1. In a BST, what is value invariant - that is, what is true about the value of a parent node, compared to the values stored in its children and descendants?

Solution: The left child (and all of its descendants) must have values $\leq$ the value of the node. Likewise, the right subtree must have values that are $\geq$.

2. In a max heap, what is value invariant?

Solution: The value of each node must be $\geq$ than the value of both of its parents.

3. In a heap, how is the data actually physically stored (as opposed to how we think about it)?

Solution: It is stored in an array.

4. If we have a max heap, and we want to remove the minimum, how would we do this? Why is this a bad idea?

Solution: We have to do a brute-force search of all of the leaves. This is $O(n)$.

Additional note, not required: Also - if we remove a value and it’s not the last one in the array, then we have to re-arrange the values to maintain the shape invariant (and it’s not clear how we would do this, or how much it might cost).

5. Assume that I already know what the “bubble up” and “bubble down” operations do in a heap. Now, explain (in English, or in some very simply pseudocode) how to perform each of the following operations: insert
**Solution:** Adjust the new value at the very end of the array. Then bubble it up to its proper position.

**delete maximum value**

**Solution:** Remove the root node, and move the last element into its place (or, if you prefer to think of it this way, “swap” the first and last values). Then, bubble down from the root node, to put the formerly-last value into its proper place.

**build-max-heap** (starting with an array of random data)

**Solution:** Starting at the last internal node (and then iterating up to the root), bubble each value down.