CSE 347 Analysis of Algorithms February 4, 2019

Lecture 6 Summary

Instructor: Brendan Juba

1. Problem: finding the closest pair of points in 2d
   (a) Easy to solve in $O(n^2)$ time, but can we solve it faster? – good candidate for divide and conquer.
   (b) In 1d: sort the points, look at neighbors, $O(n \log n)$ time. Does not generalize easily to 2d...
2. Sorting still helps: we split the points in half, recursively solve (get best distance δ) and look for a better pair crossing the middle
   (a) Given a sorted list as input, we can obtain sorted lists for the two halves in $O(n)$ time by filtering the sorted input
3. Geometric structure of combining makes the search across the middle faster
   (a) Members of the pair must lie within δ of the middle
   (b) Boxes of length $\delta/2$ on a side on either side of the middle contain at most one point
   (c) Therefore: if the points δ-close to the middle are sorted bottom-to-top, pairs of distance $< \delta$ lie within ≤ 15 positions in the list – gives $O(n)$-time search for closer pair
4. Since we split into two subproblems of size $n/2$ and take $O(n)$ time to combine, we get the same recurrence as MergesSort with an overall running time of $O(n \log n)$
5. New problem: Knapsack
   (a) Given a set of items with an associated *weight* and *value*, and a maximum weight W, choose a subset with total weight at most W of maximum value.
   (b) Natural greedy approaches fail, no obvious polynomial time algorithm $\Rightarrow$ try dynamic programming
   (c) Key idea: introduce a new parameter to index the subproblems, the remaining capacity of the knapsack