



CSE 202

Divide-and-conquer algorithms

Fan Chung Graham

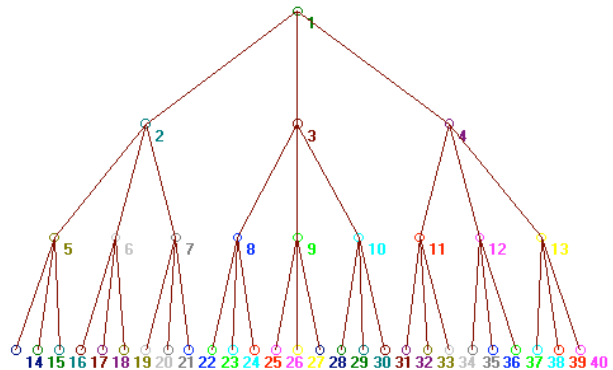
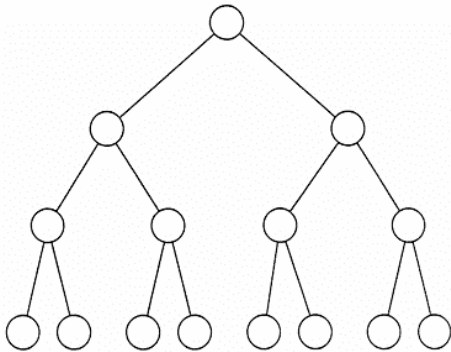
UC San Diego

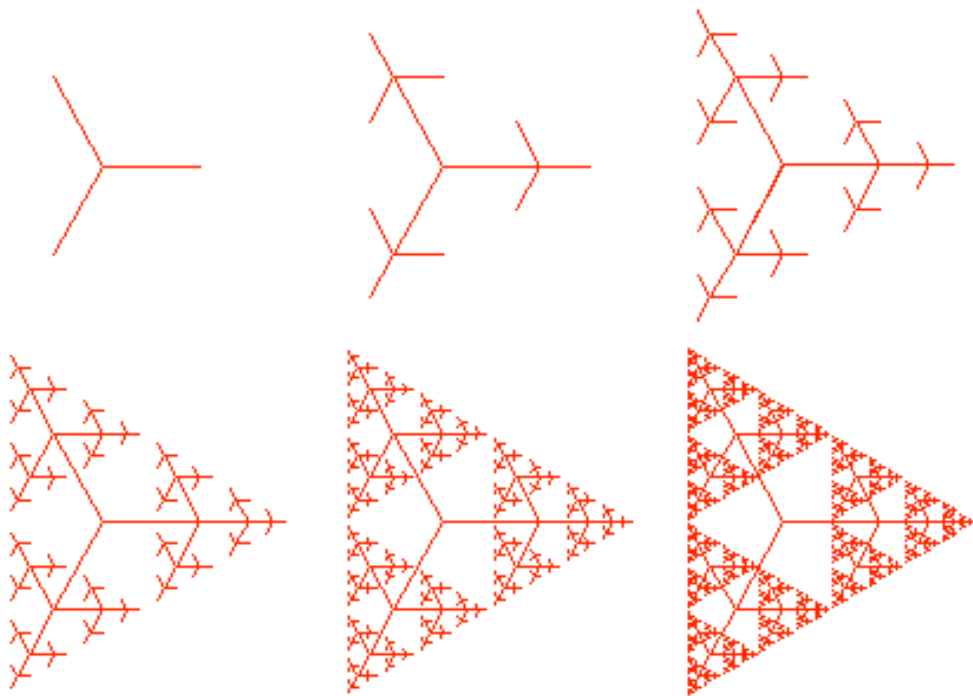
An induced subgraph of the collaboration graph (with Erdos number at most 2).

Made by Fan Chung Graham and Lincoln Lu in 2002.

A useful fact about trees

Any tree on n vertices contains a vertex v whose removal separates the remaining graph into two parts, one of which is of sizes at most $n/2$ and the other is at most $2n/3$.

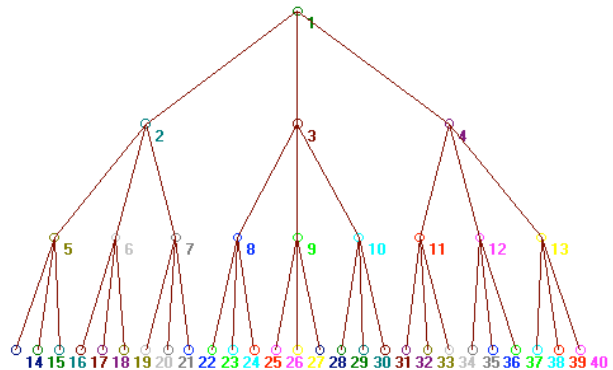
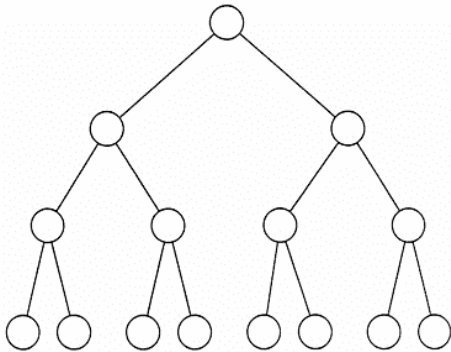




Ternary trees

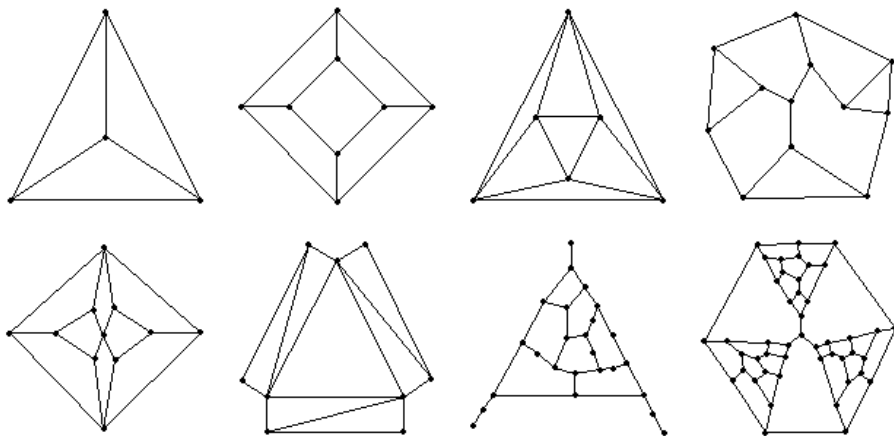
A useful fact about trees

Any tree on n vertices contains a vertex v whose removal separates the remaining graph into two parts, one of which is of sizes at most $n/2$ and the other is at most $2n/3$.

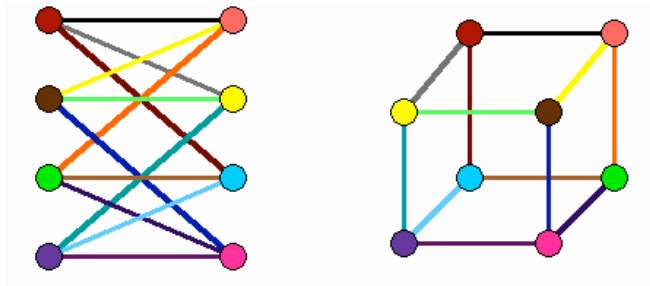


Try to write a proof for this!

A planar graph is a graph that can be drawn in the plane without crossings.

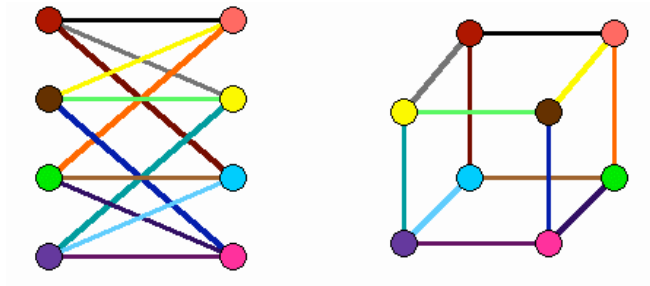


A planar graph is a graph that can be drawn in the plane without any crossing.

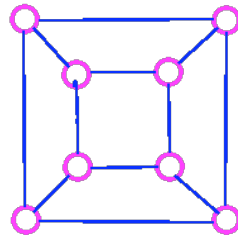
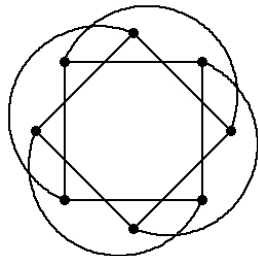


Are these planar graphs?

A planar graph is a graph that can be drawn in the plane without any crossing.



Are these planar graphs?



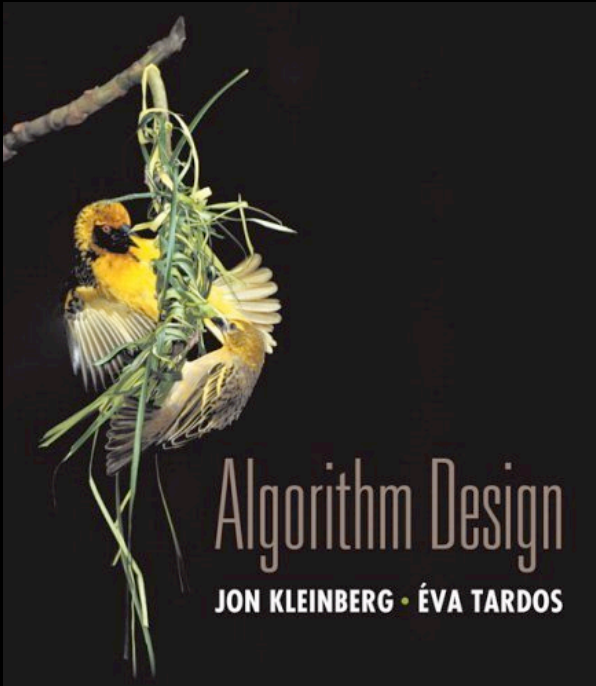
A useful fact about planar graphs

Any planar graph on n vertices contains \sqrt{n} vertices whose removal separates the remaining graph into two parts, one of which is of size at most $n/2$ and the other is at most $2n/3$.

Tarjan and Lipton, 1977

Chapter 5

Divide and Conquer



Slides by Kevin Wayne.
Copyright © 2005 Pearson-Addison Wesley.
All rights reserved.

Divide-and-Conquer

Divide-and-conquer.

- Break up problem into several parts.
- Solve each part recursively.
- Combine solutions to sub-problems into overall solution.

Most common usage.

- Break up problem of size n into **two** equal parts of size $\frac{1}{2}n$.
- Solve two parts recursively.
- Combine two solutions into overall solution in **linear time**.

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

5.1 Mergesort

Sorting

Sorting.

Given n elements, rearrange in ascending order.

3, 6, 5, 2, 1, 4

B, U, S, H

1, 2, 3, 4, 5, 6

B, H, S, U

Obvious sorting applications.

- List files in a directory.

- Organize an MP3 library.

- List names in a phone book.

- Display Google PageRank results.

Sorting

Obvious sorting applications.

- List files in a directory.

- Organize an MP3 library.

- List names in a phone book.

- Display Google PageRank results.

Problems become easier once sorted.

- Find the median.

- Binary search in a database.

- Identify statistical outliers.

- Find duplicates in a mailing list.

Sorting

Non-obvious sorting applications.

Data compression.

Computer graphics.

Interval scheduling.

Computational biology.

Minimum spanning tree.

Supply chain management.

Simulate a system of particles.

Book recommendations on Amazon.

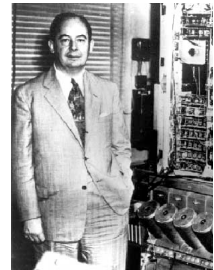
Load balancing on a parallel computer.

