Objectives

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

Review

- Describe the template for divide and conquer solutions
- How can you compute D&C running times?
  - Describe first step
  - 2 ways to solve
- What are you looking for when unrolling the recurrence?

Review: Divide-and-Conquer

> Divide et impera.
> Veni, vidi, vici.
> - *Julius Caesar*

- Divide-and-conquer process
  - Break up problem into several parts
  - Solve each part recursively
  - Combine solutions to sub-problems into overall solution
- Define a **recurrence relation** that describes the running time

Review: Recurrence Relations

- Use recurrences to analyze/determine the run time of divide and conquer algorithms
  - Number of sub problems
  - Size of sub problems
  - Number of times divided (number of levels)
  - Cost of merging problems
- How to solve
  - Unrolling
  - Substitution

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 $a_i > a_j$

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

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

What was our solution so far?

Counting Inversions: Divide-and-Conquer

- Divide: separate list into two pieces

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

Divide: O(1)

Counting Inversions: Divide-and-Conquer

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

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

Divide: O(1)

5 blue-blue inversions

5-4, 5-2, 4-2, 8-2, 10-2

8 green-green inversions

6-3, 9-3, 9-7, 12-3, 12-7, 12-11, 11-3, 11-7

What is recurrence relation so far?

Counting Inversions: Divide-and-Conquer

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

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

Divide: O(1)

Conquer: 2T(n / 2)

5 blue-blue inversions

5-4, 5-2, 4-2, 8-2, 10-2

8 green-green inversions

6-3, 9-3, 9-7, 12-3, 12-7, 12-11, 11-3, 11-7

What do we need to do next?

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

Divide: O(1)

Conquer: 2T(n / 2)

5 blue-blue inversions

8 green-green inversions

Combine cost: ???

9 blue-green inversions

5-3, 4-3, 8-6, 8-3, 8-7, 10-6, 10-9, 10-3, 10-7

Total = 5 + 8 + 9 = 22

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

Divide: O(1)

Conquer: 2T(n / 2)

5 blue-blue inversions

8 green-green inversions

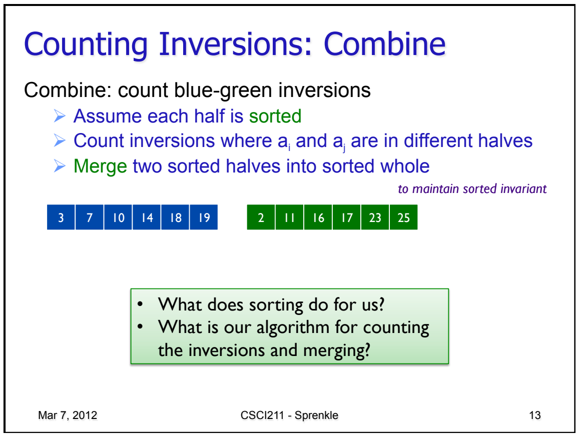
Combine: seems like $\Theta(n^2)$

9 blue-green inversions

5-3, 4-3, 8-6, 8-3, 8-7, 10-6, 10-9, 10-3, 10-7

Total = 5 + 8 + 9 = 22

What would make figuring out blue-green inversions easier?



Counting Inversions: Combine

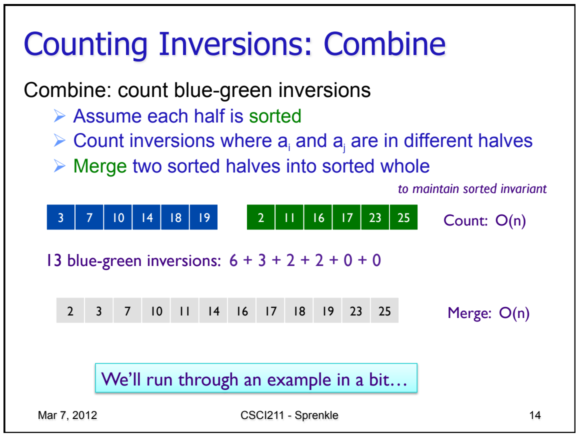
Combine: count blue-green inversions

- 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?



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

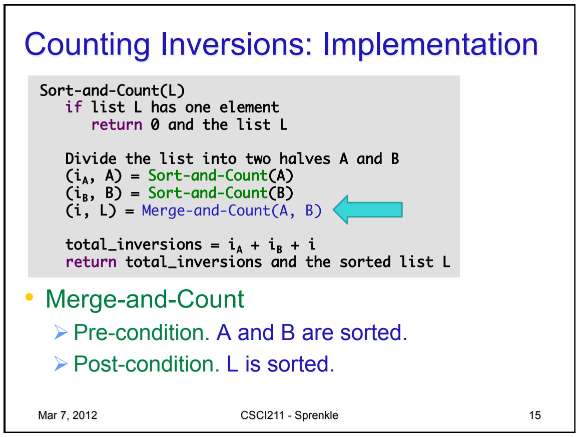
to maintain sorted invariant

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)

