Objectives

- Divide and conquer
  - Solving recurrence relations
  - Counting inversions

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
- Most common usage
  - Break up problem of size n into two equal parts of size ½n
  - Solve two parts recursively
  - Combine two solutions into overall solution

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

Another Example

- 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*,

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

    $T(2) \le c$

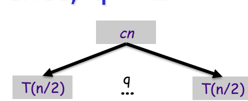
Intuition about running time?

Unrolling Recurrence, q > 2

Unrolling Recurrence, q > 2

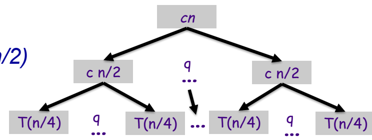
- First level:

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



Unrolling Recurrence, q > 2

- Next level:
  q T(n/4) + $c(n/2)$

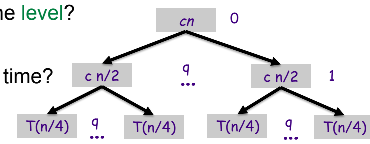


Unrolling Recurrence, q > 2

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: $\log_2 n$

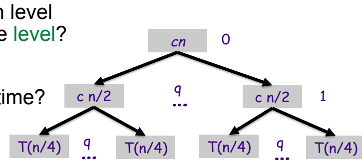
Each level takes $q^k * c * (n/2^k) = (q/2)^k cn$
→ Total work per level is *increasing* as level increases

Unrolling Recurrence, q > 2

How much does each level cost, in terms of the level?

Number of levels?

What is the total run time?



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

Geometric series: → $O(n^{\log_2 q})$ for $q > 2$

Summary

- Use recurrences to analyze 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

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)	Merge Sort	O(n log 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

Problem Context

- Movie site tries to match your movie preferences with others
  - You rank *n* movies
  - Movie site consults database to find people with similar tastes
    - **Collaborative filtering**
- Meta-search tools
  - Do same query on several search engines
  - Synthesize results by looking for similarities and differences in search engines' results rankings

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, …, a_n
  - Movies i and j *inverted* if i < j, but $a_i > a_j$

Discuss this metric

Movies

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

What are the 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, …, 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 trying to improve upon runtime

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)

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 trying to improve upon runtime

Forming 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

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

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: $O(1)$

What are the inversions in blue and green halves?

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

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

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

seems like $\Theta(n^2)$

Combine: ???

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 this example in a bit…

Merge and Count

```
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)
```


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
							6					

two sorted halves

2											

Output array

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

3	7	10	14	18	19		2	11	16	17	23	25
							6					

two sorted halves

2											

Output array

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

3	7	10	14	18	19		2	11	16	17	23	25
							6					

two sorted halves

2	3										

Output array

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

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

Output array: 2 3

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

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

Output array: 2 3 7

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

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

Output array: 2 3 7

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

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

Output array: 2 3 7 10

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

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

Output array: 2 3 7 10

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

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

Output array: 2 3 7 10 11

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

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

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

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

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

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

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

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				

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		

two sorted halves

2	3	7	10	11	14	16	17	18			

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		

two sorted halves

2	3	7	10	11	14	16	17	18			

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		

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

two sorted halves

3	7	10	14	18	19

2	11	16	17	23	25
6	3	2	2	0	

Output array

2	3	7	10	11	14	16	17	18	19	23	

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

two sorted halves

3	7	10	14	18	19

2	11	16	17	23	25
6	3	2	2	0	0

Output array

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

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

two sorted halves

3	7	10	14	18	19

2	11	16	17	23	25
6	3	2	2	0	0

Output array

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

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

Counting Inversions: Implementation

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

Recurrence relation?

```
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) = Sort-and-Count(A)
   (r_B, B) = Sort-and-Count(B)
   (r, L) = Merge-and-Count(A, B)

   return r = r_A + r_B + r and the sorted list L
```


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
   (r_A, A) = Sort-and-Count(A)
   (r_B, B) = Sort-and-Count(B)
   (r_B, L) = Merge-and-Count(A, B)

   return r = r_A + r_B + r and the sorted list L
```


Assignments

- Continue reading Chapter 5
- PS5 due Friday