...

Mergesort

Mergesort.

- Divide array into two halves.
- Recursively sort each half.
- Merge two halves to make sorted whole.



Jon von Neumann (1945)

A L G O R I T H M S

A L G O R

I T H M S

divide $O(1)$

A G L O R

H I M S T

sort $2T(n/2)$

A G H I L M O R S T

merge $O(n)$

Merging

Merging. Combine two pre-sorted lists into a sorted whole.

How to merge efficiently?



- Linear number of comparisons.
- Use temporary array.



Challenge for the bored. In-place merge. [Kronrud, 1969]

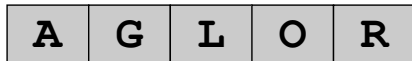
↑
using only a constant amount of extra storage

Merging

Merge.

- Keep track of smallest element in each sorted half.
- Insert smallest of two elements into auxiliary array.
- Repeat until done.

smallest



smallest

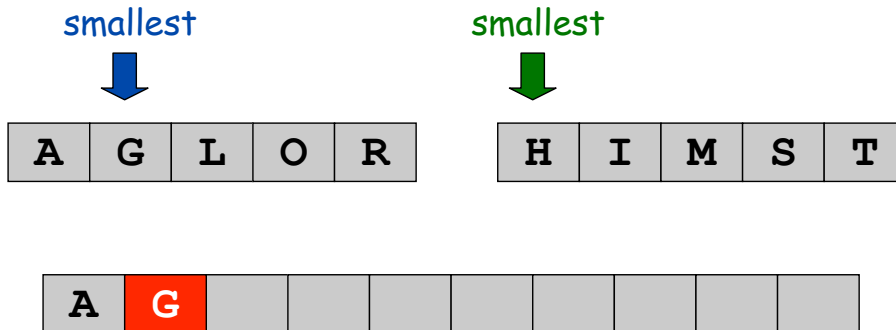


auxiliary array

Merging

Merge.

- Keep track of smallest element in each sorted half.
- Insert smallest of two elements into auxiliary array.
- Repeat until done.

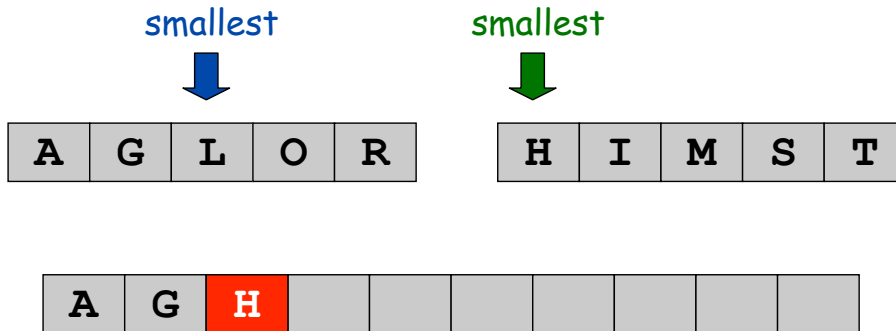


auxiliary array

Merging

Merge.

- Keep track of smallest element in each sorted half.
- Insert smallest of two elements into auxiliary array.
- Repeat until done.

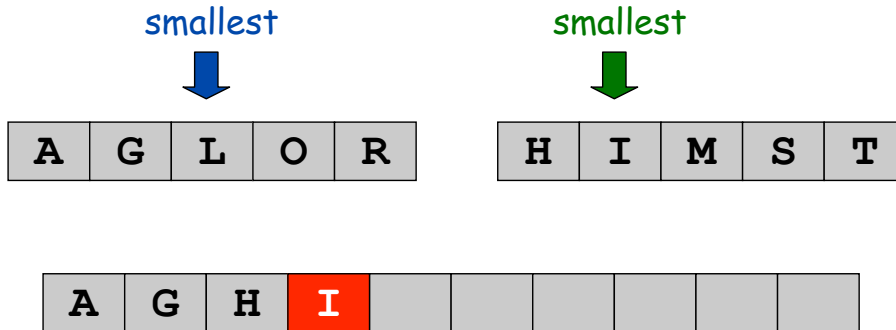


auxiliary array

Merging

Merge.

- Keep track of smallest element in each sorted half.
- Insert smallest of two elements into auxiliary array.
- Repeat until done.

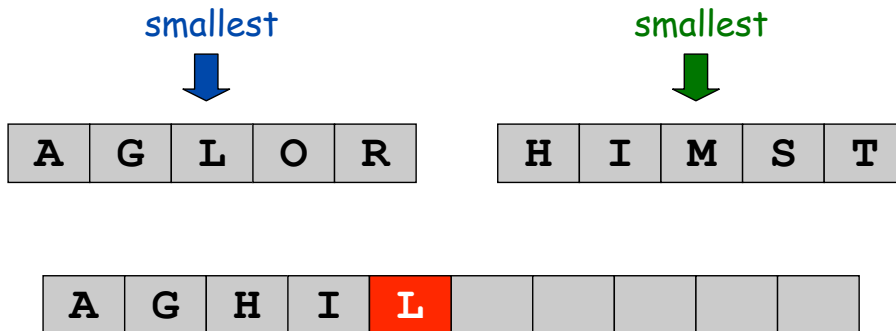


auxiliary array

Merging

Merge.

- Keep track of smallest element in each sorted half.
- Insert smallest of two elements into auxiliary array.
- Repeat until done.

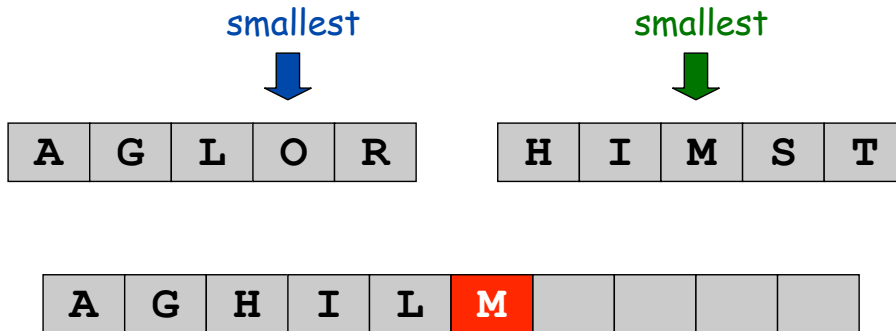


auxiliary array

Merging

Merge.

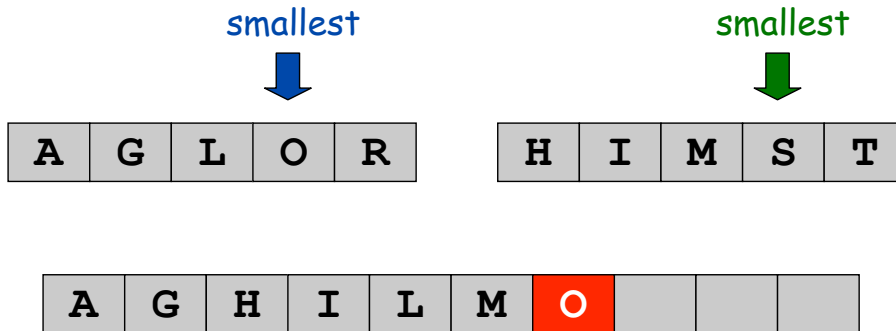
- Keep track of smallest element in each sorted half.
- Insert smallest of two elements into auxiliary array.
- Repeat until done.



Merging

Merge.

- Keep track of smallest element in each sorted half.
- Insert smallest of two elements into auxiliary array.
- Repeat until done.

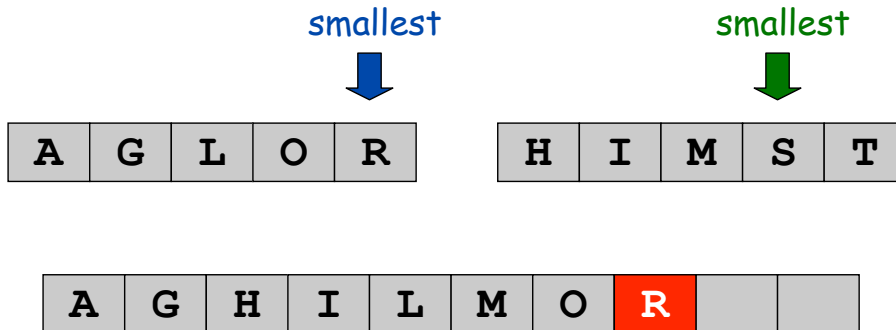


auxiliary array

Merging

Merge.

- Keep track of smallest element in each sorted half.
- Insert smallest of two elements into auxiliary array.
- Repeat until done.

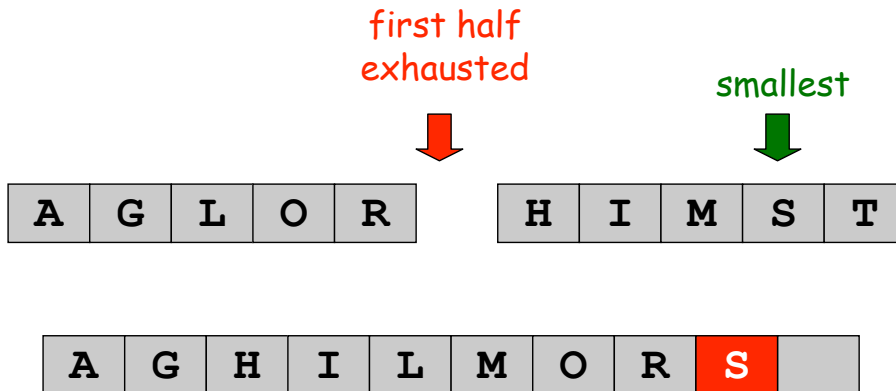


auxiliary array

Merging

Merge.

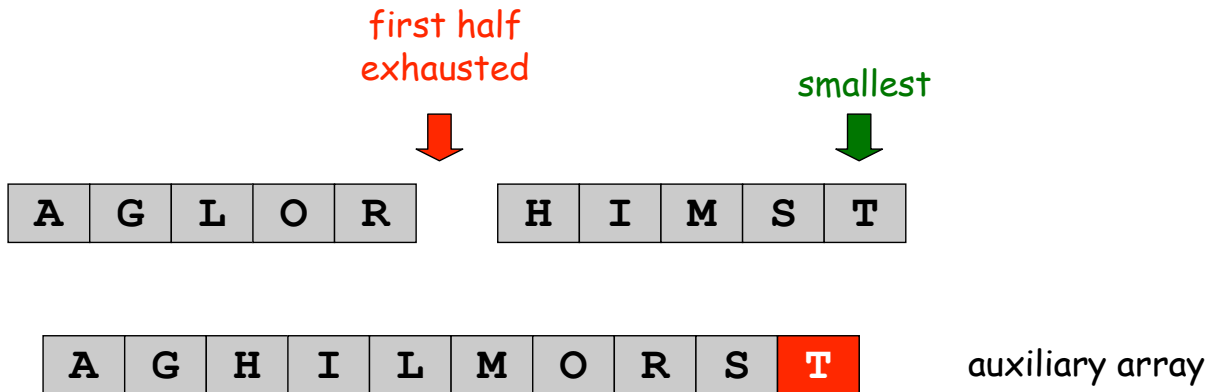
- Keep track of smallest element in each sorted half.
- Insert smallest of two elements into auxiliary array.
- Repeat until done.



Merging

Merge.

