Objectives

- Divide and conquer algorithms
  - Recurrence relations
  - Counting inversions

Review

- What kind of problems are we solving?
- What is a recurrence relation?
- How can you compute D&C running times?
  - 2 ways to solve

Review: Another Recurrence Relation

- Instead of recursively solving 2 problems, solve ***q*** problems
  - Size of problems is still n/2
- Combining solutions is still O(n)
- Recurrence relation:
  - For some constant *c*,

    $T(n) \leq q\,T(n/2) + cn$ when $n > 2$

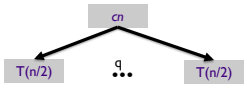
    $T(2) \leq c$

Intuition about running time?

Unrolling Recurrence, q > 2

- First level:

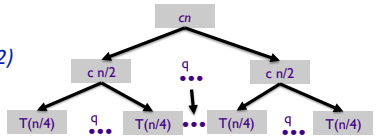
  $q\,T(n/2) + cn$



Unrolling Recurrence, q > 2

- Next level:

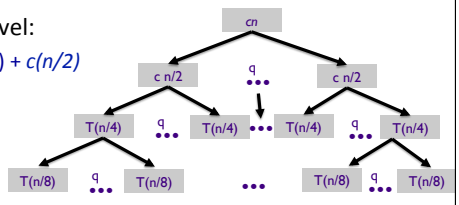
  $q\,T(n/4) + c(n/2)$



Unrolling Recurrence, q > 2

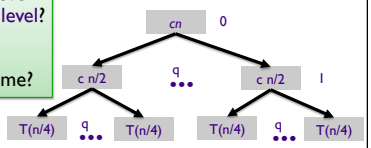
- Next level:

  $q\,T(n/4) + c(n/2)$



Unrolling Recurrence, q > 2

How much does each level cost, in terms of the level?
Number of levels?
What is the total run time?



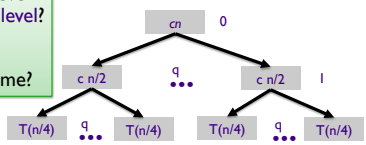
q^k problems at level k
Size: $n/2^k$

Number of levels: $\log_2 n$

Each level takes $q^k * c * (n/2^k) = (q/2)^i$ cn
→ Total work per level is *increasing* as level increases

Unrolling Recurrence, q > 2

How much does each level cost, in terms of the level?
Number of levels?
What is the total run time?



$T(n) \leq \Sigma_{j=0,\log n} (q/2)^i$ cn

Geometric series:
(constant ratio between successive terms)
Multiplying previous term by (q/2)

→ $O(n^{\log_2 q})$

Unrolling the Recurrence

- Generalize: What are the steps?

Summary

- Use recurrences to analyze the run time of divide and conquer algorithms
- Need to figure out
  - Number of sub problems
  - Size of sub problems
  - Number of times divided (number of levels)
  - Cost of merging problems

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 $a_i > a_j$

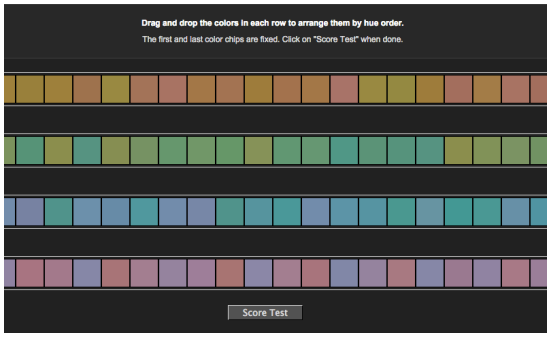
Movies

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

Inversions:
3-2, 4-2

http://www.xrite.com/online-color-test-challenge

Color Comparison Test



http://www.xrite.com/online-color-test-challenge

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

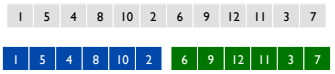
Assume number represents where item *should* be in the list, i.e., where it is in someone else's list

