Fractional Knapsack

- $n$ items, each with weight $w_i$ & value $v_i$
- eg. item | 1 | 2 | 3 | 4
  | weight | 5 | 9 | 5 | 1
  | value  | $\$3$ | $\$4$ | $\$3$ | $\$1$

Goal: Maximize value subject to weight constraint

Solution: Sort items according to best value/weight ratio ($v_i/w_i$)
Call sorted list $s_1, s_2, ..., s_n$
Greedy take as much as possible of most beneficial item, $s_1$,
then $s_2$, then $s_3$, etc.... That's it!

Running Time: $O(n \log n)$

Proof sketch: Suppose optimal solution leaves some of $s_i$ and takes some of $s_i$ where $i > 1$. Then exist $\epsilon > 0$ such that $\epsilon \cdot w_i$ is positive and less than the weight of the fraction of $s_i$ taken. Then can substitute $\epsilon \cdot$ fraction of $s_i$ to improve solution.

If $s_1$ doesn't fill knapsack, repeat arg on $s_2, s_3, ... s_n$ with smaller knapsack

Note: Argument totally fails in non-fractional case (which is NP-hard), similar to difference between linear-programming and integer linear-programming.
Computational Geometry

Convex Hull

Given n points in 2-D, output list of points which form the "hull".

"Convex hull" = the smallest convex polygon containing all the points.

E.g., Hull is formed by P1, P2, P4, P8, P10, P11, P5.

Q: What's the best running time we can hope to get?
A: O(n log n). Any convex hull algorithm can be used to sort points in 1-D. Just plot on line, add extra point, and find convex hull of that.

How to compute Hull? Many methods. We'll use Graham's Scan.

Intuition: like fractional knapsack, start by sorting points in some order, then greedily considering them in turn.

Idea: Pick leftmost point (bottom-left if more than one). Call it p1.
Sort all other points according to polar angle with p1.

Consider points P1, P2, P3... in turn.
When "left turn" is made to connect p1, push p1 onto a stack.
- When "right turn" is made to connect $p_i$, pop off stack
  until $p_i$ connects to top element with a left turn
- When finished, stack contains the convex hull

Analysis: In total, each vertex is pushed on the stack at most once, and popped from stack at most once.
Thus scanning is $O(n)$. The initial sorting time of $O(n \log n)$ dominates.