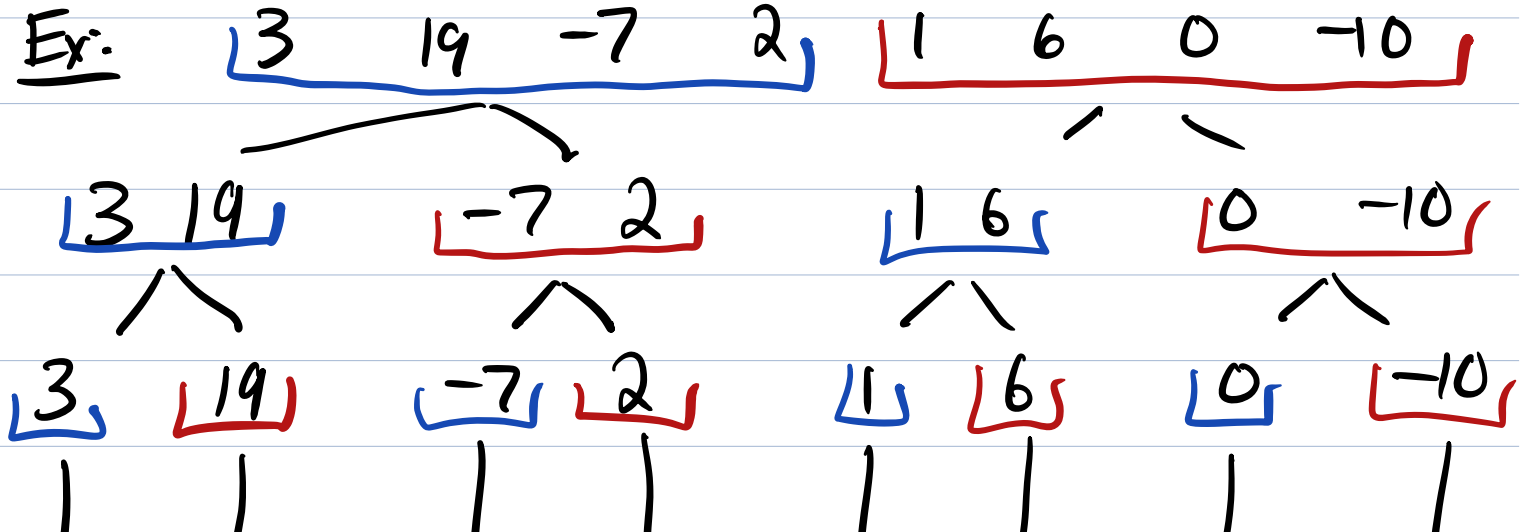
MSSC 6000
Feb 23 2022 - Day 15

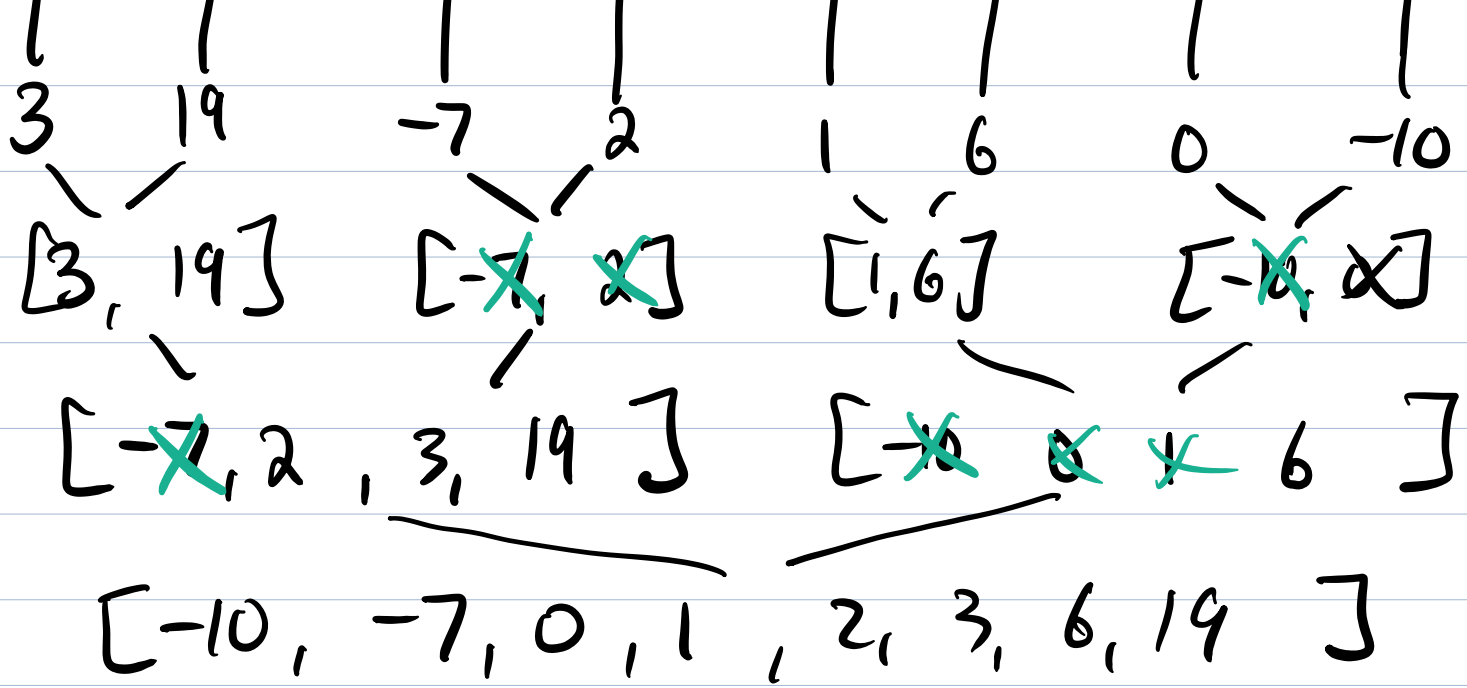
Topic #6 - Divide and Conquer (continued)