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

Should be at position 5

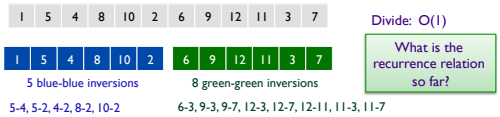
Counting Inversions: Divide-and-Conquer

- Divide: separate list into two pieces



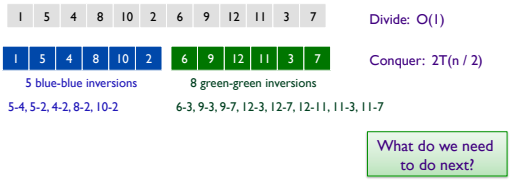
Counting Inversions: Divide-and-Conquer

- Divide: separate list into two pieces
- Conquer: recursively count inversions in each half



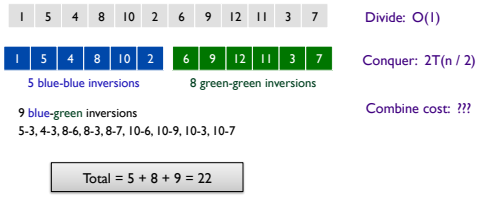
Counting Inversions: Divide-and-Conquer

- Divide: separate list into two pieces
- Conquer: recursively count inversions in each half



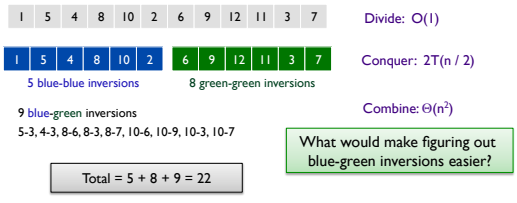
Counting Inversions: Divide-and-Conquer

- 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



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



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

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

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

Count: O(n)