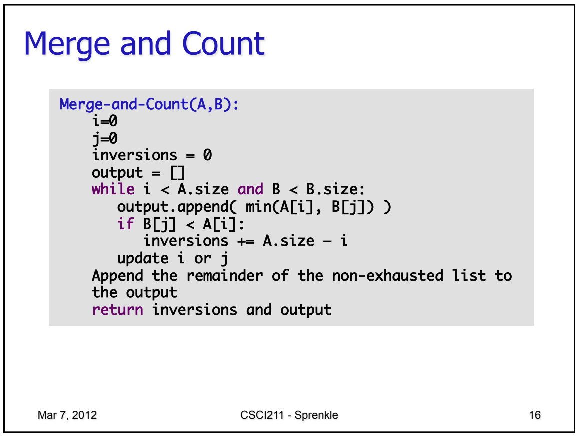
We'll run through an example in a bit…



Counting Inversions: Implementation

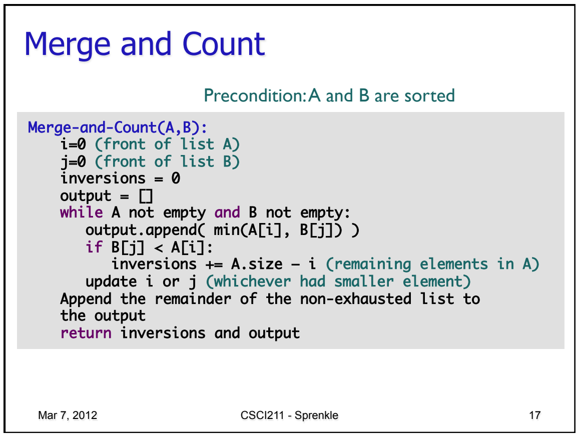
```
Sort-and-Count(L)
   if list L has one element
      return 0 and the list L

   Divide the list into two halves A and B
   (i_A, A) = Sort-and-Count(A)
   (i_B, B) = Sort-and-Count(B)
   (i, L) = Merge-and-Count(A, B)

   total_inversions = i_A + i_B + i
   return total_inversions and the sorted list L
```