* Divide-and-conquer can do it in $O(n \cdot \log(n))$
  (1) Split our input elements in half (or close enough)

  (2) Sort each half (recursively by starting this algo. on each half)

  (3) Combine the two sorted halves into one big sorted list.





Mergesort

Pseudo code

Q = list of #s

```
function merge_sort(Q):
    if |Q|=1:
        return Q
    L = left half of Q
    R = right half of Q
    L = merge_sort(L)
    R = merge_sort(R)

    new_list = [ ]
    while |L|+|R| > 0:
        take L[0] or R[0], whichever
```

now, we have a sorted left and a sorted right

```
        is smaller, remove it, and
        append to new_list
    return new_list
```

What's the runtime? Harder because it's recursive. What we can do is find a recurrence for the runtime.

Define $T(n)$ = the runtime when the input has size n.

Steps:

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

Recurrence: $T(n) = 2 \cdot T(\frac{n}{2}) + n$

There is a theorem called The Master Theorem that tells you how to convert a recurrence to a formula.

Solution: $T(n) = O(n \cdot \log(n))$.

A few notes:

* We are are splitting the input in half, not the search space.
* These D+C algos are usually not obvious. Many times there isn't know.
* The hard part is always the merge.

Ex #2 - The simplest D+C algo. "binary search"

"Guess the #" # between 1 and 100
7 tries, I'll tell you higher or lower.

~~50~~ lower ✓ | ~~25~~ higher | ~~37~~ higher | ~~43~~ lower | ~~40~~ lower

~~39~~ lower | (38)

$2^6 = 64$
$2^7 = 128$

Binary search - you throw away half of your input list each time

Finding an element in a list that is already

sorted.

guess

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

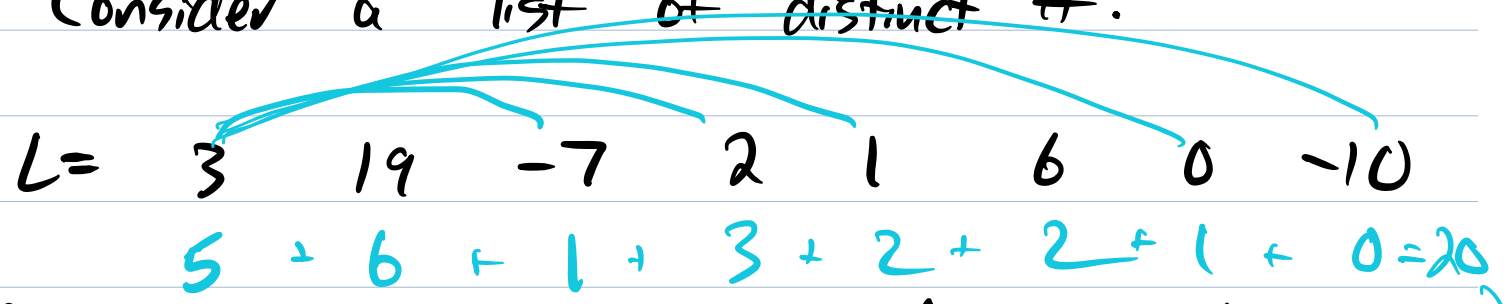
$T(n) = O(\log(n))$

$O(\log_2(n))$

these are equal in big-O notation

Ex #3 - Counting Inversions.

Consider a list of distinct #.



An inversion is a pair (L_i, L_j) where $i < j$ but $L_i > L_j$ (an out-of-order pair)

20 inversions.

Goal: compute the # of inversions in a list of n elements

Obvious Algorithm: Check every pair: $O(n^2)$

Divide + Conquer: $O(n \cdot \log(n))$