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

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
```

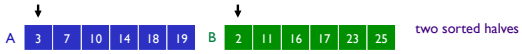

Merge and Count

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



Total:

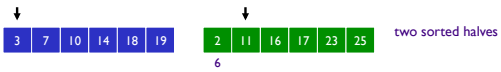
Merge and Count

- 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

Merge and Count

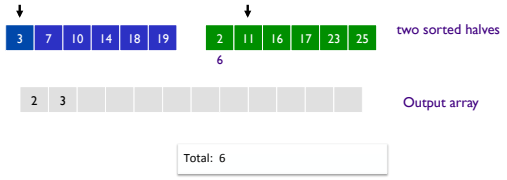
- Given two sorted halves, count number of inversions where a_i and a_j are in different halves
- Combine two sorted halves into sorted whole



Total: 6

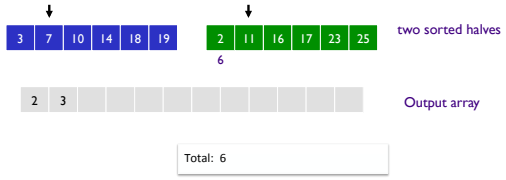
Merge and Count

- 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



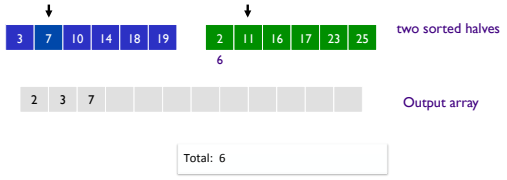
Merge and Count

- 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



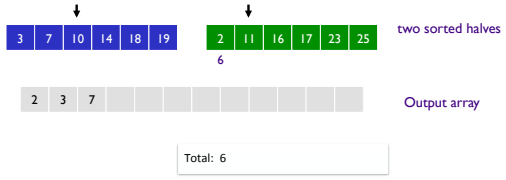
Merge and Count

- 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



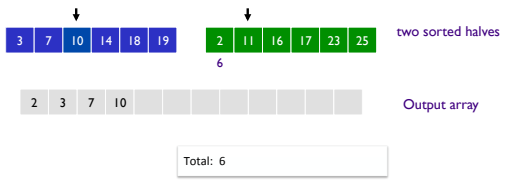
Merge and Count

- 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



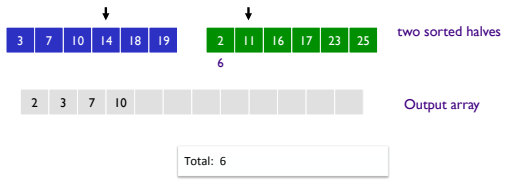
Merge and Count

- 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



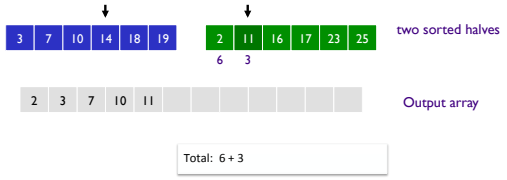
Merge and Count

- 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

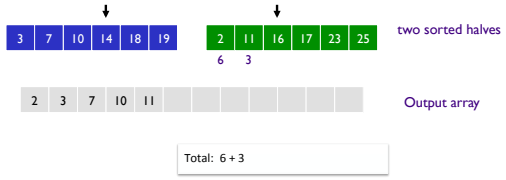
- Given two sorted halves, count number of inversions where a_i and a_j are in different halves
- Combine two sorted halves into sorted whole



Total: 6 + 3

Merge and Count

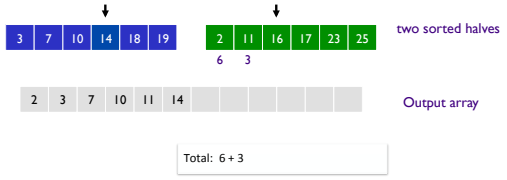
- Given two sorted halves, count number of inversions where a_i and a_j are in different halves
- Combine two sorted halves into sorted whole



Total: 6 + 3

Merge and Count

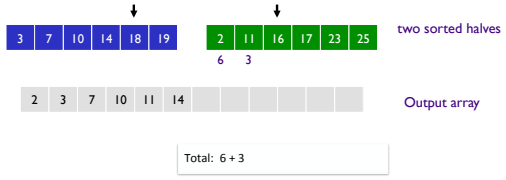
- Given two sorted halves, count number of inversions where a_i and a_j are in different halves
- Combine two sorted halves into sorted whole



Total: 6 + 3

Merge and Count

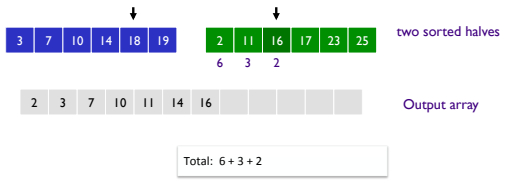
- Given two sorted halves, count number of inversions where a_i and a_j are in different halves
- Combine two sorted halves into sorted whole



Total: 6 + 3

Merge and Count

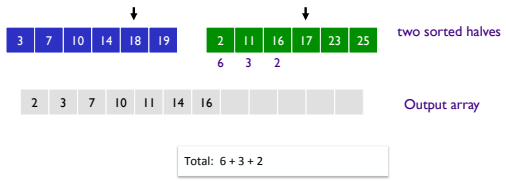
- Given two sorted halves, count number of inversions where a_i and a_j are in different halves
- Combine two sorted halves into sorted whole



Total: 6 + 3 + 2

