Wed, Feb 28, 2024

Scientific Computing

Announcements:

→ HW 3 due Fri, March 8

→ Wed March 6: In-class midterm

→ O.H. today, 2pm-3pm in CU 307

→ O.H. Thursday, 10:30am-11:30am on Microsoft Teams

Topic 6 - Divide + Conquer (continued)

Ex #3 Counting Inversions

Input: a list of distinct #s

$L = 3, 19, -7, 2, 1, 6, 0, -10$

An *inversion* is a pair (L_i, L_j) where $i < j$ and $L_i > L_j$.

(In words, a pair of elements where the first is bigger than the second)

Goal: count the # of inversions

This list: $5+6+1+3+2+2+1 = 20$ inversions

__Brute force:__ Check all pairs.
The # of pairs is $\binom{n}{2} = O(n^2)$

__Divide + Conquer:__

$L = \underbrace{3 \quad 19 \quad -7 \quad 2}, \underbrace{1 \quad 6 \quad 0 \quad -10}$

4 inversions fully in green

5 inversions fully in orange

So, 9 inversions <u>within</u> a half. How do we count the inversions where the first element is in the left half and the second is in the right?

Checking all pairs (green, red) works but is basically brute force.

Here's the trick: While we're counting inversions, we'll also <u>sort</u> the lists, which we know takes $O(n \cdot \log(n))$.

$L = \underbrace{3 \quad 19 \quad -7 \quad 2}_{4 \text{ inversions}}, \underbrace{1 \quad 6 \quad 0 \quad -10}_{5 \text{ inversions}}$

$-7 \quad 2 \quad 3 \quad 19 \qquad -10 \quad 0 \quad 1 \quad 6$

Now we need to use this information to count all inversions AND sort the whole list.

We recombine the two sorted lists into one big sorted list, just like mergesort. Can we detect inversions <u>between</u> the two lists?

$L = \underbrace{3 \quad 19 \quad -7 \quad 2}$ | $\underbrace{1 \quad 6 \quad 0 \quad -10}$

4 inversions $\qquad\qquad$ 5 inversions

$\cancel{-7} \quad \cancel{2} \quad \cancel{3} \quad \cancel{19} \qquad\qquad \cancel{-10} \quad \cancel{0} \quad \cancel{1} \quad \cancel{6}$

${}_4\,-10 \quad -7 \quad {}_3\,0 \quad {}_3\,1 \quad 2 \quad 3 \quad {}_1\,6 \quad 19$

Since we picked from purple while there were 4 blues, we know -10 is part of 4 crossing inversions

$4+3+3+1 = 11$ crossing inversions

$$\begin{array}{r} +4 \\ +5 \\ \hline 20 \end{array}$$

$\rightsquigarrow O(n \cdot \log(n))$

<u>Ex #4</u>: Closest Pair of Points (70s)

<u>Input</u>: n points in the xy-plane

$P = \{p_1, p_2, \ldots, p_n\}$

<u>Goal</u>: Find the pair (p_i, p_j) with $i \neq j$ that is the closest to each other.

<u>Brute force</u>: $\binom{n}{2}$ $\quad O(n^2)$

There is a D+C algorithm that is not too difficult that is $O(n \cdot \log(n))$.

Other famous D+C algorithms.

Integer Multiplication

Input: Two n-digit integers x and y

Output: $x \cdot y$

$$\left.\begin{array}{r} 172 \\ \times\ 424 \\ \hline 688 \\ 3440 \\ 68800 \\ \hline 72928 \end{array}\right\} O(n^2)$$

D+C algo:

$$T(n) = 3 \cdot T\left(\tfrac{n}{2}\right) + n$$

$$\Rightarrow T(n) = O\left(n^{\log_2(3)}\right)$$

$$= O\left(n^{1.59\ldots}\right)$$

Lecture 7 - Backtracking

Like D+C: * It's a way to find a guaranteed optimal solution.
* Does so without brute force
* When you code it, you often use recursion

Otherwise it's a very different idea.

Vague premise: Build up solutions bit-by-bit, one part at a time, and give up when a partially built solution is destined to always violate constraints.

Ex #1: Knapsack

Capacity: 10

item	weight	value
1	8	13
2	3	7
3	5	10
4	5	10
5	2	1
6	2	1
7	2	1

With brute force:

Possibilities: $\emptyset$, {1}, {2}, ...

{1,3,4,5,7}

too heavy, and still too heavy if you remove any single item

Backtracking

$C = 10$

	1	2	3	4	5	6	7
w/v	8/13	3/7	5/10	5/10	2/1	2/1	2/1

every possibility on this branch will be to heavy so we "prune" it