MSSC 6000
Feb 23 2022 – Day 14

Topic #6 – Divide and Conquer

How can we find optimal solutions?

(1) If we don't really care about being optimal – greedy algorithms

* (2) Check everything in the search space one-by-one: brute force

(3) Wander around the search space randomly, keeping track of the best solution you've seen so far: random search

(4) Wander around the search space cleverly, keeping track of the best solution you've seen so far: metaheuristics.

* (5) Check or otherwise rule out everything in the search space: divide-and-conquer, backtracking, branch-and-bound.

* (6) Do some clever computations that allow you to score big chunks of the search space all at once: dynamic programming

*Guaranteed to be optimal

→ Demo: list slicing
→ Demo: recursion

"Divide-and-Conquer" is an algorithmic paradigm that is roughly:

(1) Split the input in half
(2) Solves the problem on each half separately. (recursion)
(3) Combines those two answers back into one big answer.

Classic Example: Sorting a list

* You can phrase as an optimization problem.
Search space = all permutations of the list
size = $n!$
Score of a list = the # of pairs that are out of order.
Goal: minimize the score.

* Obviously optimal algorithm: (greedy-ish)
- Find the smallest element, put it first
- Find the next smallest element, put it second, etc.

(insertion sort)

How many steps does this take? (list of size n)

- Finding the k^{th} smallest thing takes n steps (have to search the whole list)
- We have to do this n times.

⇒ $O(n^2)$

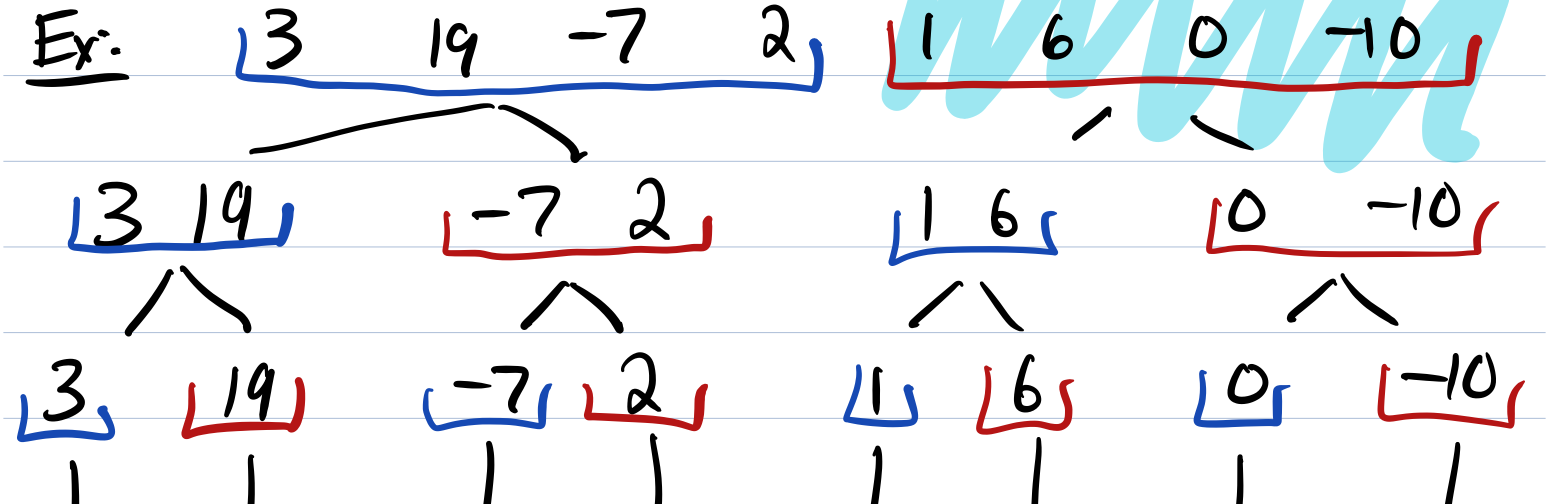
* Divide-and-conquer can do it in $O(n \cdot \log(n))$