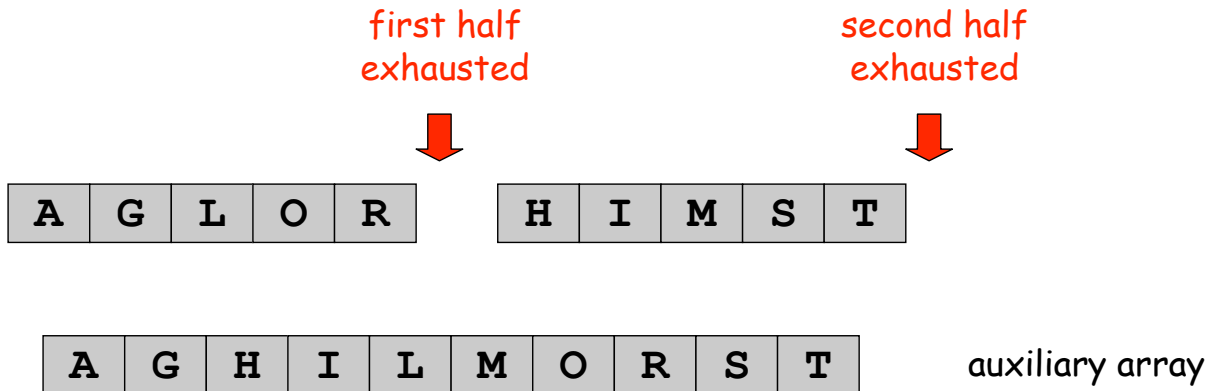
- Keep track of smallest element in each sorted half.
- Insert smallest of two elements into auxiliary array.
- Repeat until done.



Merging

Merge.

- Keep track of smallest element in each sorted half.
- Insert smallest of two elements into auxiliary array.
- Repeat until done.



Merging

Merging. Combine two pre-sorted lists into a sorted whole.

How to merge efficiently?



- Linear number of comparisons.
- Use temporary array.



Challenge for the bored. In-place merge. [Kronrud, 1969]

↑
using only a constant amount of extra storage

A Useful Recurrence Relation

Def. $T(n)$ = number of comparisons to mergesort an input of size n .

Mergesort recurrence.

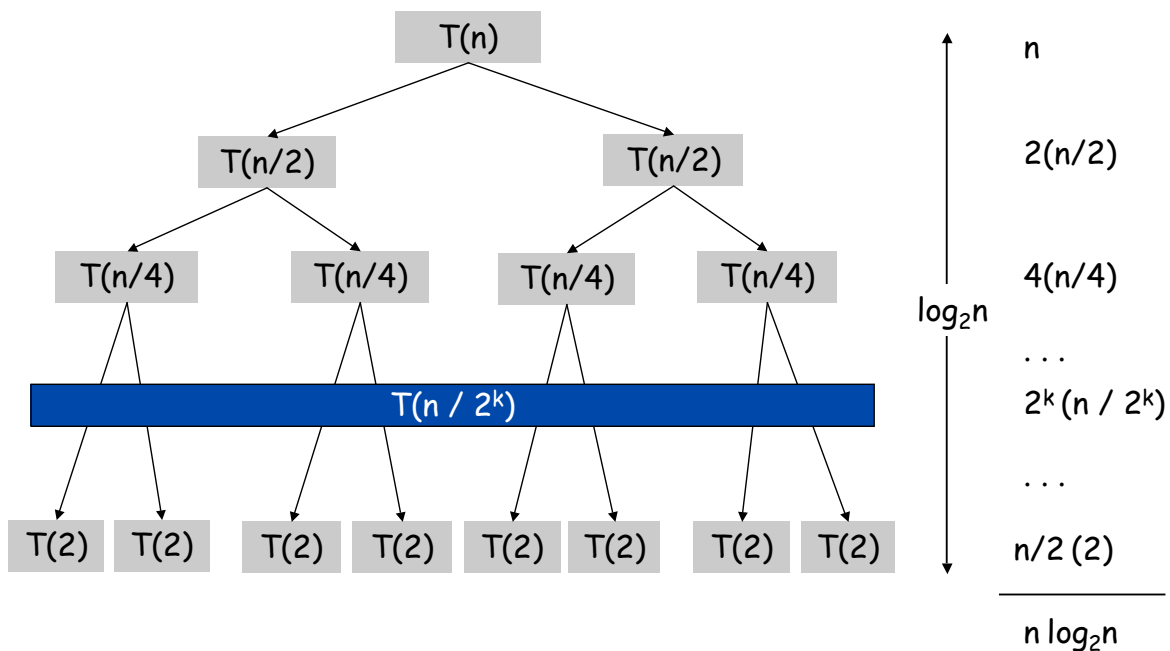
$$T(n) \leq \begin{cases} 0 & \text{if } n=1 \\ \underbrace{T(\lceil n/2 \rceil)}_{\text{solve left half}} + \underbrace{T(\lfloor n/2 \rfloor)}_{\text{solve right half}} + \underbrace{n}_{\text{merging}} & \text{otherwise} \end{cases}$$

Solution. $T(n) = O(n \log_2 n)$.

Assorted proofs. We describe several ways to prove this recurrence. Initially we assume n is a power of 2 and replace \leq with $=$.

Proof by Recursion Tree

$$T(n) = \begin{cases} 0 & \text{if } n = 1 \\ \underbrace{2T(n/2)}_{\text{sorting both halves}} + \underbrace{n}_{\text{merging}} & \text{otherwise} \end{cases}$$



Proof by Telescoping

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

↑
assumes n is a power of 2

$$T(n) = \begin{cases} 0 & \text{if } n=1 \\ \underbrace{2T(n/2)}_{\text{sorting both halves}} + \underbrace{n}_{\text{merging}} & \text{otherwise} \end{cases}$$

Pf. For $n > 1$:

$$\begin{aligned} \frac{T(n)}{n} &= \frac{2T(n/2)}{n} + 1 \\ &= \frac{T(n/2)}{n/2} + 1 \\ &= \frac{T(n/4)}{n/4} + 1 + 1 \\ &\dots \\ &= \frac{T(n/n)}{n/n} + \underbrace{1 + \dots + 1}_{\log_2 n} \\ &= \log_2 n \end{aligned}$$

Proof by Induction

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

↑
assumes n is a power of 2

$$T(n) = \begin{cases} 0 & \text{if } n=1 \\ \underbrace{2T(n/2)}_{\text{sorting both halves}} + \underbrace{n}_{\text{merging}} & \text{otherwise} \end{cases}$$

Pf. (by induction on n)

- Base case: $n = 1$.
- Goal: show that $T(2n) = 2n \log_2(2n)$.

$$\begin{aligned} T(2n) &= 2T(n) + 2n \\ &= 2n \log_2 n + 2n \\ &= 2n(\log_2(2n) - 1) + 2n \\ &= 2n \log_2(2n) \end{aligned}$$

5.3 Counting Inversions

Counting Inversions

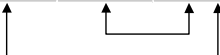
Music site tries to match your song preferences with others.

- You rank n songs.
- Music site consults database to find people with **similar** tastes.

Similarity metric: number of inversions between two rankings.

- My rank: $1, 2, \dots, n$.
- Your rank: a_1, a_2, \dots, a_n .
- Songs i and j **inverted** if $i < j$, but $a_i > a_j$.

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



Inversions
3-2, 4-2

Brute force: check all $\Theta(n^2)$ pairs i and j .

Applications

Applications.

- Voting theory.
- Collaborative filtering.
- Measuring the "sortedness" of an array.
- Sensitivity analysis of Google's ranking function.
- Rank aggregation for meta-searching on the Web.
- Nonparametric statistics (e.g., Kendall's Tau distance).

Counting Inversions: Divide-and-Conquer

Divide-and-conquer.

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

Counting Inversions: Divide-and-Conquer

Divide-and-conquer.

- **Divide:** separate list into two pieces.

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
---	---	---	---	----	---	---	---	----	----	---	---

Counting Inversions: Divide-and-Conquer

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
---	---	---	---	----	---	---	---	----	----	---	---

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

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

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

Counting Inversions: Divide-and-Conquer

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

9 blue-green inversions

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

Combine: ???

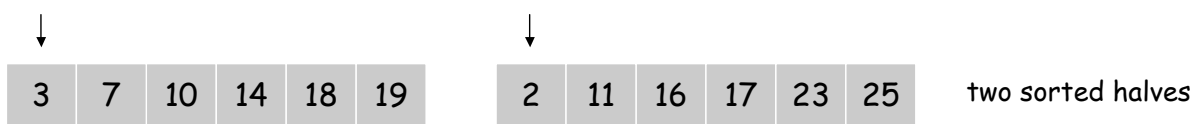
Total = $5 + 8 + 9 = 22$.

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.

$i = 6$



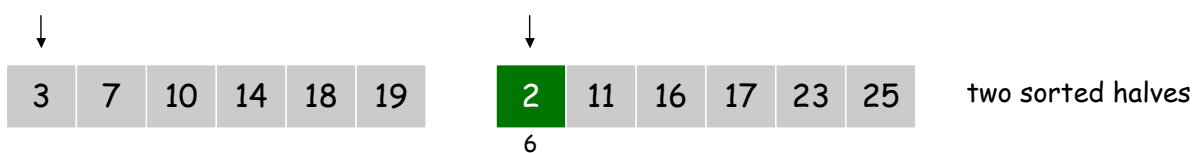
Total:

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.

$i = 6$



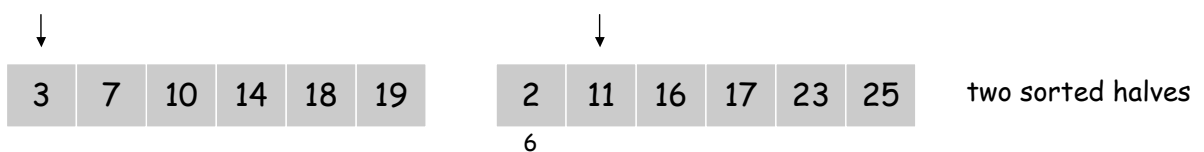
Total: 6

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.

$i = 6$



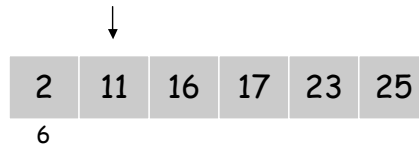
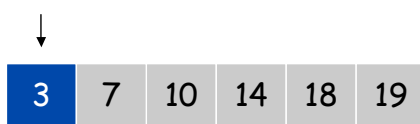
Total: 6

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.

$i = 6$



two sorted halves



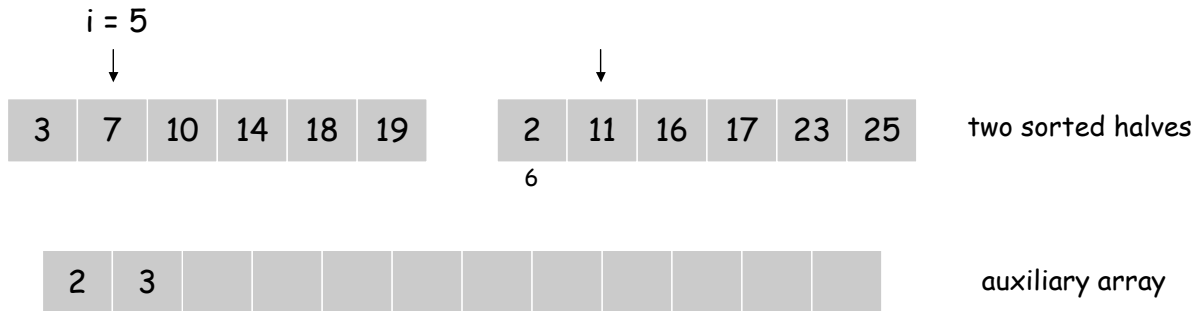
auxiliary array

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.