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



Merge and Count

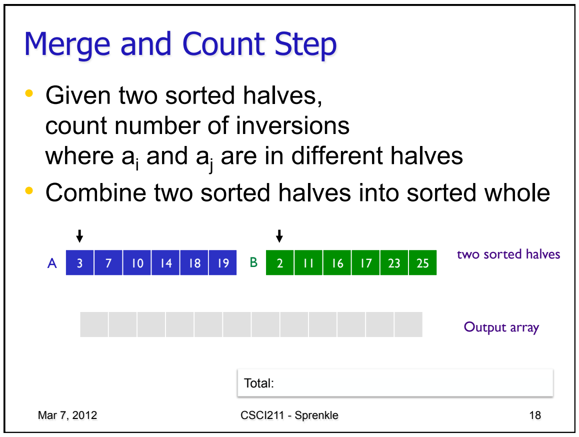
```
Merge-and-Count(A,B):
    i=0
    j=0
    inversions = 0
    output = []
    while i < A.size and B < B.size:
       output.append( min(A[i], B[j]) )
       if B[j] < A[i]:
          inversions += A.size – i
       update i or j
    Append the remainder of the non-exhausted list to
    the output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



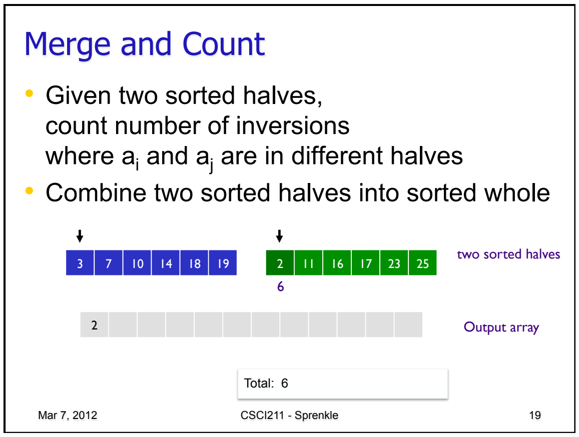
Merge and Count Step

- 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

A 3 | 7 | 10 | 14 | 18 | 19   B 2 | 11 | 16 | 17 | 23 | 25   two sorted halves

Output array

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

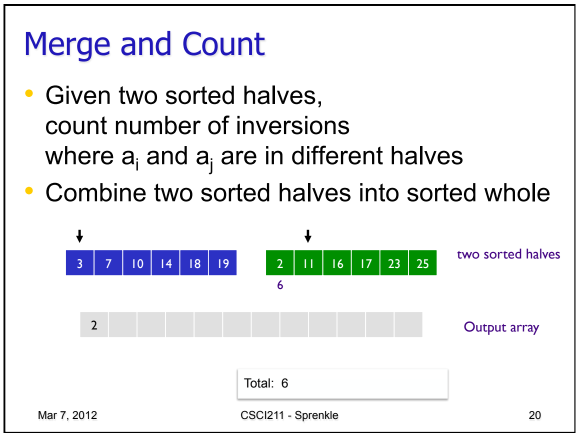
3 7 10 14 18 19   2 11 16 17 23 25   two sorted halves

6

2   Output array

Total: 6

Mar 7, 2012   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

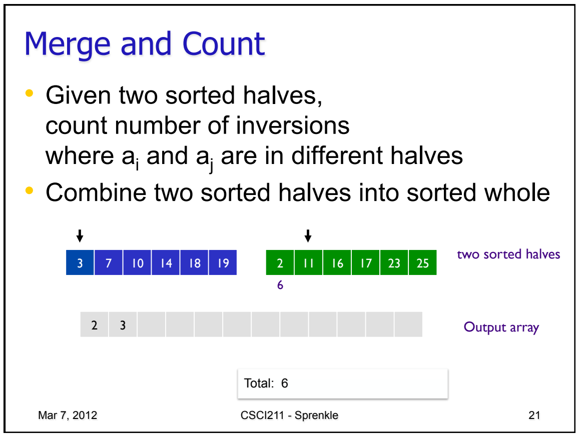
3 7 10 14 18 19   2 11 16 17 23 25   two sorted halves

6

2   Output array

Total: 6

Mar 7, 2012   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

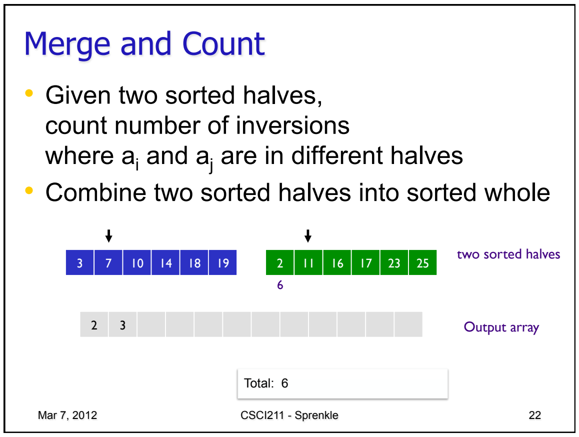
3 7 10 14 18 19   2 11 16 17 23 25   two sorted halves

6

2 3   Output array

Total: 6

Mar 7, 2012   CSCI211 - Sprenkle   21



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

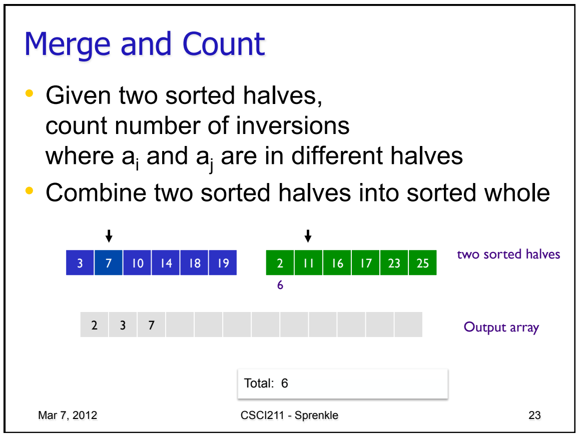
3 7 10 14 18 19   2 11 16 17 23 25   two sorted halves

6

2 3   Output array

Total: 6

Mar 7, 2012   CSCI211 - Sprenkle   22



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

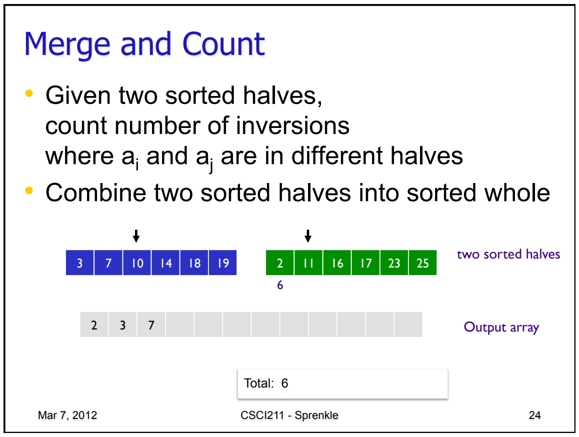
3 7 10 14 18 19   2 11 16 17 23 25   two sorted halves

6

2 3 7   Output array

Total: 6

