1. Suppose that in the 0/1-knapsack problem, the order of the items if 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). Assume that you are given an unsorted set of items.

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—a Tesla (why not?:-)—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.

Note: One way to prove this is to observe how Kruskal’s algorithm would operate on this graph. A more instructive way is to 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.)

\[\text{Over} \implies\]
Show the requested information before and after each transformation given in Figs. 13.5–13.7. You may show the information in the figures themselves or in a table.
This problem is to help reinforce your understanding of the various cases that arise during insertion and deletion in red-black trees. However, you do not have to memorize the various cases for the test.

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 \( \mathcal{R} \) be a set of \( n \) rectangles in the \( xy \)-plane, with sides parallel to the coordinate axes, and let \( \mathcal{P} \) be a set of \( n \) points in the plane. Given \( \mathcal{R} \) and \( \mathcal{P} \), our goal is to decide for each point in \( \mathcal{P} \) whether or not the point lies inside at least one rectangle of \( \mathcal{R} \), i.e., the output should simply be “IN” if the point lies in the interior or on the boundary of at least one rectangle of \( \mathcal{R} \), and “OUT” otherwise. (Think of the rectangles as windows on a computer screen and the points as mouse-clicks, for instance.) Give a sweepline algorithm for this problem which runs in \( O(n \log n) \) time.
Your answer should include: (a) a brief description of the main ideas behind the algorithm, including data structures used, (b) pseudocode, and (c) an analysis of the running time.
Assume that each rectangle \( R_i \in \mathcal{R} \) is specified by its lower-left corner \((\ell_i, b_i)\) and upper-right corner \((r_i, t_i)\). For simplicity, assume that all \( x \)-coordinates and all \( y \)-coordinates in \( \mathcal{R} \cup \mathcal{P} \) are distinct.
You may use any data structure studied in class as a black-box, so long as you do not modify it.