1. Suppose we want to maintain a dynamic set of numbers, subject to the following operations:

- **Insert(x)**: Add x to the set. (Assume x is not already in the set.)
- **PrintDeleteBetween(a, b)**: Print every element x in the range a \(\leq\) x \(\leq\) b in increasing order, and then delete those elements from the set.

For example, if the current set is \{1, 5, 3, 4, 8\}, then

- **PrintDeleteBetween(4, 6)** prints the numbers 4 and 5 and changes the set to \{1, 3, 8\}.
- **PrintDeleteBetween(6, 7)** prints nothing and does not change the set.
- **PrintDeleteBetween(0, 10)** prints the sequence 1, 3, 4, 5, 8 and deletes everything.

Describe a data structure that supports both operations in \(O(\log n)\) amortized time, where \(n\) is the current number of elements in the set.

[Hint: As warmup, argue that the obvious implementation of **PrintDeleteBetween**—while the successor of a is less than or equal to b, print it and delete it—runs in \(O(\log N)\) amortized time, where \(N\) is the maximum number of elements that are ever in the set.]

2. Describe a data structure that stores a set of numbers (which is initially empty) and supports the following operations in \(O(1)\) amortized time:

- **Insert(x)**: Insert x into the set. (You can safely assume that x is not already in the set.)
- **FindMin**: Return the smallest element of the set (or **null** if the set is empty).
- **DeleteBottomHalf**: Remove the smallest \([n/2]\) elements the set. (That's smallest by value, not smallest by insertion time.)

3. Consider the following solution for the union-find problem, called **union-by-weight**. Each set leader \(\overline{x}\) stores the number of elements of its set in the field \(weight(\overline{x})\). Whenever we **Union** two sets, the leader of the smaller set becomes a new child of the leader of the larger set (breaking ties arbitrarily).

```
\[
\text{MakeSet}(x):
\begin{align*}
\text{parent}(x) & \leftarrow x \\
\text{weight}(x) & \leftarrow 1
\end{align*}
\]

\[
\text{Find}(x):
\begin{align*}
\text{while } x \neq \text{parent}(x) \\
\quad x & \leftarrow \text{parent}(x) \\
\text{return } x
\end{align*}
\]

\[
\text{Union}(x, y):
\begin{align*}
\overline{x} & \leftarrow \text{Find}(x) \\
\overline{y} & \leftarrow \text{Find}(y) \\
\text{if } \text{weight}(\overline{x}) > \text{weight}(\overline{y}) \\
\quad \text{parent}(\overline{y}) & \leftarrow \overline{x} \\
\quad \text{weight}(\overline{x}) & \leftarrow \text{weight}(\overline{x}) + \text{weight}(\overline{y}) \\
\text{else} \\
\quad \text{parent}(\overline{x}) & \leftarrow \overline{y} \\
\quad \text{weight}(\overline{x}) & \leftarrow \text{weight}(\overline{x}) + \text{weight}(\overline{y})
\end{align*}
\]
```

Prove that if we always use union-by-weight, the worst-case running time of **Find(x)** is \(O(\log n)\), where \(n\) is the cardinality of the set containing \(x\).