Mar 7, 2012   CSCI211 - Sprenkle   23



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

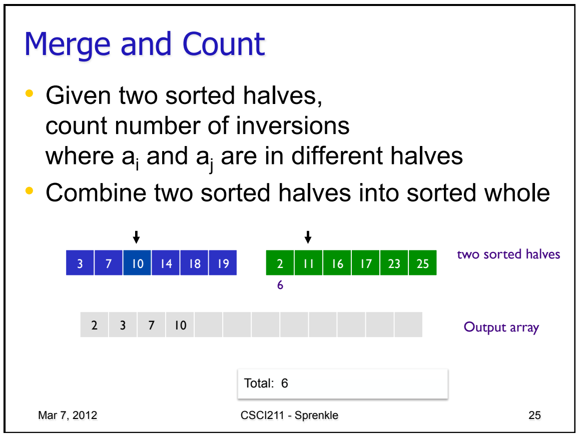
3 7 10 14 18 19   2 11 16 17 23 25   two sorted halves

6

2 3 7   Output array

Total: 6

Mar 7, 2012   CSCI211 - Sprenkle   24



Merge and Count

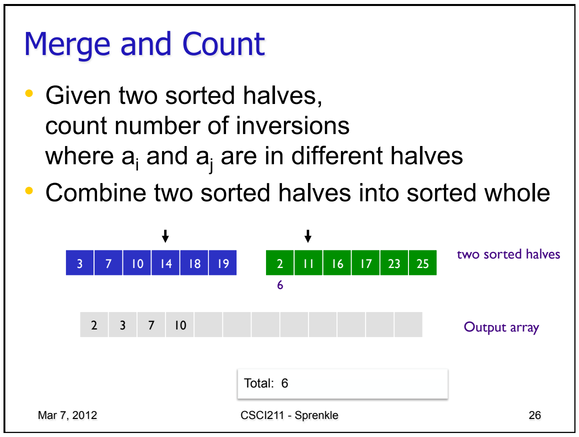
- Given two sorted halves,
  count number of inversions
  where a_i and a_j are in different halves
- Combine two sorted halves into sorted whole

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

2	3	7	10									Output array

Total: 6

Mar 7, 2012 CSCI211 - Sprenkle 25



Merge and Count

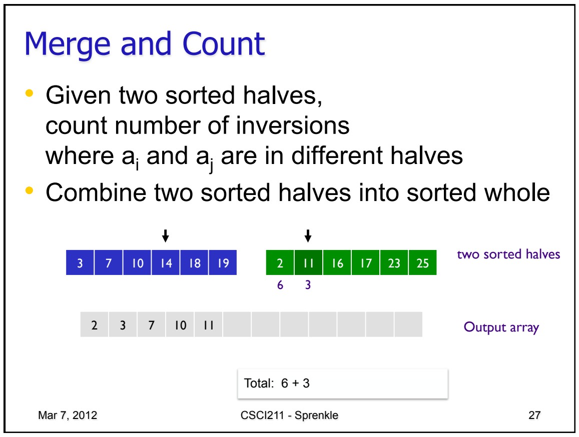
- Given two sorted halves,
  count number of inversions
  where a_i and a_j are in different halves
- Combine two sorted halves into sorted whole

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

2	3	7	10									Output array

Total: 6

Mar 7, 2012 CSCI211 - Sprenkle 26



Merge and Count

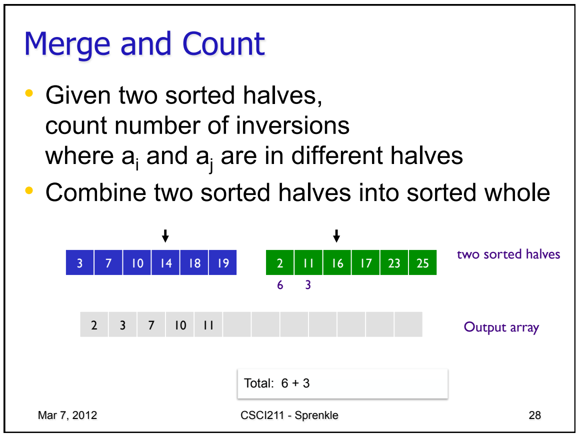
- Given two sorted halves,
  count number of inversions
  where a_i and a_j are in different halves
- Combine two sorted halves into sorted whole

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

2	3	7	10	11								Output array

Total: 6 + 3

Mar 7, 2012 CSCI211 - Sprenkle 27



Merge and Count

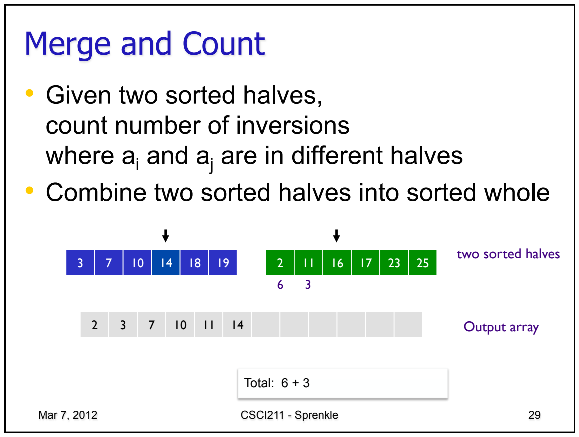
- Given two sorted halves,
  count number of inversions
  where a_i and a_j are in different halves
- Combine two sorted halves into sorted whole

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

2	3	7	10	11								Output array

Total: 6 + 3

Mar 7, 2012 CSCI211 - Sprenkle 28



Merge and Count

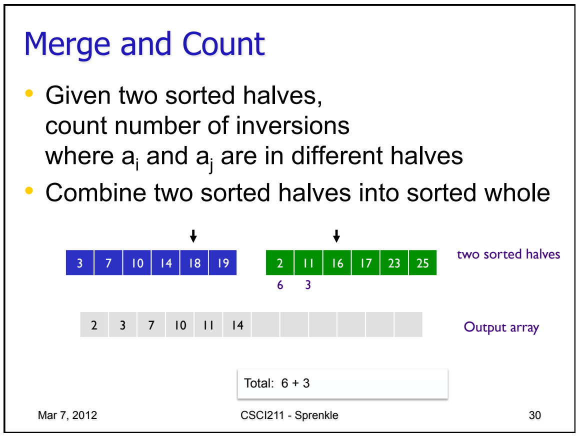
- Given two sorted halves,
