge-and-Count(A, B)  O(n)

   total_inversions = iA + iB + i
   return total_inversions and the sorted list L
```

Looking Ahead

- Wiki: 4.8, 5.1-5.3
- PS 7 due Friday
- Exam 2 handed out on Friday
  - Greedy and D&C
  - Due following Friday