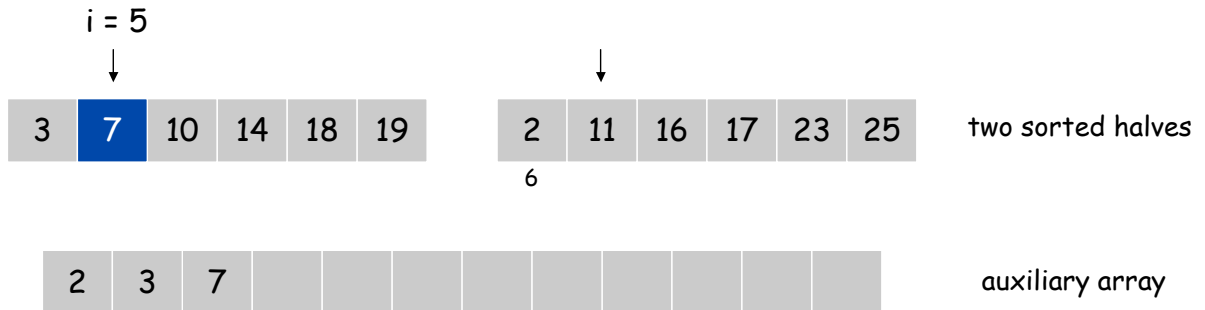


Total: 6

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.

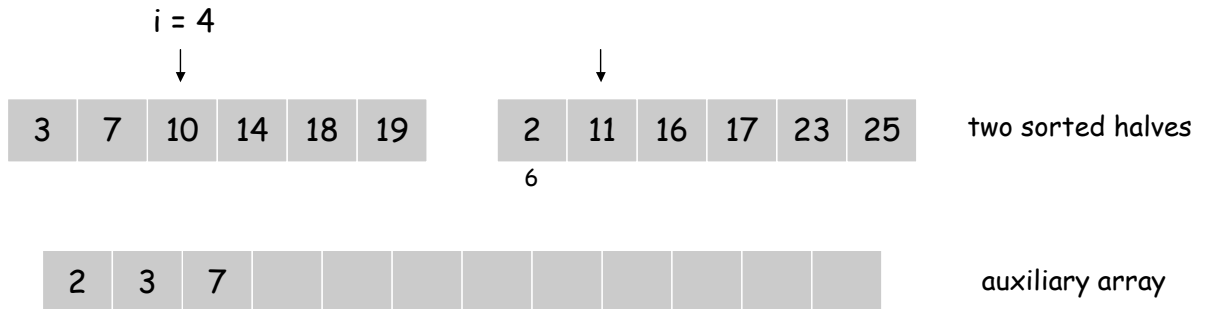


Total: 6

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.

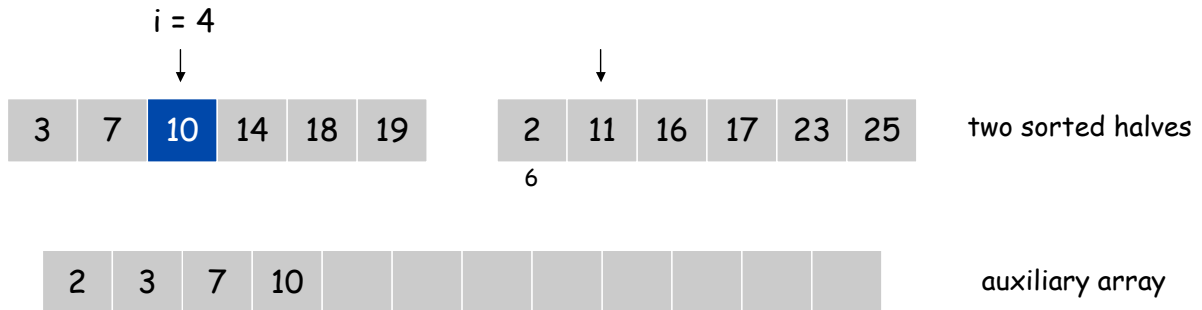


Total: 6

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.

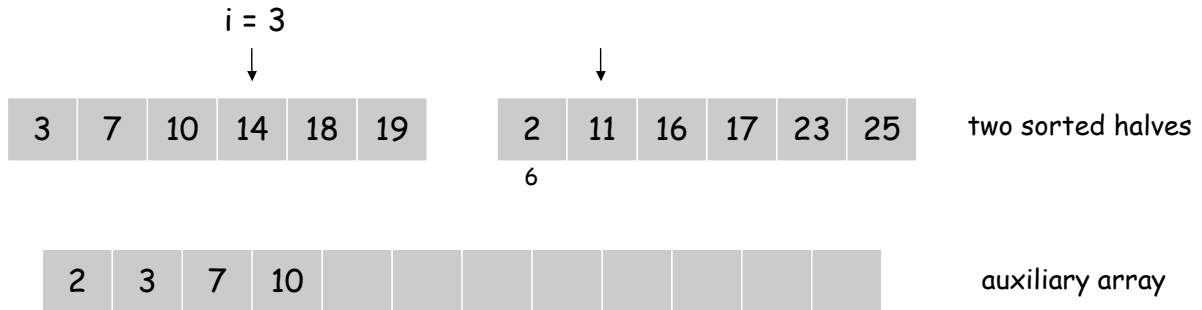


Total: 6

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.

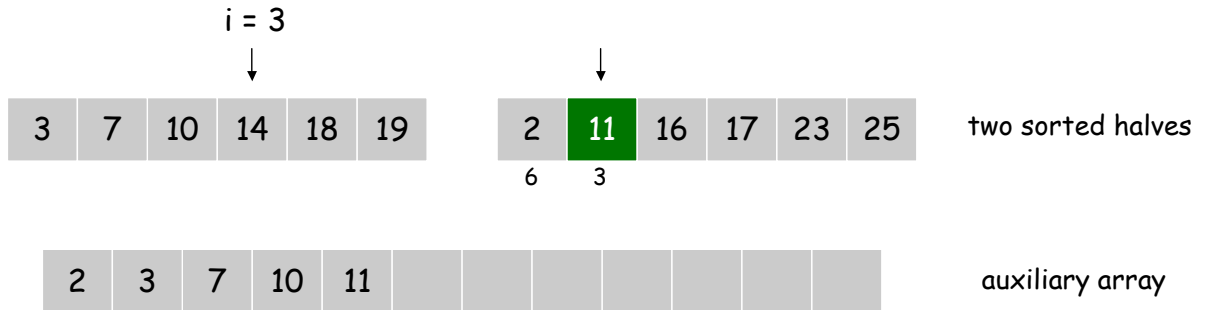


Total: 6

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.

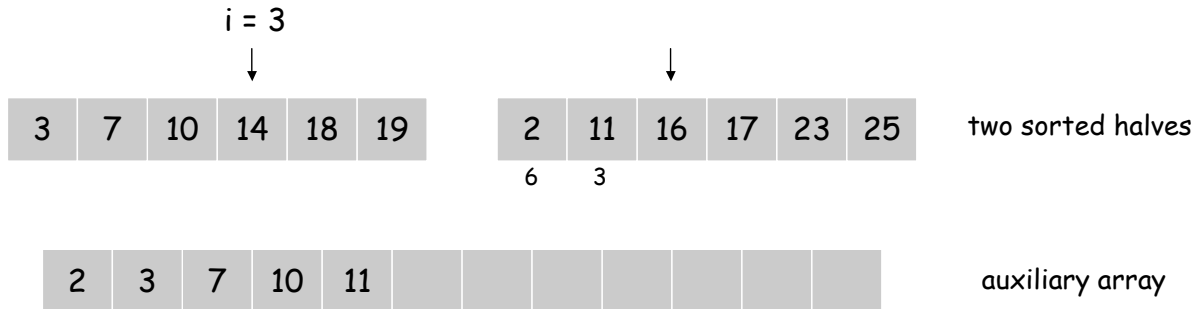


Total: 6 + 3

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.

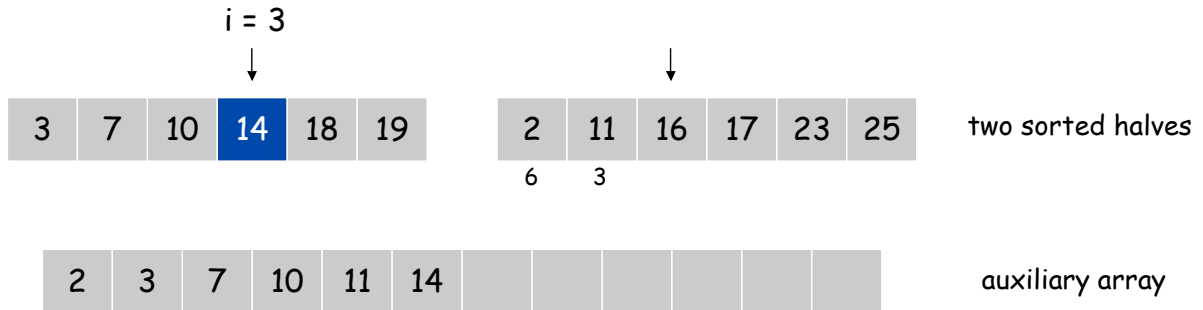


Total: 6 + 3

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.

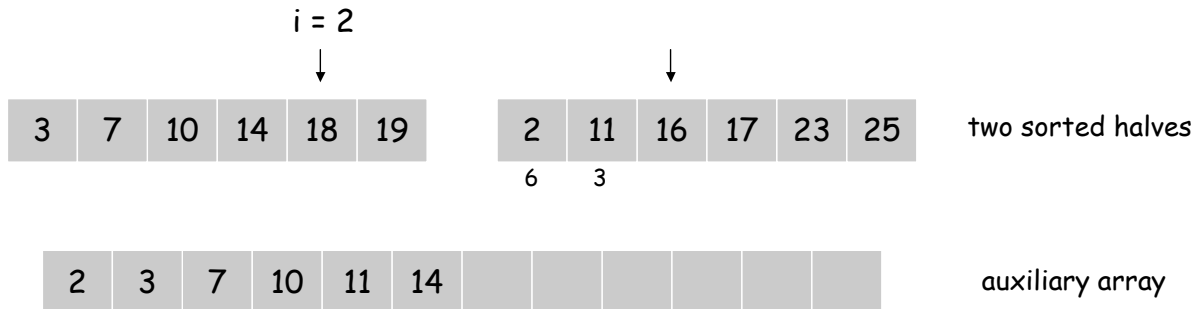


Total: 6 + 3

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.

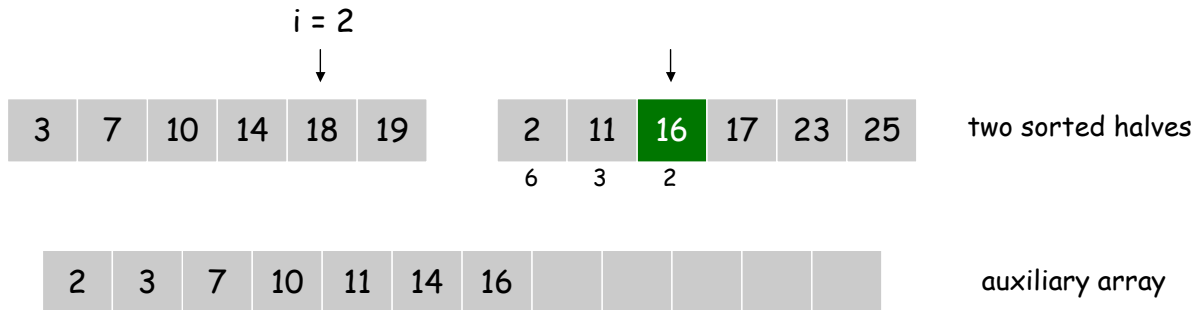


Total: 6 + 3

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.