  count number of inversions
  where a_i and a_j are in different halves
- Combine two sorted halves into sorted whole

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

2	3	7	10	11	14							Output array

Total: 6 + 3

Mar 7, 2012 CSCI211 - Sprenkle 29



Merge and Count

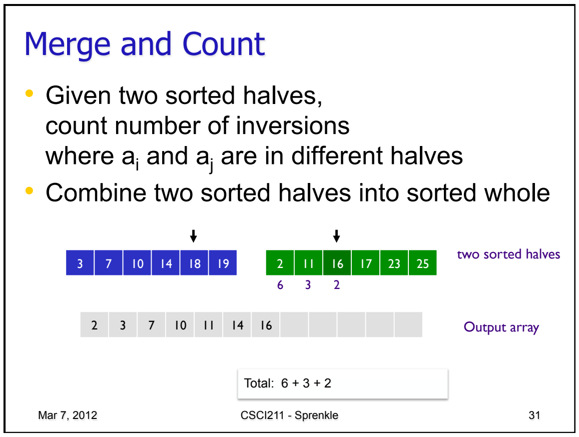
- Given two sorted halves,
  count number of inversions
  where a_i and a_j are in different halves
- Combine two sorted halves into sorted whole

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

2	3	7	10	11	14							Output array

Total: 6 + 3

Mar 7, 2012 CSCI211 - Sprenkle 30



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

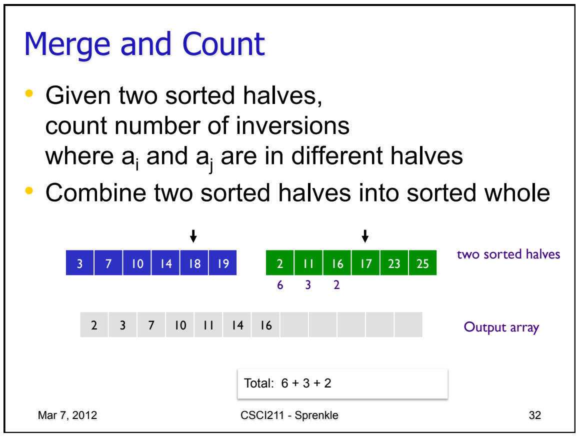
3 7 10 14 18 19 | 2 11 16 17 23 25 two sorted halves

6 3 2

2 3 7 10 11 14 16 Output array

Total: 6 + 3 + 2

Mar 7, 2012 CSCI211 - Sprenkle 31



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

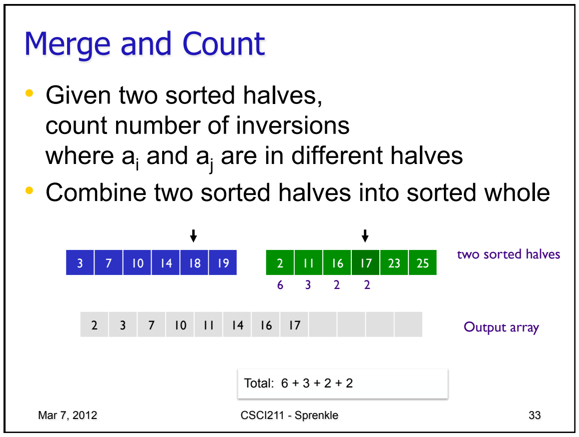
3 7 10 14 18 19 | 2 11 16 17 23 25 two sorted halves

6 3 2

2 3 7 10 11 14 16 Output array

Total: 6 + 3 + 2

Mar 7, 2012 CSCI211 - Sprenkle 32



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

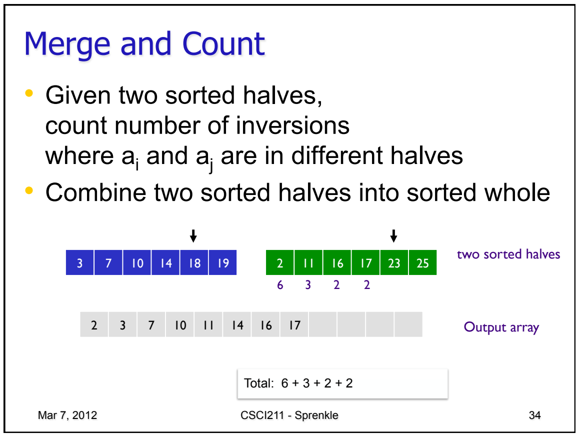
3 7 10 14 18 19 | 2 11 16 17 23 25 two sorted halves

6 3 2 2

2 3 7 10 11 14 16 17 Output array

Total: 6 + 3 + 2 + 2

Mar 7, 2012 CSCI211 - Sprenkle 33



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

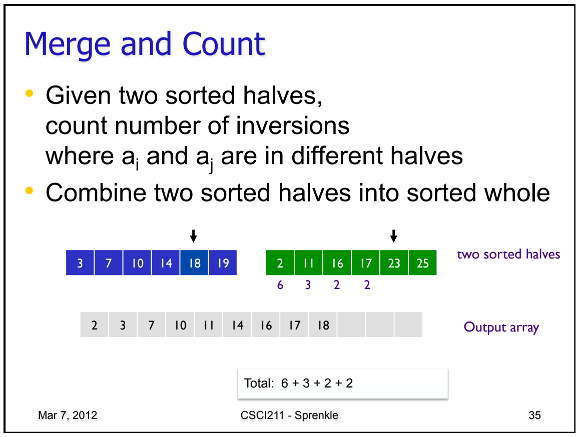
3 7 10 14 18 19 | 2 11 16 17 23 25 two sorted halves

6 3 2 2

2 3 7 10 11 14 16 17 Output array

Total: 6 + 3 + 2 + 2

Mar 7, 2012 CSCI211 - Sprenkle 34



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

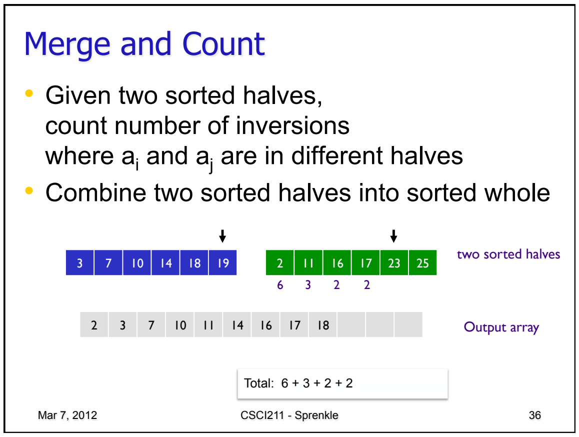
3 7 10 14 18 19 | 2 11 16 17 23 25 two sorted halves

