1. Suppose that in the 0/1-knapsack problem, the order of the items when sorted by increasing weight (i.e., \(w_i\)'s) is the same as when they are sorted by decreasing value (i.e., \(v_i\)'s). Describe, in words, an \(O(n \log n)\)-time greedy algorithm to compute a subset of items of maximum value whose total weight is at most the knapsack capacity (i.e., \(W\)).

Use the 2-step proof method, i.e., state and establish carefully the greedy choice and optimal substructure properties, to prove that your algorithm is correct. A careful, well-worded proof is expected.

2. Imagine that you are planning a road trip from Boston to Seattle along Interstate 90. Your electric car has a range of \(m\) miles on a full charge. You have a map that shows the locations of all charging stations along your route, from east to west. (There’s actually an app for this.) So that you do not get stranded, assume that no two successive stations are more than \(m\) miles apart and that you start your trip with a full charge. The goal is to minimize the number of charging stops that you need to make.

Describe briefly, in words, a greedy algorithm for this problem and prove it correct using the 2-step method, i.e., state and establish the greedy choice and optimal substructure properties. A careful, well-worded proof is expected.

3. Let \(G\) be a connected, undirected, edge-weighted graph. Prove that \(G\) has a unique MST if all the edge weights are distinct.

It is tempting to try and prove this result by simply observing how Kruskal’s algorithm would operate on this graph. Why is this not enough? Instead prove this result “non-algorithmically”, by using the notion of a cut.

4. Recall the definition of the exchange property for a matroid \(M = (S, I)\); call this “Def. 1”. Consider the following alternative definition; call this “Def. 2”:

“If \(A, B \in I\) and \(|B| = |A| + 1\), then there is some \(x \in B - A\) such that \(A \cup \{x\} \in I\).”

Prove that if \(M\) satisfies Def. 2 then it satisfies Def. 1, and vice versa.

5. In class it was shown that a red-black tree with \(n\) internal nodes has height at most \(2 \log(n + 1)\). Show that this bound is asymptotically tight, i.e., describe a red-black tree on \(n\) nodes and height \(h\) for which the ratio \(h/(2 \log(n + 1))\) approaches 1 as \(n\) approaches infinity. (The tree is not unique.)

Over \(\iff\)
6. Become familiar with insertion and deletion in red-black trees. You do not have to memorize the various cases, but you should be comfortable simulating updates on examples. Use the tree in Fig. 13.1(c) (page 310) for practice and try out updates that generate the various cases. For example, inserting 40 generates (the mirror image of) Case 1, deleting 28 generates Case 4, etc.

7. Let $\otimes$ be a constant-time-computable associative binary operator (e.g., addition or multiplication over the reals). Let $T$ be a red-black tree on $n$ keys. Besides a key, each internal node, $x$, has a real-valued attribute $a(x)$. Additionally, let $f(x)$ be an auxiliary field at $x$ which is defined as $f(x) = a(x_1) \otimes a(x_2) \otimes \cdots \otimes a(x_m)$, where $x_1, x_2, \ldots, x_m$ is the in-order listing of the nodes in $x$’s subtree (inclusive of $x$). The field $f(\cdot)$ is not of interest for external nodes.

Use the General Augmentation Theorem (GAT) to show that $f(\cdot)$ can be maintained at the internal nodes of $T$ during insertions and deletions, without affecting the logarithmic time bounds for those operations. You must use the GAT instead of establishing the result from scratch.

Use your result above to argue that the size field in an order-statistic tree can be maintained efficiently, by defining $a(x)$ and $\otimes$ appropriately.

8. Let $S$ be a set of $n$ horizontal line segments in the plane (each specified by its two endpoints) and let $P$ be a set of $n$ points in the plane. We are interested in determining for each point $p \in P$ the first line segment of $S$ above (resp. below) $p$ that is hit by a vertical ray emanating upwards (resp. downwards) from $p$. Describe a sweepline algorithm for this problem which runs in $O(n \log n)$ time in the worst case.

Your answer should include (i) a description of the main ideas and data structures underlying your solution (from which the correctness of your solution should be evident), (ii) pseudocode, and (iii) an analysis of the running time.

You may use any data structure studied in class as a black-box, so long as you do not modify it.