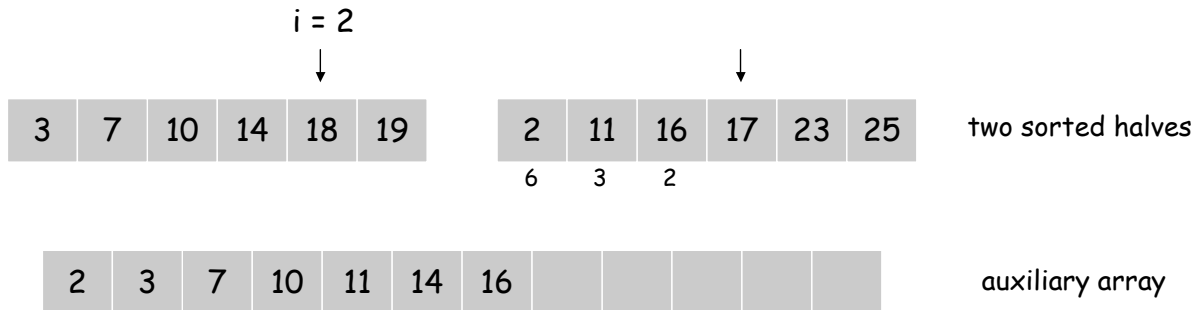


Total: $6 + 3 + 2$

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.

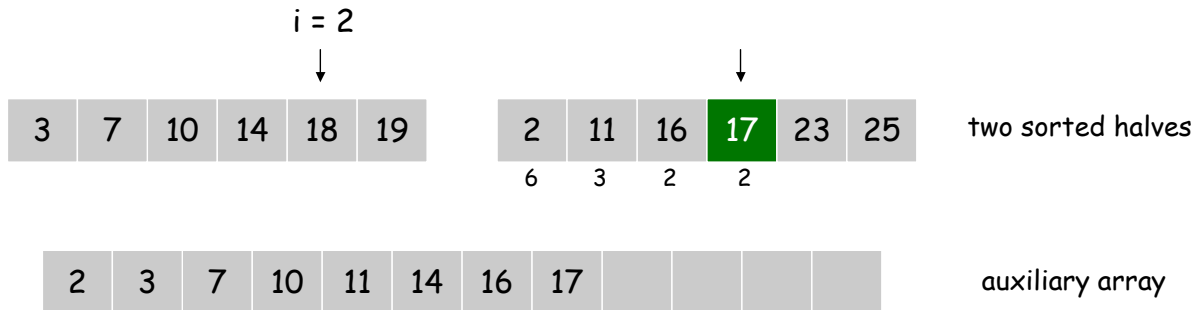


Total: $6 + 3 + 2$

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.

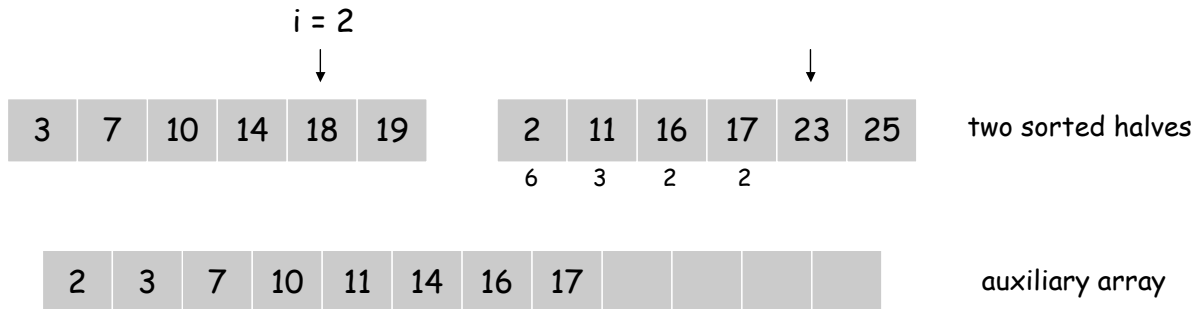


Total: $6 + 3 + 2 + 2$

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.

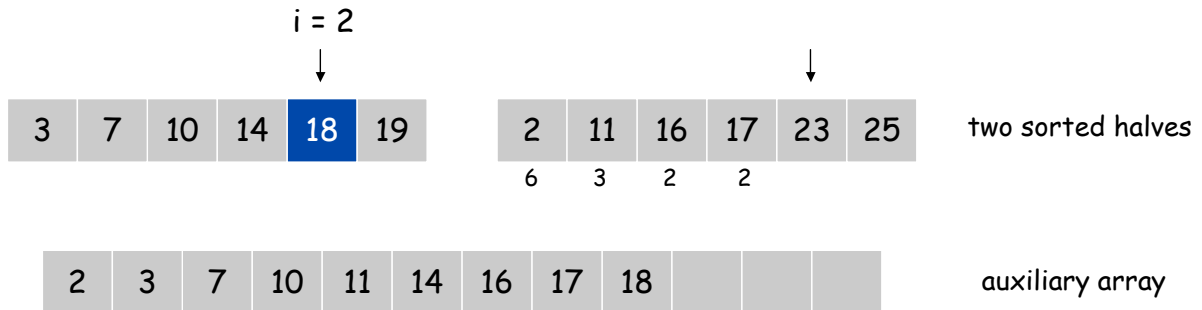


Total: $6 + 3 + 2 + 2$

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.

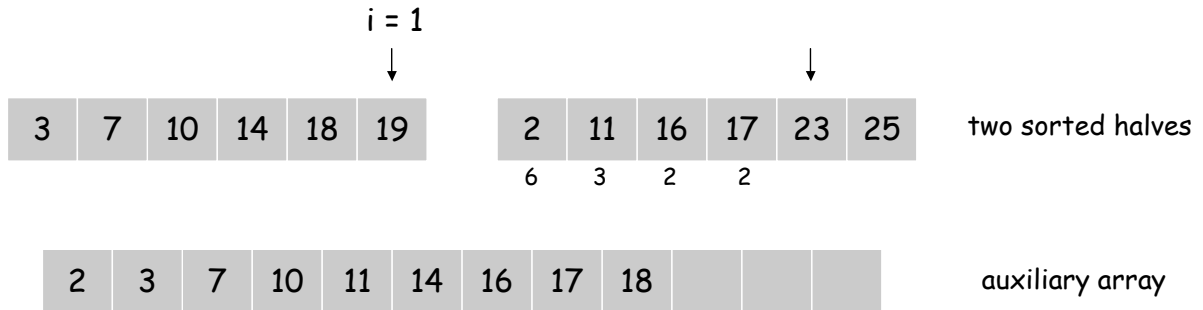


Total: $6 + 3 + 2 + 2$

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.

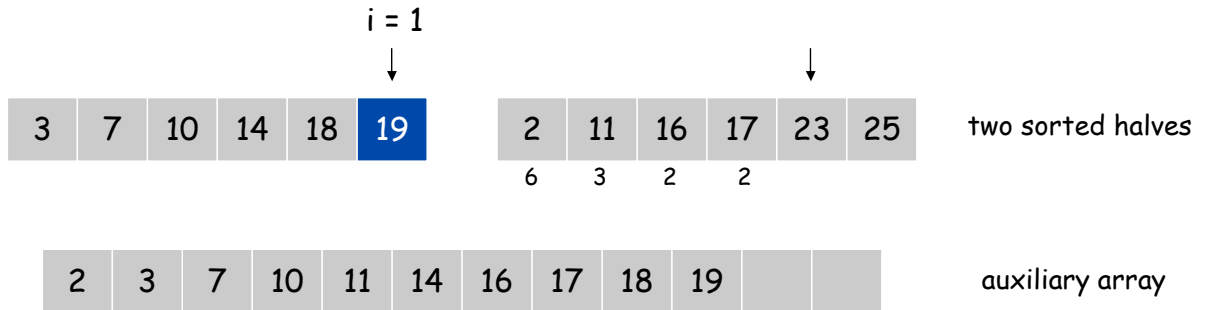


Total: $6 + 3 + 2 + 2$

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.

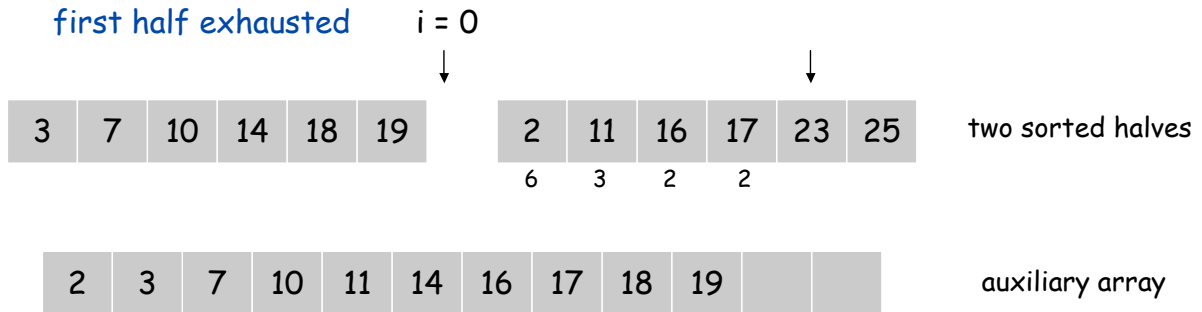


Total: $6 + 3 + 2 + 2$

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.

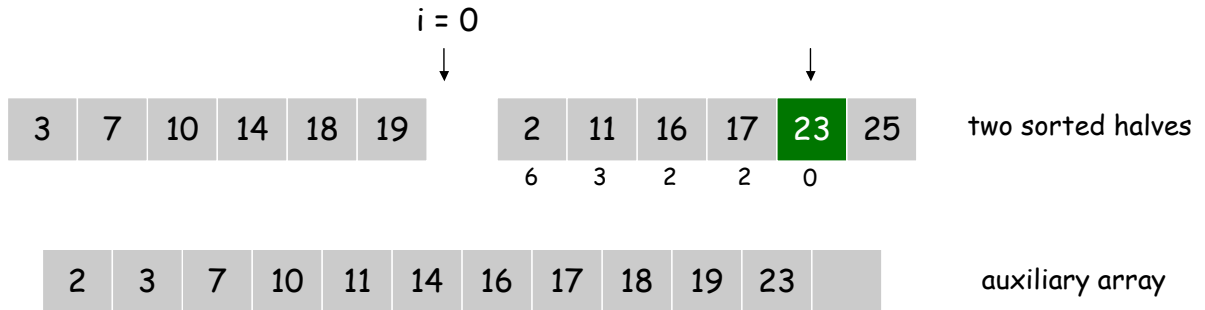


Total: $6 + 3 + 2 + 2$

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.

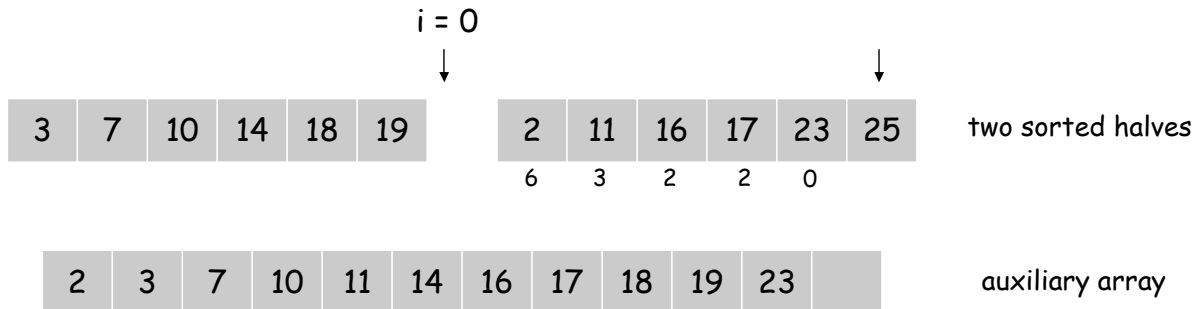


Total: $6 + 3 + 2 + 2 + 0$

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.

