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.)
   - PRINT&DELETEBETWEEN(a, b): Print every element x in the range $a \le x \le b$ in increasing order, and then delete those elements from the set.

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

   - PRINT&DELETEBETWEEN($4, 6$) prints the numbers 4 and 5 and changes the set to $\{1, 3, 8\}$.
   - PRINT&DELETEBETWEEN($6, 7$) prints nothing and does not change the set.
   - PRINT&DELETEBETWEEN($0, 10$) 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 PRINT&DELETEBETWEEN*—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 $\lceil n/2 \rceil$ 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 $\bar{x}$ stores the number of elements of its set in the field *weight*($\bar{x}$). 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).

```
MAKESET(x):
  parent(x) ← x
  weight(x) ← 1

FIND(x):
  while x ≠ parent(x)
      x ← parent(x)
  return x

UNION(x, y)
  x̄ ← FIND(x)
  ȳ ← FIND(y)
  if weight(x̄) > weight(ȳ)
      parent(ȳ) ← x̄
      weight(x̄) ← weight(x̄) + weight(ȳ)
  else
      parent(x̄) ← ȳ
      weight(x̄) ← weight(x̄) + weight(ȳ)
```

   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.