6 3 2 2

2 3 7 10 11 14 16 17 18 Output array

Total: 6 + 3 + 2 + 2

Mar 7, 2012 CSCI211 - Sprenkle 35



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 two sorted halves

6 3 2 2

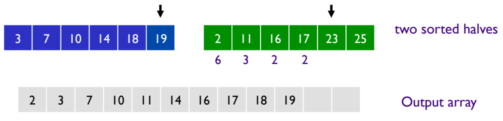
2 3 7 10 11 14 16 17 18 Output array

Total: 6 + 3 + 2 + 2

Mar 7, 2012 CSCI211 - Sprenkle 36

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

two sorted halves

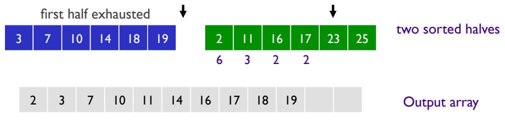
2 3 7 10 11 14 16 17 18 19

Output array

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



first half exhausted

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

6 3 2 2

two sorted halves

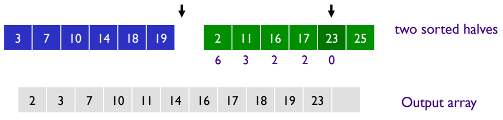
2 3 7 10 11 14 16 17 18 19

Output array

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

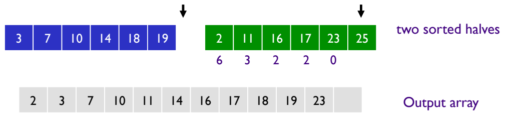
2 3 7 10 11 14 16 17 18 19 23

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

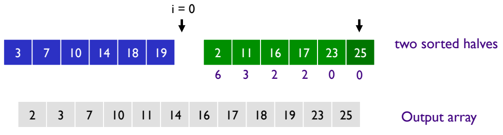
2 3 7 10 11 14 16 17 18 19 23

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



i = 0

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

6 3 2 2 0 0

two sorted halves

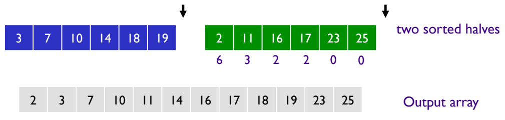
2 3 7 10 11 14 16 17 18 19 23 25

Output array

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



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

Output array

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
   ($i_A$, A) = Sort-and-Count(A)
   ($i_B$, B) = Sort-and-Count(B)
   (i, L) = Merge-and-Count(A, B)

   total_inversions = $i_A$ + $i_B$ + 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:

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

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