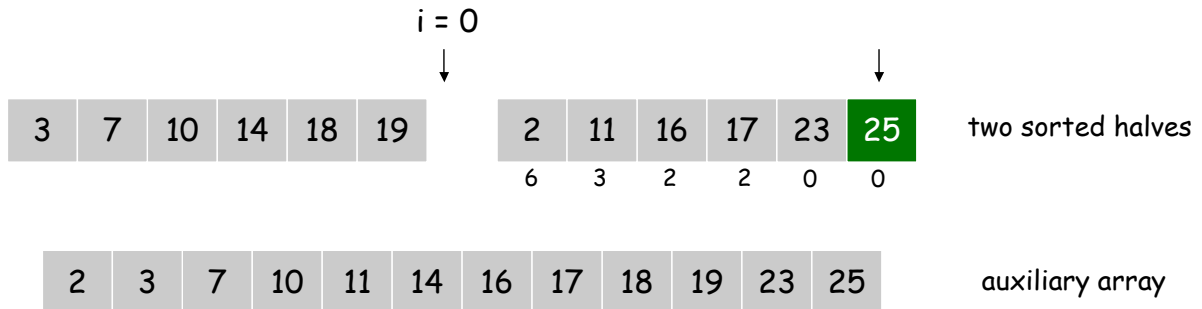


Total: $6 + 3 + 2 + 2 + 0$

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.

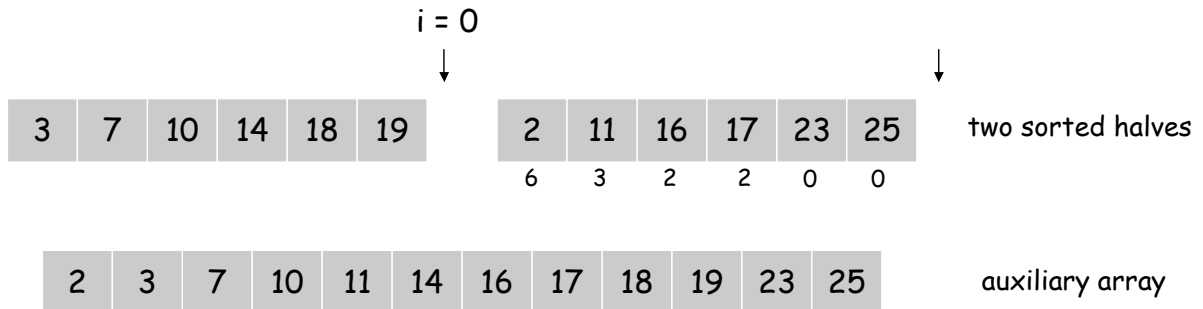


Total: $6 + 3 + 2 + 2 + 0 + 0$

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.



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

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
6	3	2	2	0	0

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

Count: $O(n)$

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

Merge: $O(n)$

$$T(n) \leq T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + O(n) \Rightarrow T(n) = O(n \log n)$$

Counting Inversions: Implementation

Pre-condition. [Merge-and-Count] A and B are sorted.

Post-condition. [Sort-and-Count] L is sorted.

```
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  
    ( $r_A$ , A)  $\leftarrow$  Sort-and-Count(A)  
    ( $r_B$ , B)  $\leftarrow$  Sort-and-Count(B)  
    ( $r$ , L)  $\leftarrow$  Merge-and-Count(A, B)  
  
    return  $r = r_A + r_B + r$  and the sorted list L  
}
```

5.4 Closest Pair of Points

Closest Pair of Points

Closest pair. Given n points in the plane, find a pair with smallest Euclidean distance between them.

Fundamental geometric primitive.

- Graphics, computer vision, geographic information systems, molecular modeling, air traffic control.
- Special case of nearest neighbor, Euclidean MST, Voronoi.

↑ fast closest pair inspired fast algorithms for these problems

Brute force. Check all pairs of points p and q with $\Theta(n^2)$ comparisons.

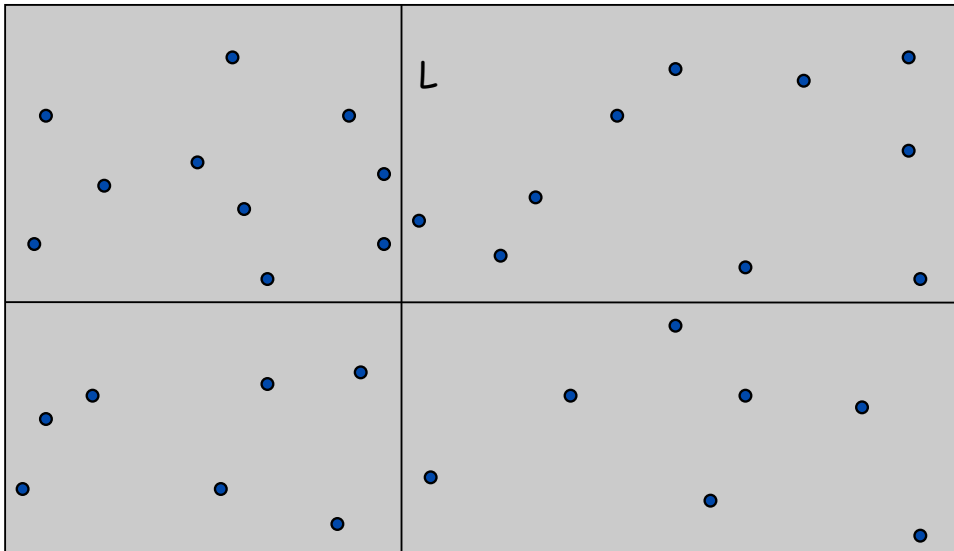
1-D version. $O(n \log n)$ easy if points are on a line.

Assumption. No two points have same x coordinate.

↑
to make presentation cleaner

Closest Pair of Points: First Attempt

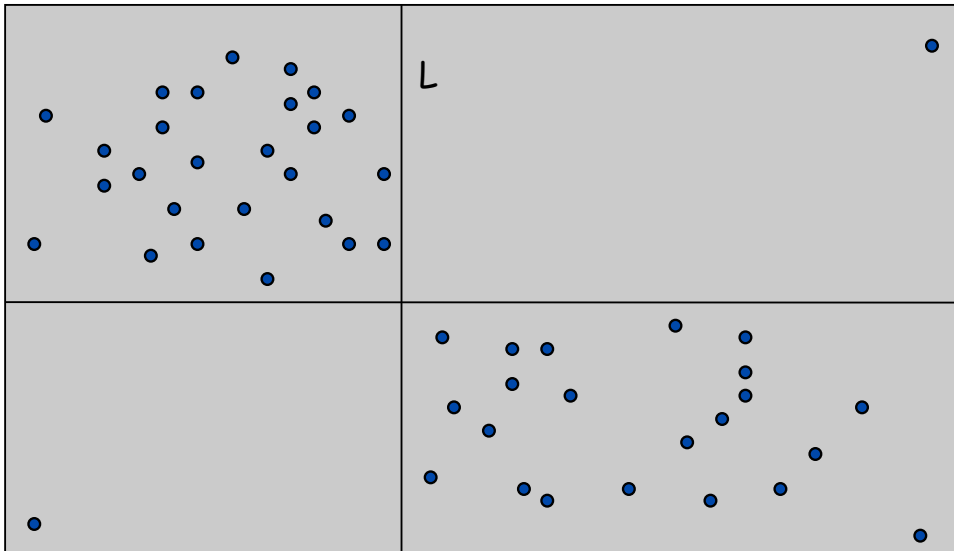
Divide. Sub-divide region into 4 quadrants.



Closest Pair of Points: First Attempt

Divide. Sub-divide region into 4 quadrants.

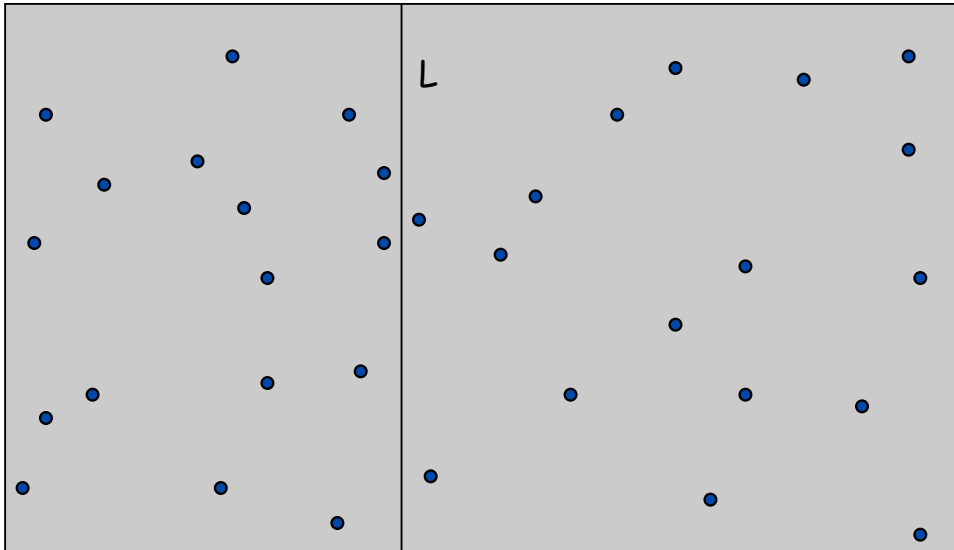
Obstacle. Impossible to ensure $n/4$ points in each piece.



Closest Pair of Points

Algorithm.

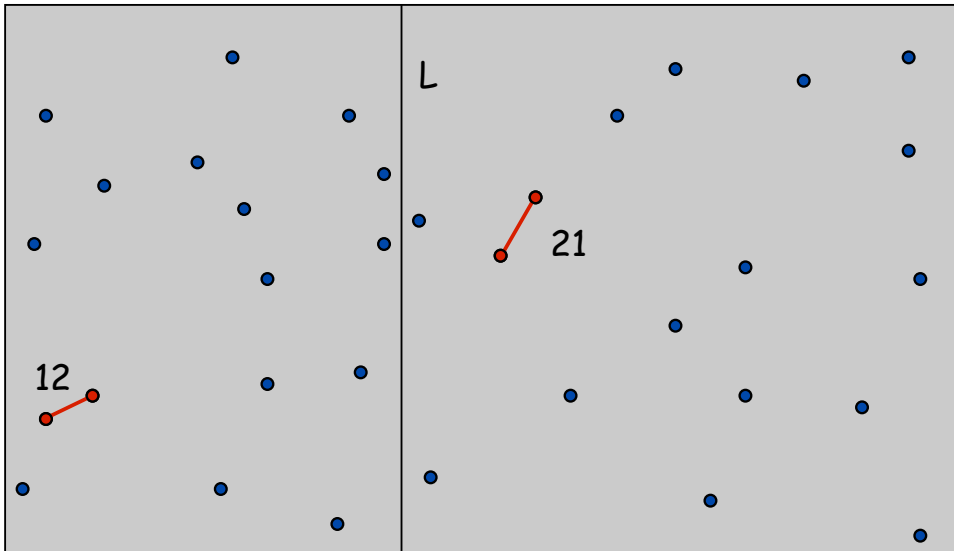
- **Divide:** draw vertical line L so that roughly $\frac{1}{2}n$ points on each side.



Closest Pair of Points

Algorithm.

