Mon, Feb 26, 2024
Scientific Computing

Announcements:

- → HW 3 due Fri, March 8
- → Wed March 6: In-class midterm
- → Thursday Office Hours: iffy

Topic 6 - Divide + Conquer (continued)

What's the runtime of merge sort?
Hard to answer because it's recursive.

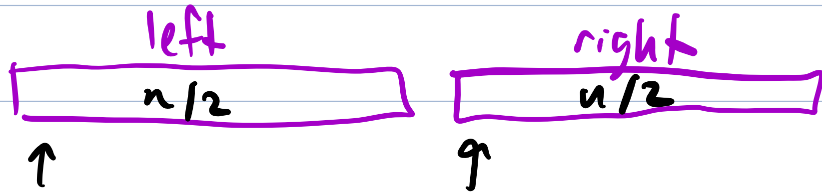
Step 1) Recurrence for the runtime.

Let "$T(n)$" be the runtime when the input has size n.

Steps of merge sort:
Apply to the left half: $T(n/2)$

Apply to the right half: $T(n/2)$

Merge:



n steps

Recurrence: $$T(n) = \underbrace{T\left(\tfrac{n}{2}\right)}_{\text{apply to left}} + \underbrace{T\left(\tfrac{n}{2}\right)}_{\text{apply to right}} + \underbrace{n}_{\text{merge}}$$

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

Step 2: Apply "The Master Theorem" that converts many recurrences into formulas for $T(n)$.

$$T(n) = O(n \cdot \log(n))$$

A few notes

* We split the input in half, not the search space
* Coming up D+C algos is very not obvious, and it only works on some problems.

Ex #2 - The simplest divide + conquer algorithm: "binary search".

* I'm thinking of a # between 1 and 100. You get 7 guesses, and after each guess, I'll tell you "higher" or "lower".

50: lower
25: lower
13: higher
19: lower
15: ✓

Guaranteed to succeed if you always pick the middle number because you throw half of the possibilities every time.

Why 7? $2^7 = 128 \geq 100$

$2^6 = 64 < 100$

$\log_2(n)$

no "2"

Recurrence: $T(n) = 1 + T\left(\frac{n}{2}\right)$

Master Theorem: $T(n) = O(\log(n))$

$O(\log_{10}(n))$ is the same as $O(\log_2(n))$ because they differ by a constant multiple ("change of base formula") and big-O notation ignores constant multiples.

$O(5n) = O(n)$

Binary Search:

Input is a sorted list, and one thing "x"

Question: Is "x" in the list?

Searching an unsorted list for an element is slow: $O(n)$. (Check every spot 1 by 1)

Searching a <u>sorted</u> list for an element is fast using binary search: $O(\log(n))$.

1) check the middle element
   if it $= x$, done
2) If it's $< x$, throw the left half away, repeat on the right half
3) If it's $> x$, throw the right half away, repeat on left half

<u>Ex #3</u> Counting Inversions

Input: a list of <u>distinct</u> #s

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

An <u>inversion</u> 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

<u>Brute force</u>: Check all pairs.
The # of pairs is $\binom{n}{2} = O(n^2)$

<u>Divide + Conquer</u>:

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

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.