```
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
   ($i_A$, A) = Sort-and-Count(A)    T(n/2)
   ($i_B$, B) = Sort-and-Count(B)    T(n/2)
   (i, L) = Merge-and-Count(A, B) O(n)

   return i = $i_A$ + $i_B$ + i and the sorted list L
```

CLOSEST PAIR OF POINTS

Computational Geometry

- Algorithms and data structures for geometrical objects
  - Points, line segments, polygons, etc.
  - Common motivator: large data sets → efficiency
- Some Applications
  - Graphics
  - Robotics
    - motion planning and visibility problems
  - Geographic information systems (GIS)
    - geometrical location and search, route planning

Closest Pair of Points

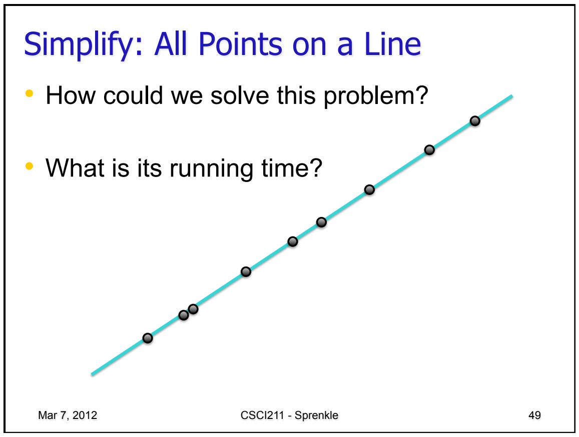
- Closest pair. Given *n* points in the plane, find a pair with smallest Euclidean distance between them.
  - Special case of nearest neighbor, Euclidean MST, Voronoi

fast closest pair inspired
fast algorithms for these problems

- Brute force?

Closest Pair of Points

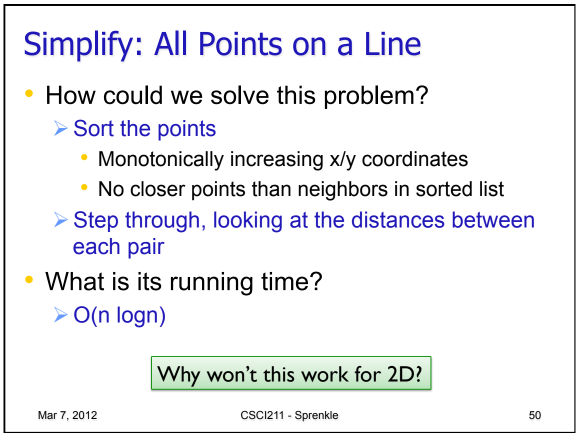
- Closest pair. Given *n* points in the plane, find a pair with smallest Euclidean distance between them.
  - Special case of nearest neighbor, Euclidean MST, Voronoi.
- Brute force. Check all pairs of points p and q with $\Theta(n^2)$ comparisons



Simplify: All Points on a Line

- How could we solve this problem?
- What is its running time?

Mar 7, 2012 CSCI211 - Sprenkle 49

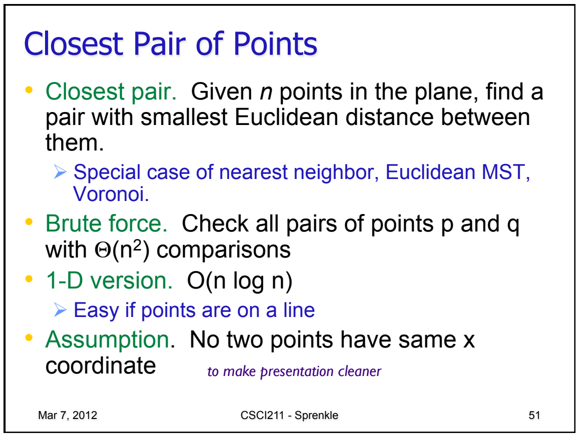


Simplify: All Points on a Line

- How could we solve this problem?
  - Sort the points
    - Monotonically increasing x/y coordinates
    - No closer points than neighbors in sorted list
  - Step through, looking at the distances between each pair
- What is its running time?
  - O(n logn)

Why won't this work for 2D?

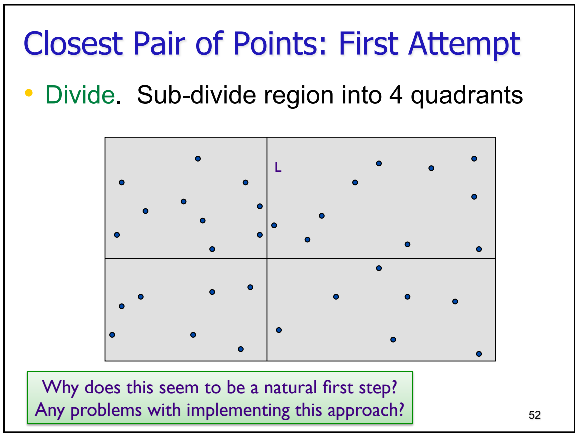
Mar 7, 2012 CSCI211 - Sprenkle 50



Closest Pair of Points

- Closest pair. Given *n* points in the plane, find a pair with smallest Euclidean distance between them.
  - Special case of nearest neighbor, Euclidean MST, Voronoi.
- Brute force. Check all pairs of points p and q with $\Theta(n^2)$ comparisons
- 1-D version. O(n log n)
  - Easy if points are on a line