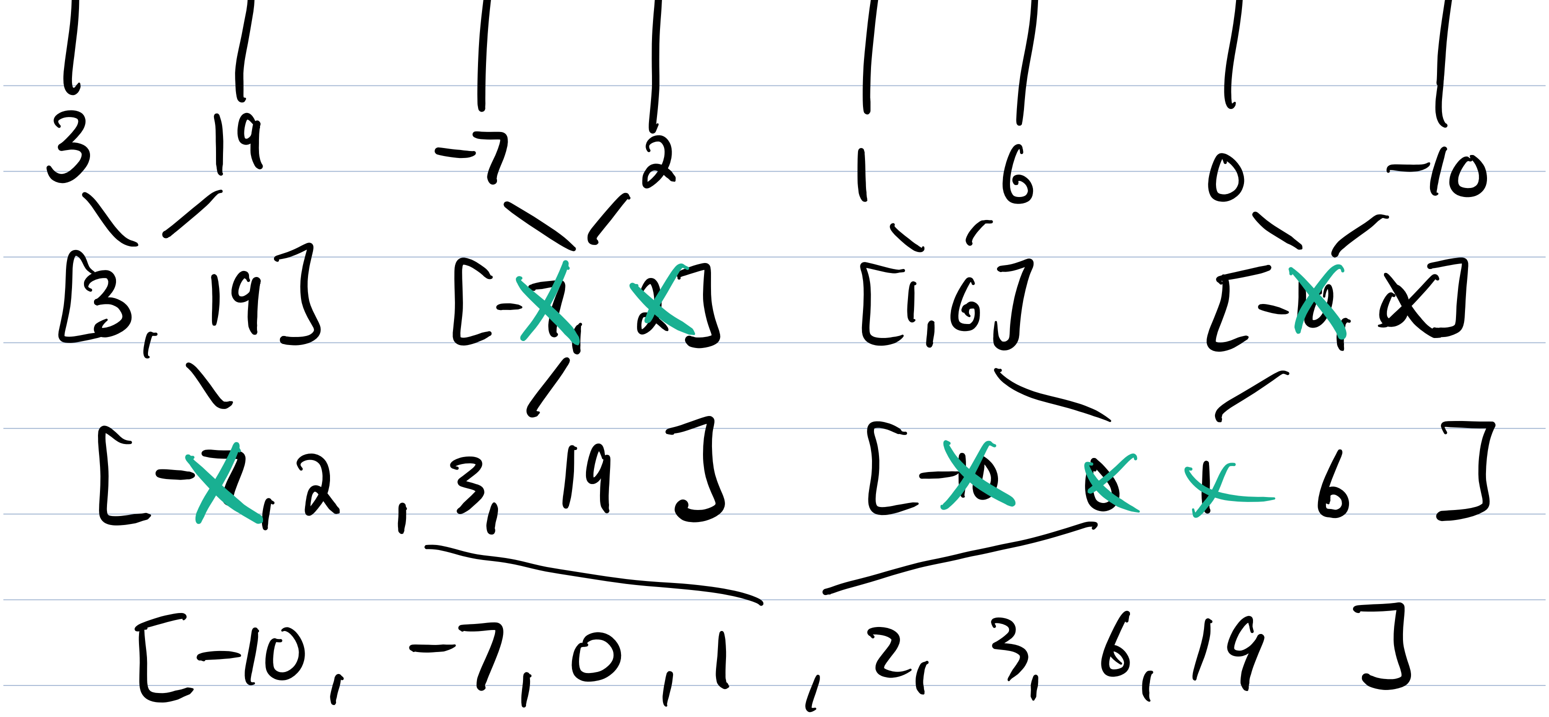
(1) Split your input elements in half (or close enough)

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

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

Ex:





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