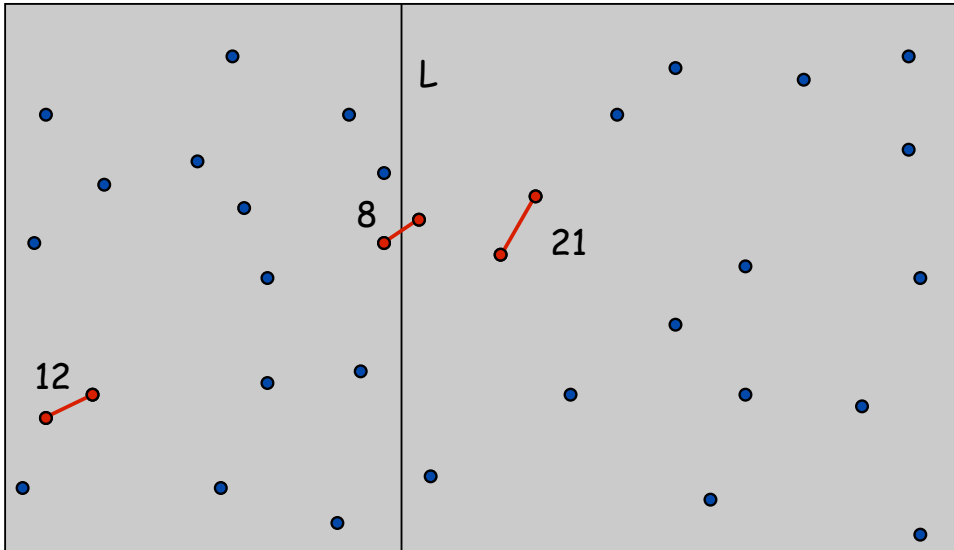
- Divide: draw vertical line L so that roughly $\frac{1}{2}n$ points on each side.
- **Conquer**: find closest pair in each side recursively.



Closest Pair of Points

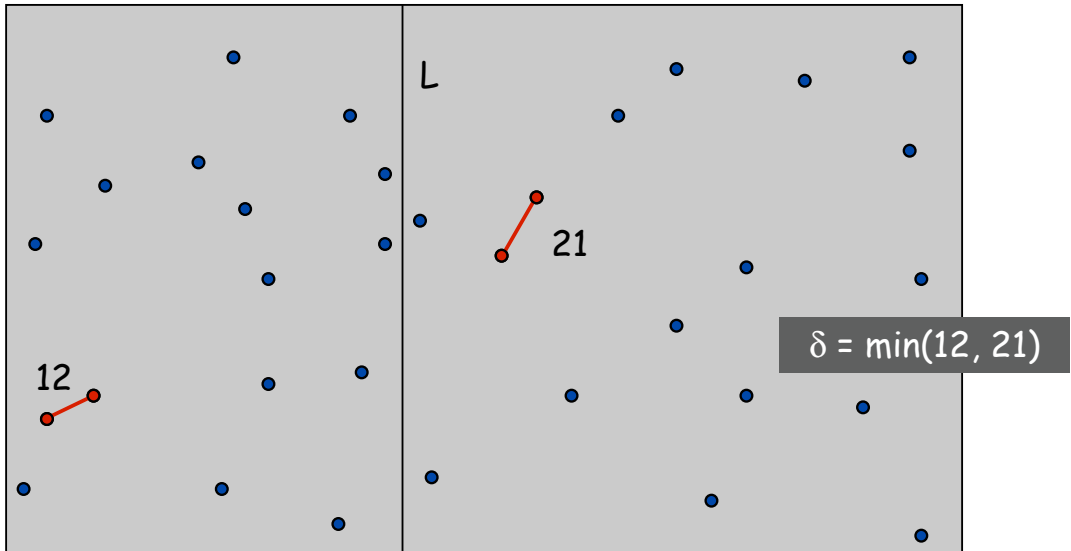
Algorithm.

- Divide: draw vertical line L so that roughly $\frac{1}{2}n$ 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.



Closest Pair of Points

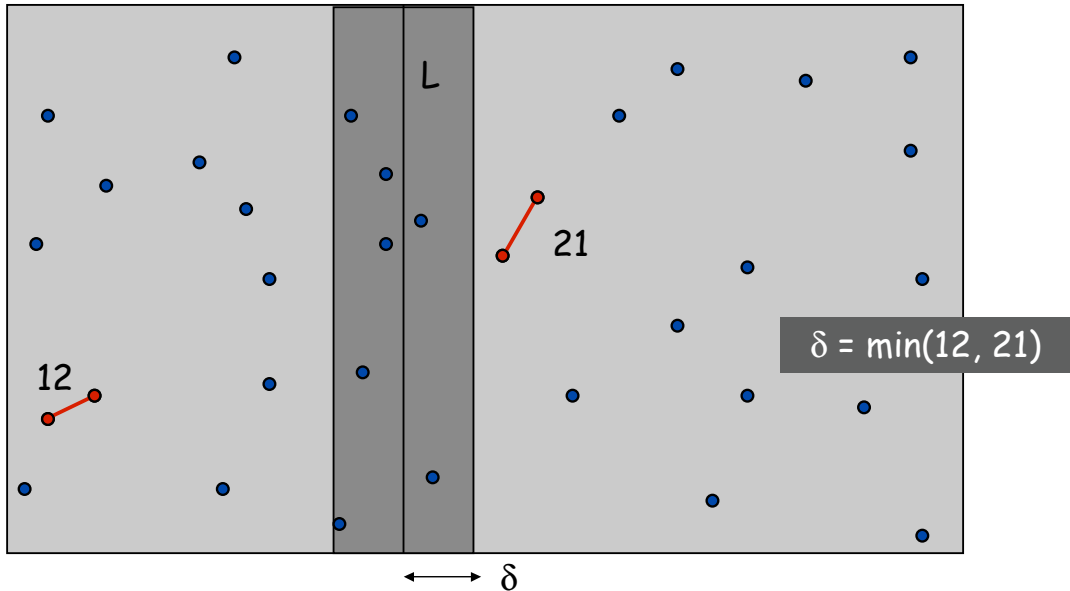
Find closest pair with one point in each side, **assuming that distance $< \delta$** .



Closest Pair of Points

Find closest pair with one point in each side, **assuming that distance $< \delta$** .

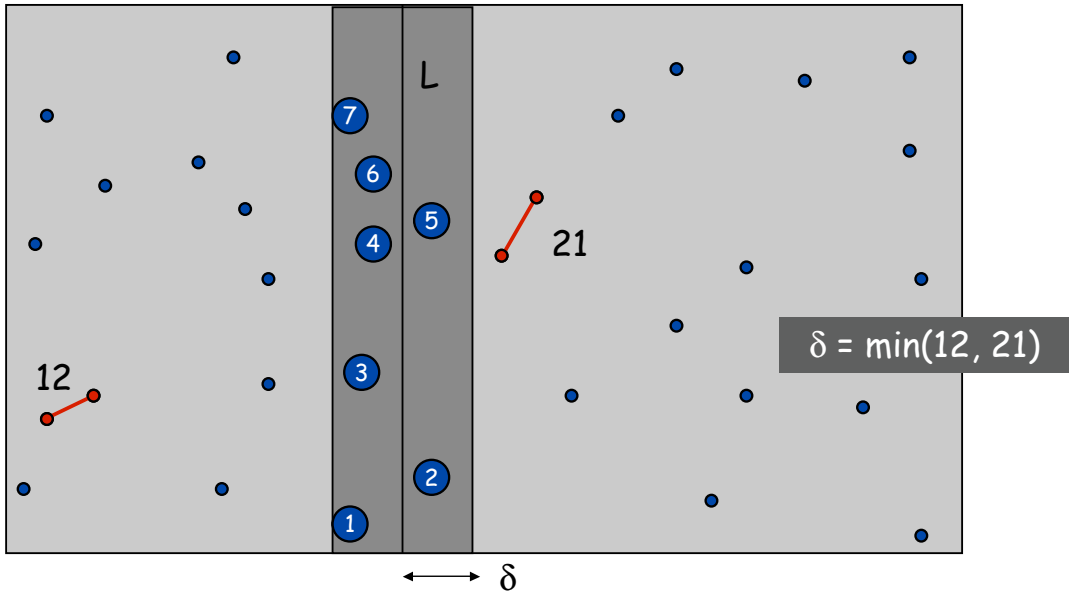
- Observation: only need to consider points within δ of line L .



Closest Pair of Points

Find closest pair with one point in each side, **assuming that distance $< \delta$** .

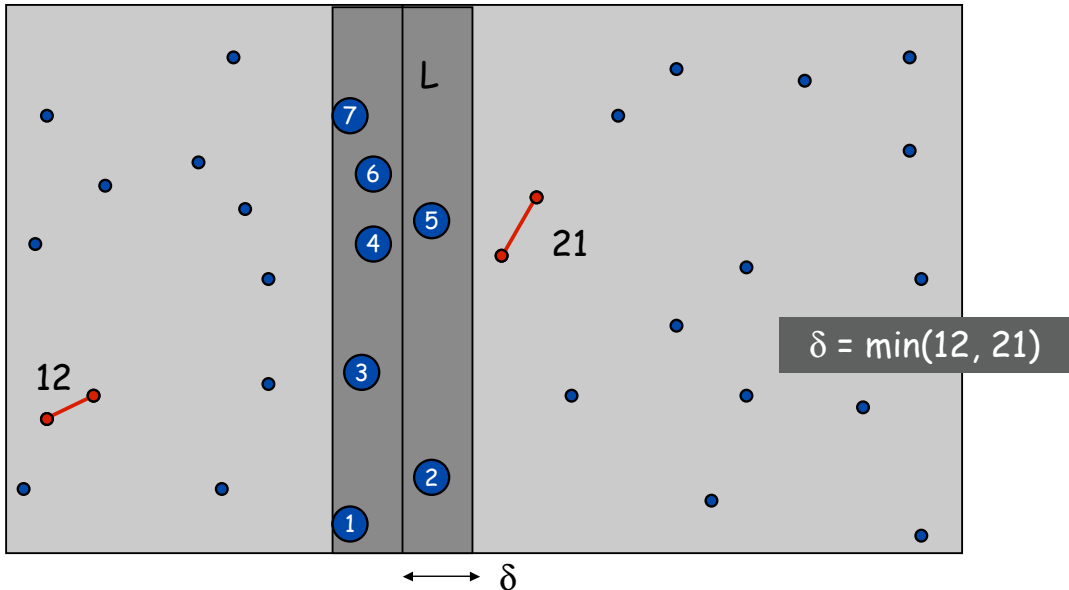
- Observation: only need to consider points within δ of line L .
- Sort points in 2δ -strip by their y coordinate.



Closest Pair of Points

Find closest pair with one point in each side, **assuming that distance $< \delta$** .

- Observation: only need to consider points within δ of line L .
- Sort points in 2δ -strip by their y coordinate.
- Only check distances of those within 11 positions in sorted list!



Closest Pair of Points

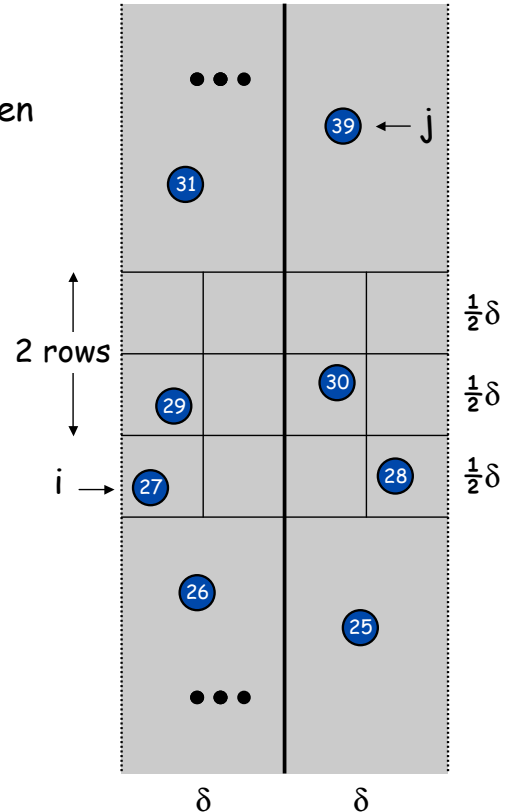
Def. Let s_i be the point in the 2δ -strip, with the i^{th} smallest y -coordinate.