- Assumption. No two points have same x coordinate *to make presentation cleaner*

Mar 7, 2012 CSCI211 - Sprenkle 51

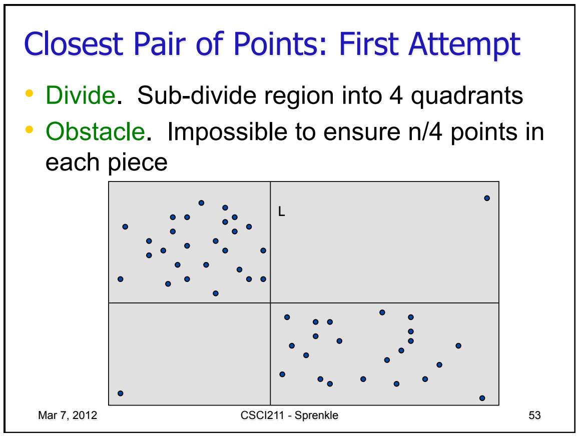


Closest Pair of Points: First Attempt

- Divide. Sub-divide region into 4 quadrants

Why does this seem to be a natural first step?
Any problems with implementing this approach?

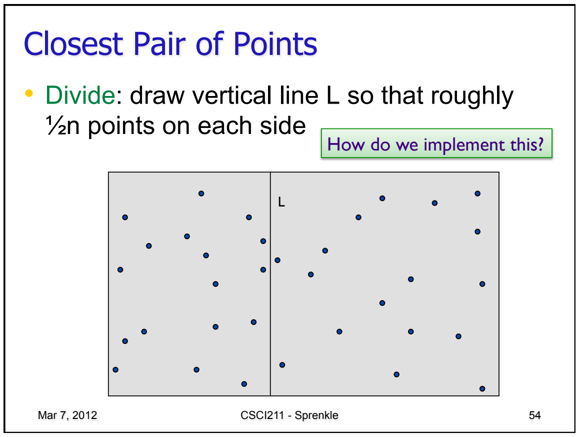
52



Closest Pair of Points: First Attempt

- Divide. Sub-divide region into 4 quadrants
- Obstacle. Impossible to ensure n/4 points in each piece

Mar 7, 2012 CSCI211 - Sprenkle 53

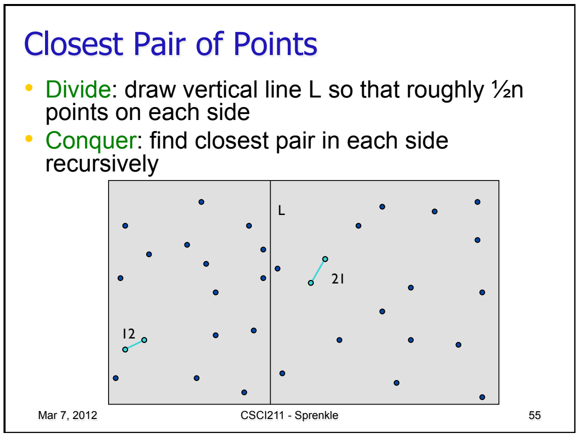


Closest Pair of Points

- Divide: draw vertical line L so that roughly ½n points on each side

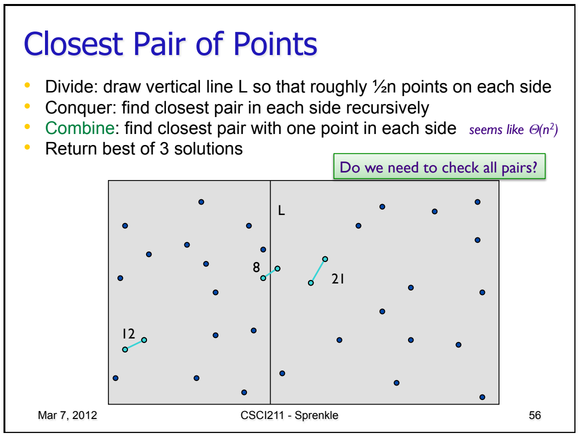
How do we implement this?

Mar 7, 2012 CSCI211 - Sprenkle 54



Closest Pair of Points

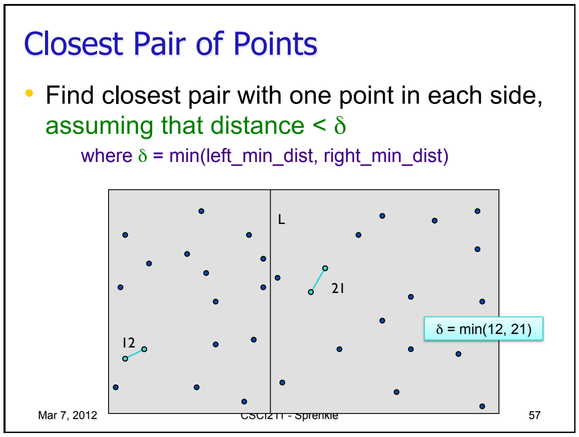
- Divide: draw vertical line L so that roughly ½n points on each side
- Conquer: find closest pair in each side recursively



Closest Pair of Points

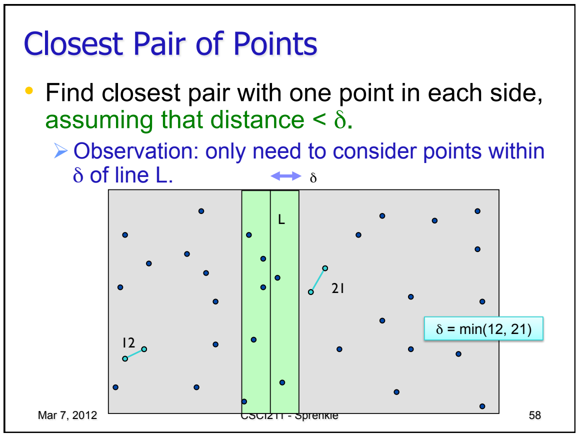
- Divide: draw vertical line L so that roughly ½n points on each side
- Conquer: find closest pair in each side recursively
- Combine: find closest pair with one point in each side *seems like $\Theta(n^2)$*
- Return best of 3 solutions

Do we need to check all pairs?



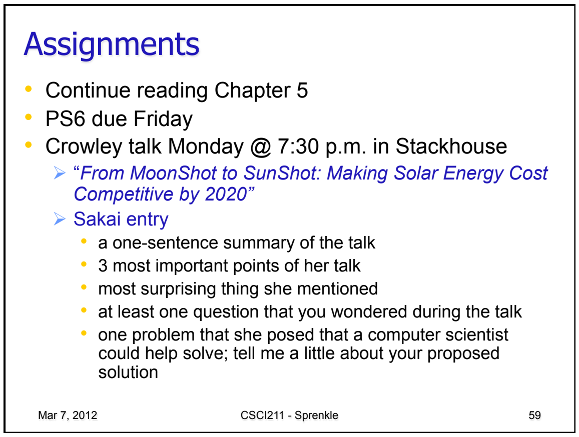
Closest Pair of Points

- Find closest pair with one point in each side, assuming that distance < δ
  - where δ = min(left_min_dist, right_min_dist)



Closest Pair of Points

- Find closest pair with one point in each side, assuming that distance < δ.
  - Observation: only need to consider points within δ of line L.



Assignments

- Continue reading Chapter 5
- PS6 due Friday
- Crowley talk Monday @ 7:30 p.m. in Stackhouse
  - *"From MoonShot to SunShot: Making Solar Energy Cost Competitive by 2020"*
  - Sakai entry
    - a one-sentence summary of the talk
    - 3 most important points of her talk
    - most surprising thing she mentioned
    - at least one question that you wondered during the talk
    - one problem that she posed that a computer scientist could help solve; tell me a little about your proposed solution