Merge and Count

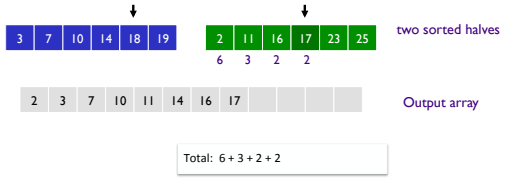
- Given two sorted halves, count number of inversions where a_i and a_j are in different halves
- Combine two sorted halves into sorted whole



Total: 6 + 3 + 2

Merge and Count

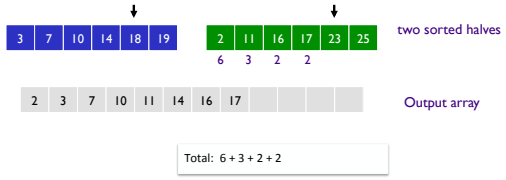
- Given two sorted halves, count number of inversions where a_i and a_j are in different halves
- Combine two sorted halves into sorted whole



Total: 6 + 3 + 2 + 2

Merge and Count

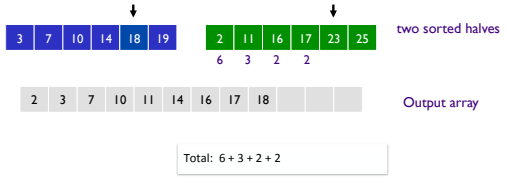
- Given two sorted halves, count number of inversions where a_i and a_j are in different halves
- Combine two sorted halves into sorted whole



Total: 6 + 3 + 2 + 2

Merge and Count

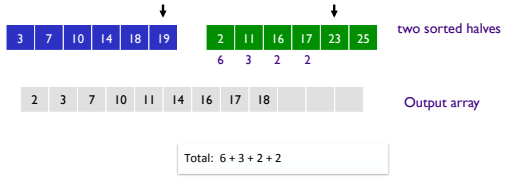
- Given two sorted halves, count number of inversions where a_i and a_j are in different halves
- Combine two sorted halves into sorted whole



Total: 6 + 3 + 2 + 2

Merge and Count

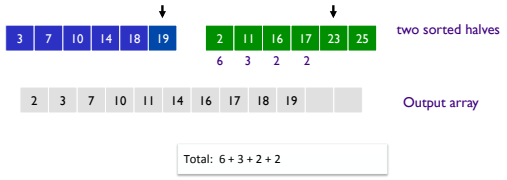
- Given two sorted halves, count number of inversions where a_i and a_j are in different halves
- Combine two sorted halves into sorted whole



Total: 6 + 3 + 2 + 2

Merge and Count

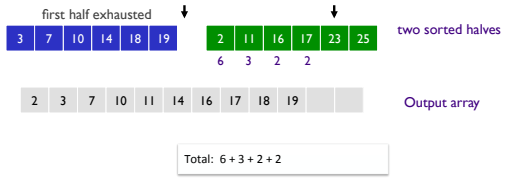
- Given two sorted halves, count number of inversions where a_i and a_j are in different halves
- Combine two sorted halves into sorted whole



Total: 6 + 3 + 2 + 2

Merge and Count

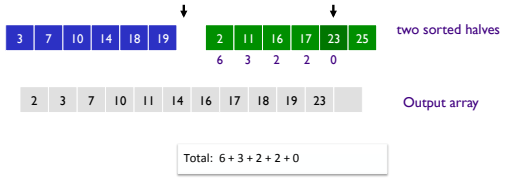
- Given two sorted halves, count number of inversions where a_i and a_j are in different halves
- Combine two sorted halves into sorted whole



Total: 6 + 3 + 2 + 2

Merge and Count

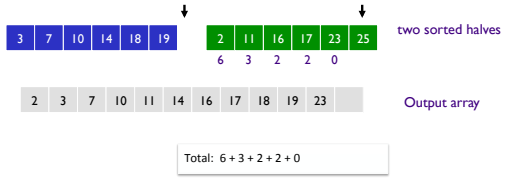
- Given two sorted halves, count number of inversions where a_i and a_j are in different halves
- Combine two sorted halves into sorted whole



Total: 6 + 3 + 2 + 2 + 0

Merge and Count

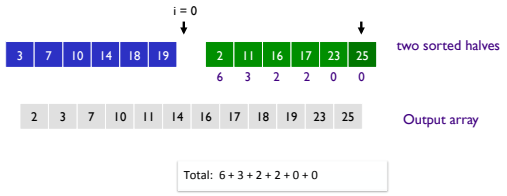
- Given two sorted halves, count number of inversions where a_i and a_j are in different halves
- Combine two sorted halves into sorted whole



Total: 6 + 3 + 2 + 2 + 0

Merge and Count

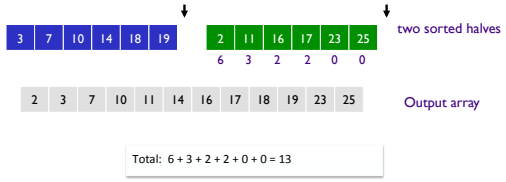
- Given two sorted halves, count number of inversions where a_i and a_j are in different halves
- Combine two sorted halves into sorted whole



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

Looking Ahead

- Problem Set 7 – due next Friday
- Wiki – 4.8, 5-5.2