Friday, Feb 17, 2023
Lecture # 14
MSSC 6000

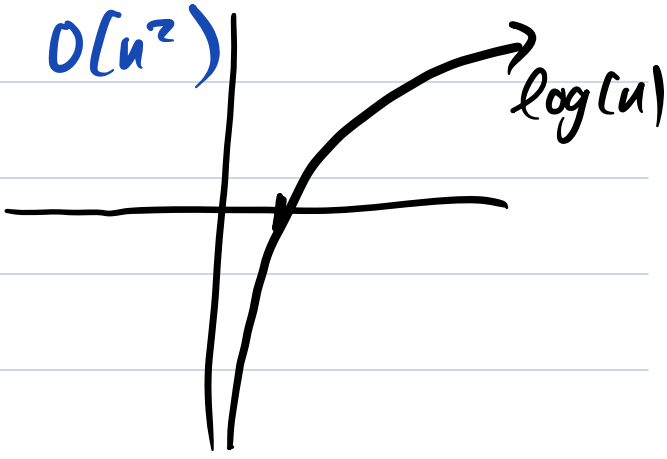
Announcements

* HW 2 due Wednesday, Feb 22, 11:59pm

Topic 6 - Divide and Conquer

Sorting a list (easy)

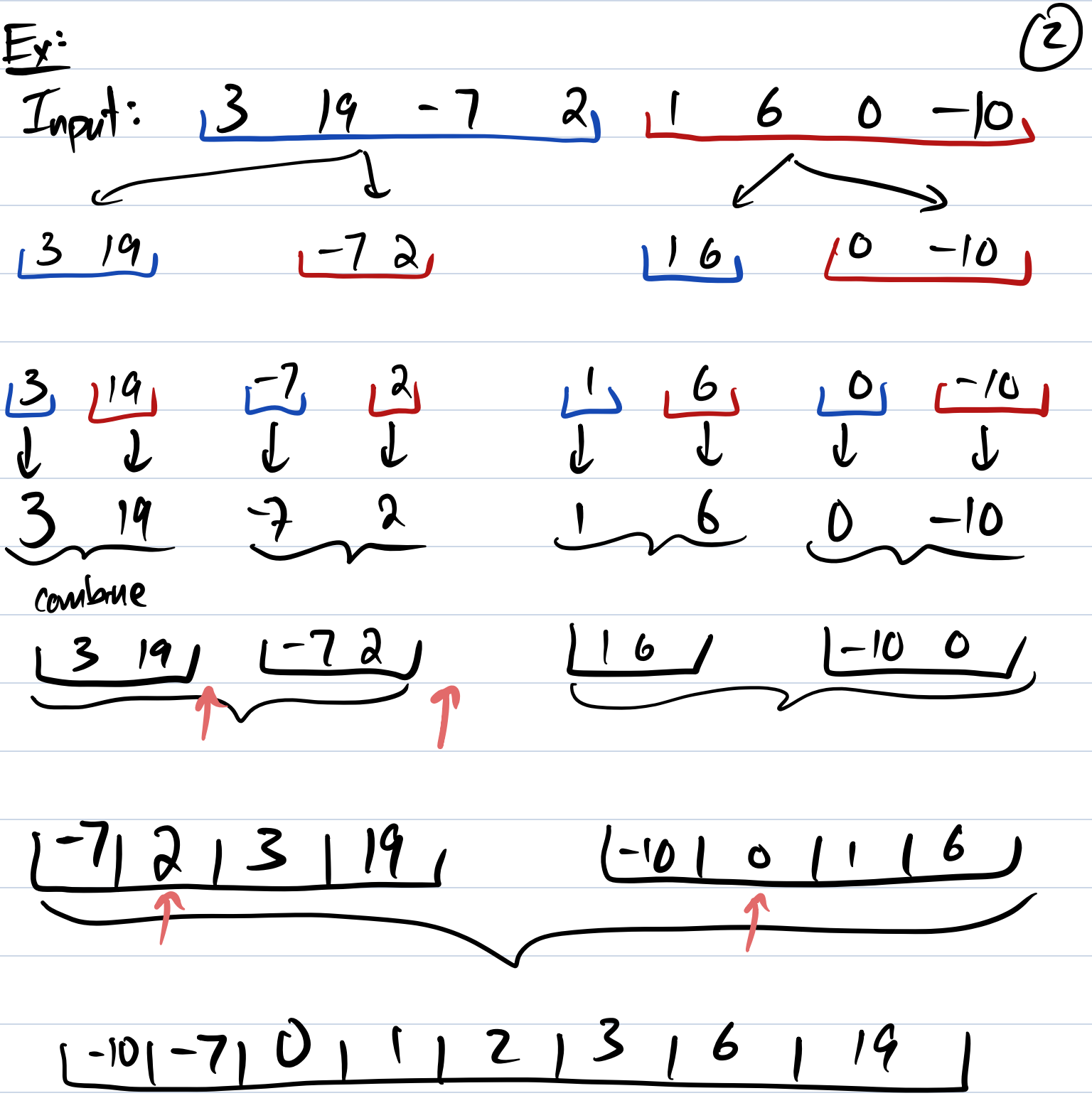
* Divide-and-Conquer can do this in $O(n \cdot \log(n))$. vs $O(n^2)$



$O(n \cdot \log(n))$ is way better than $O(n^2)$.

1) Split the input #s in half
2) Sort each half (recursively, by D+C'ing again)
3) Combine the two sorted halves into one big sorted list.

Ex:



To combine two sorted halves into one big sorted lists only takes $O(n)$ steps.

Pseudocode

```
function merge_sort(Q):        Q = list of #s
    if |Q|=1:
        return Q
    L := left half of Q
    R := right half of Q
    L = merge_sort(L)
    R = merge_sort(R)
    # now L and R are sorted and
    # we want to combine them in a fast way
    new_list := [ ]
    while |L| + |R| > 0 :
        take L[0] or R[0], whichever is
        smaller, remove it, and add to
        new_list
    return new_list
```

":=" defining

What's the runtime? Recursion makes this a hard question. What we can do is: find a recurrence for the runtime

Suppose the runtime is $T(n)$ when the input has size n.

Steps:

applies to the left half	$T(n/2)$
applies to the right half	$T(n/2)$
combines	n

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

There's a theorem in CS called The Master Theorem that tells you how to convert a recurrence into a formula.

In this case: $T(n) = O(n \cdot \log(n))$

↝ Jupyter Notebook sorting demo