Claim. If $|i - j| \geq 12$, then the distance between s_i and s_j is at least δ .

Pf.

- No two points lie in same $\frac{1}{2}\delta$ -by- $\frac{1}{2}\delta$ box.
- Two points at least 2 rows apart have distance $\geq 2(\frac{1}{2}\delta)$. ▪

Fact. Still true if we replace 12 with 7.



Closest Pair Algorithm

```
Closest-Pair( $p_1, \dots, p_n$ ) {  
  Compute separation line  $L$  such that half the points  
  are on one side and half on the other side.  $O(n \log n)$   
  
   $\delta_1 = \text{Closest-Pair}(\text{left half})$   $2T(n / 2)$   
   $\delta_2 = \text{Closest-Pair}(\text{right half})$   
   $\delta = \min(\delta_1, \delta_2)$   
  
  Delete all points further than  $\delta$  from separation line  $L$   $O(n)$   
  
  Sort remaining points by  $y$ -coordinate.  $O(n \log n)$   
  
  Scan points in  $y$ -order and compare distance between  
  each point and next 11 neighbors. If any of these  
  distances is less than  $\delta$ , update  $\delta$ .  $O(n)$   
  
  return  $\delta$ .  
}
```

Closest Pair of Points: Analysis

Running time.

$$T(n) \leq 2T(n/2) + O(n \log n) \Rightarrow T(n) = O(n \log^2 n)$$

Q. Can we achieve $O(n \log n)$?

A. Yes. Don't sort points in strip from scratch each time.

- Each recursive returns two lists: all points sorted by y coordinate, and all points sorted by x coordinate.
- Sort by **merging** two pre-sorted lists.

$$T(n) \leq 2T(n/2) + O(n) \Rightarrow T(n) = O(n \log n)$$

5.5 Integer Multiplication

Divide-and-Conquer Multiplication: Warmup

To multiply two n -digit integers:

- Multiply four $\frac{1}{2}n$ -digit integers.
- Add two $\frac{1}{2}n$ -digit integers, and shift to obtain result.

$$\begin{aligned}x &= 2^{n/2} \cdot x_1 + x_0 \\y &= 2^{n/2} \cdot y_1 + y_0 \\xy &= (2^{n/2} \cdot x_1 + x_0)(2^{n/2} \cdot y_1 + y_0) = 2^n \cdot x_1y_1 + 2^{n/2} \cdot (x_1y_0 + x_0y_1) + x_0y_0\end{aligned}$$

$$T(n) = \underbrace{4T(n/2)}_{\text{recursive calls}} + \underbrace{\Theta(n)}_{\text{add, shift}} \Rightarrow T(n) = \Theta(n^2)$$

↑
assumes n is a power of 2

Karatsuba Multiplication

To multiply two n -digit integers:

- Add two $\frac{1}{2}n$ digit integers.
- Multiply **three** $\frac{1}{2}n$ -digit integers.
- Add, subtract, and shift $\frac{1}{2}n$ -digit integers to obtain result.

$$\begin{aligned}x &= 2^{n/2} \cdot x_1 + x_0 \\y &= 2^{n/2} \cdot y_1 + y_0 \\xy &= 2^n \cdot x_1 y_1 + 2^{n/2} \cdot (x_1 y_0 + x_0 y_1) + x_0 y_0 \\&= 2^n \cdot x_1 y_1 + 2^{n/2} \cdot ((x_1 + x_0)(y_1 + y_0) - x_1 y_1 - x_0 y_0) + x_0 y_0\end{aligned}$$

A B A C C

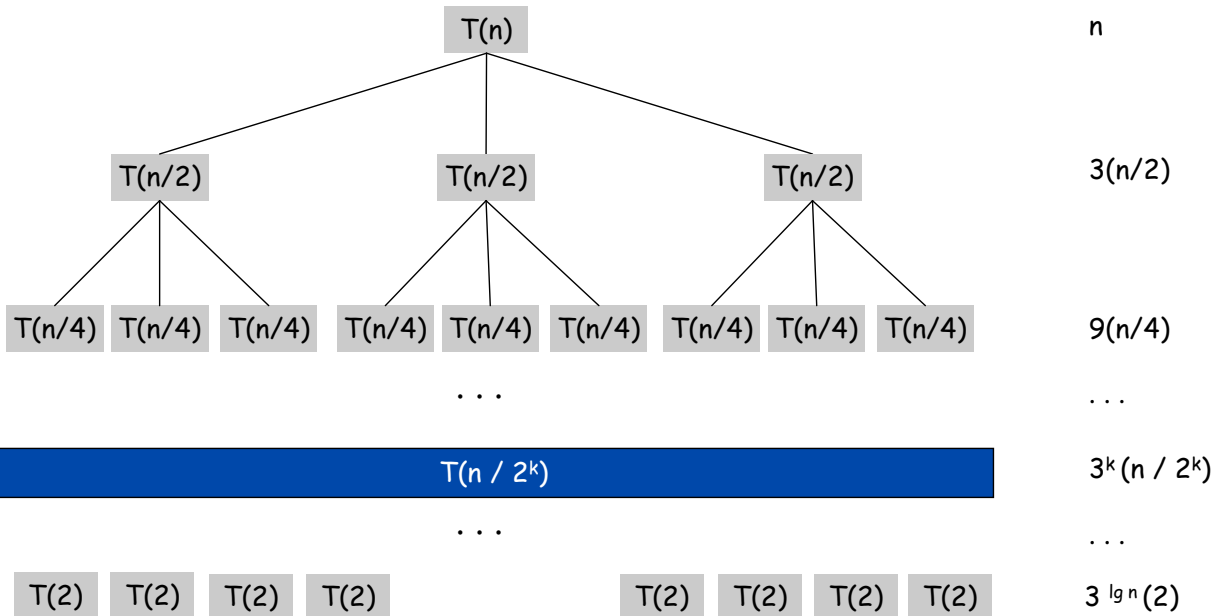
Theorem. [Karatsuba-Ofman, 1962] Can multiply two n -digit integers in $O(n^{1.585})$ bit operations.

$$\begin{aligned}T(n) &\leq \underbrace{T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + T(1 + \lceil n/2 \rceil)}_{\text{recursive calls}} + \underbrace{\Theta(n)}_{\text{add, subtract, shift}} \\ \Rightarrow T(n) &= O(n^{\log_2 3}) = O(n^{1.585})\end{aligned}$$

Karatsuba: Recursion Tree

$$T(n) = \begin{cases} 0 & \text{if } n=1 \\ 3T(n/2) + n & \text{otherwise} \end{cases}$$

$$T(n) = \sum_{k=0}^{\log_2 n} n \left(\frac{3}{2}\right)^k = \frac{\left(\frac{3}{2}\right)^{1+\log_2 n} - 1}{\frac{3}{2} - 1} = 3n^{\log_2 3} - 2$$



Matrix Multiplication

Matrix Multiplication

Matrix multiplication. Given two n -by- n matrices A and B , compute $C = AB$.

$$c_{ij} = \sum_{k=1}^n a_{ik} b_{kj}$$

$$\begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ c_{21} & c_{22} & \cdots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & \cdots & c_{nn} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \times \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{nn} \end{bmatrix}$$

Brute force. $\Theta(n^3)$ arithmetic operations.

Fundamental question. Can we improve upon brute force?

Matrix Multiplication: Warmup

Divide-and-conquer.

- Divide: partition A and B into $\frac{1}{2}n$ -by- $\frac{1}{2}n$ blocks.
- Conquer: multiply 8 $\frac{1}{2}n$ -by- $\frac{1}{2}n$ recursively.
- Combine: add appropriate products using 4 matrix additions.

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \times \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

$$\begin{aligned} C_{11} &= (A_{11} \times B_{11}) + (A_{12} \times B_{21}) \\ C_{12} &= (A_{11} \times B_{12}) + (A_{12} \times B_{22}) \\ C_{21} &= (A_{21} \times B_{11}) + (A_{22} \times B_{21}) \\ C_{22} &= (A_{21} \times B_{12}) + (A_{22} \times B_{22}) \end{aligned}$$

$$T(n) = \underbrace{8T(n/2)}_{\text{recursive calls}} + \underbrace{\Theta(n^2)}_{\text{add, form submatrices}} \Rightarrow T(n) = \Theta(n^3)$$

Matrix Multiplication: Key Idea

Key idea. multiply 2-by-2 block matrices with only **7** multiplications.

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \times \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

$$C_{11} = P_5 + P_4 - P_2 + P_6$$

$$C_{12} = P_1 + P_2$$

$$C_{21} = P_3 + P_4$$

$$C_{22} = P_5 + P_1 - P_3 - P_7$$

$$P_1 = A_{11} \times (B_{12} - B_{22})$$

$$P_2 = (A_{11} + A_{12}) \times B_{22}$$

$$P_3 = (A_{21} + A_{22}) \times B_{11}$$

$$P_4 = A_{22} \times (B_{21} - B_{11})$$

$$P_5 = (A_{11} + A_{22}) \times (B_{11} + B_{22})$$

$$P_6 = (A_{12} - A_{22}) \times (B_{21} + B_{22})$$

$$P_7 = (A_{11} - A_{21}) \times (B_{11} + B_{12})$$

- 7 multiplications.
- 18 = 10 + 8 additions (or subtractions).

Fast Matrix Multiplication

Fast matrix multiplication. (Strassen, 1969)

- Divide: partition A and B into $\frac{1}{2}n$ -by- $\frac{1}{2}n$ blocks.
- Compute: 14 $\frac{1}{2}n$ -by- $\frac{1}{2}n$ matrices via 10 matrix additions.
- Conquer: multiply 7 $\frac{1}{2}n$ -by- $\frac{1}{2}n$ matrices recursively.
- Combine: 7 products into 4 terms using 8 matrix additions.

Analysis.

- Assume n is a power of 2.
- $T(n)$ = # arithmetic operations.

$$T(n) = \underbrace{7T(n/2)}_{\text{recursive calls}} + \underbrace{\Theta(n^2)}_{\text{add, subtract}} \Rightarrow T(n) = \Theta(n^{\log_2 7}) = O(n^{2.81})$$

Fast Matrix Multiplication in Practice

Implementation issues.

- Sparsity.
- Caching effects.
- Numerical stability.
- Odd matrix dimensions.
- Crossover to classical algorithm around $n = 128$.

Common misperception: "Strassen is only a theoretical curiosity."

- Advanced Computation Group at Apple Computer reports 8x speedup on G4 Velocity Engine when $n \sim 2,500$.
- Range of instances where it's useful is a subject of controversy.

Remark. Can "Strassenize" $Ax=b$, determinant, eigenvalues, and other matrix ops.

Fast Matrix Multiplication in Theory

Q. Multiply two 2-by-2 matrices with only 7 scalar multiplications?

A. Yes! [Strassen, 1969]

$$\Theta(n^{\log_2 7}) = O(n^{2.81})$$

Q. Multiply two 2-by-2 matrices with only 6 scalar multiplications?

A. Impossible. [Hopcroft and Kerr, 1971]

$$\Theta(n^{\log_2 6}) = O(n^{2.59})$$

Q. Two 3-by-3 matrices with only 21 scalar multiplications?

A. Also impossible.

$$\Theta(n^{\log_3 21}) = O(n^{2.77})$$

Q. Two 70-by-70 matrices with only 143,640 scalar multiplications?

A. Yes! [Pan, 1980]

$$\Theta(n^{\log_{70} 143640}) = O(n^{2.80})$$

Decimal wars.

- December, 1979: $O(n^{2.521813})$.
- January, 1980: $O(n^{2.521801})$.

Fast Matrix Multiplication in Theory

Best known. $O(n^{2.376})$ [Coppersmith-Winograd, 1987.]

Conjecture. $O(n^{2+\epsilon})$ for any $\epsilon > 0$.

Caveat. Theoretical improvements to Strassen are progressively less practical.