

Divide and Conquer

CMSC 451, Summer 2009

- ## Group Project
- Explain project
 - Choose groups and register on submit server
 - Submit server test suite will be live later this week... exact submission instructions may change.

 - Remember: Homework 1 due now!

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**.

Consequence.

- Brute force: n^2 .
- Divide-and-conquer: $n \log n$.

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

3

5.1 Mergesort

Sorting

Sorting. Given n elements, rearrange in ascending order.

Applications.

- Sort a list of names.
- Organize an MP3 library. obvious applications
- Display Google PageRank results.
- List RSS news items in reverse chronological order.

- Find the median.
- Find the closest pair. problems become easy once items are in sorted order
- Binary search in a database.
- Identify statistical outliers.
- Find duplicates in a mailing list.

- Data compression.
- Computer graphics.
- Computational biology.
- Supply chain management. non-obvious applications
- Book recommendations on Amazon.
- Load balancing on a parallel computer.
- ...

5

Mergesort

Mergesort.

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

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	$O(n)$

6

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

7

Merging

Merge.

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

auxiliary array

8

Merging

Merge.

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

auxiliary array

9

Merging

Merge.

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

auxiliary array

10

Merging

Merge.

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

auxiliary array

11

Merging

Merge.

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

auxiliary array

12

Merging

Merge.

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

auxiliary array

13

Merging

Merge.

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

auxiliary array

14

Merging

Merge.

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

auxiliary array

15

Merging

Merge.

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

auxiliary array

16

Merging

Merge.

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

auxiliary array

17

Merging

Merge.

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

auxiliary array

18

A Useful Recurrence Relation

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

Mergesort recurrence.

$$T(n) \leq \begin{cases} 0 & \text{if } n = 1 \\ T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + n & \text{otherwise} \end{cases}$$

solve left half
solve right half
merging

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

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

sorting both halves
merging

Proof by Telescoping

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

↑
assumes n is a power of 2

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

sorting both halves
merging

Pf. For $n > 1$:

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

Proof by Induction

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

↑
assumes n is a power of 2

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

sorting both halves
merging

Pf. (by induction on n)

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

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

By the inductive hypothesis
 $\log_2 n = \log_2(2n / 2)$

Analysis of Mergesort Recurrence

Claim. If $T(n)$ satisfies the following recurrence, then $T(n) \leq n \lceil \lg n \rceil$.

↑
 $\log_2 n$

$$T(n) \leq \begin{cases} 0 & \text{if } n = 1 \\ T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + n & \text{otherwise} \end{cases}$$

solve left half
solve right half
merging

Pf. (by induction on n)

- Base case: $n = 1$.
- Define $n_1 = \lceil n/2 \rceil$, $n_2 = \lfloor n/2 \rfloor$.
- Induction step: assume true for $1, 2, \dots, n-1$.

$$\begin{aligned} T(n) &\leq T(n_1) + T(n_2) + n \\ &\leq n_1 \lceil \lg n_1 \rceil + n_2 \lceil \lg n_2 \rceil + n \\ &\leq n \lceil \lg n_2 \rceil + n \\ &\leq n(\lceil \lg n \rceil - 1) + n \\ &= n \lceil \lg n \rceil \end{aligned}$$

$$\begin{aligned} n_2 &= \lfloor n/2 \rfloor \\ &\leq \lfloor 2^{\lceil \lg n \rceil} / 2 \rfloor \\ &= 2^{\lceil \lg n \rceil - 1} \\ &\Rightarrow \lceil \lg n_2 \rceil \leq \lceil \lg n \rceil - 1 \end{aligned}$$

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	Inversions 3-2, 4-2
You	1	3	4	2	5	

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.

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

Divide: $O(1)$.

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
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

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
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

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.

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: $O(n)$

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

Merge and Count

Merge and count step.

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

$i = 6$

↓

3	7	10	14	18	19
---	---	----	----	----	----

↓

2	11	16	17	23	25
---	----	----	----	----	----

two sorted halves

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

auxiliary array

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.

$i = 6$

↓

3	7	10	14	18	19
---	---	----	----	----	----

↓

2	11	16	17	23	25
---	----	----	----	----	----

two sorted halves

2											
---	--	--	--	--	--	--	--	--	--	--	--

auxiliary array

Total: 6

Merge and Count

Merge and count step.

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

$i = 6$

↓

3	7	10	14	18	19
---	---	----	----	----	----

↓

2	11	16	17	23	25
---	----	----	----	----	----

two sorted halves

2											
---	--	--	--	--	--	--	--	--	--	--	--

auxiliary array

Total: 6

Merge and Count

Merge and count step.

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

$i = 6$

↓

3	7	10	14	18	19
---	---	----	----	----	----

↓

2	11	16	17	23	25
---	----	----	----	----	----

two sorted halves

2	3										
---	---	--	--	--	--	--	--	--	--	--	--

auxiliary array

Total: 6

Merge and Count

Merge and count step.

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

$i = 5$

↓

3	7	10	14	18	19
---	---	----	----	----	----

↓

2	11	16	17	23	25
---	----	----	----	----	----

two sorted halves

2	3										
---	---	--	--	--	--	--	--	--	--	--	--

auxiliary array

Total: 6

Merge and Count

Merge and count step.

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

37

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.

38

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.

39

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.

40

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.

41

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.

42

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.

43

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.

44

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.

45

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.

46

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.

47

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.

48

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: $6 + 3 + 2 + 2$

49

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: $6 + 3 + 2 + 2$

50

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: $6 + 3 + 2 + 2$

51

Merge and Count

Merge and count step.

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

first half exhausted

Total: $6 + 3 + 2 + 2$

52

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

53

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

54

Merge and Count

Merge and count step.

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

$i = 0$

3	7	10	14	18	19	↓	2	11	16	17	23	25	two sorted halves
						$i = 0$							
2	3	7	10	11	14	16	17	18	19	23	25	auxiliary array	

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

55

Merge and Count

Merge and count step.

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

$i = 0$

3	7	10	14	18	19	↓	2	11	16	17	23	25	two sorted halves
						$i = 0$							
2	3	7	10	11	14	16	17	18	19	23	25	auxiliary array	

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

56

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
  (rA, A) ← Sort-and-Count(A)
  (rB, B) ← Sort-and-Count(B)
  (r, L) ← Merge-and-Count(A, B)

  return r = rA + rB + r and the sorted list L
}
    
```

57

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.

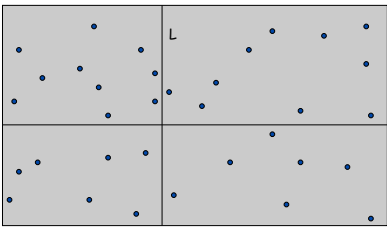
1-D version. $O(n \log n)$ easy if points are on a line.

Assumption. No two points have same x coordinate.
to make presentation cleaner

59

Closest Pair of Points: First Attempt

Divide. Sub-divide region into 4 quadrants.



60

Closest Pair of Points: First Attempt

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

61

Closest Pair of Points

Algorithm.

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

62

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.

63

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 $O(n^2)$
- Return best of 3 solutions.

64

Closest Pair of Points

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

65

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.

66

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.

67

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!

68

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 $\geq 2(\frac{1}{2}\delta)$.

69

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)
   $\delta_1 = \text{Closest-Pair}(\text{left half})$  O(n/2)
   $\delta_2 = \text{Closest-Pair}(\text{right half})$  O(n/2)
   $\delta = \min(\delta_1, \delta_2)$ 
  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  $\delta$ .
}
    
```

70