Question 1

Let $S = (x_1, x_2, \ldots, x_n)$ be the given sequence of numbers and let $T$ be a complete binary tree over $S$. Then every node of $v$ represents a subsequence $S_v = (x_{i_v}, x_{i_v+1}, \ldots, x_{j_v})$ of $S$ corresponding to its descendant leaves. Our range minima data structure consists of three parts:

- The tree $T$,
- An LCA data structure $D$ for $T$, and
- For every node $v$ of $T$, two sequences $L_v$ and $R_v$ defined as follows: Both sequences have the same length as $S_v$, that is, $L_v = (l^v_{i_v}, l^v_{i_v+1}, \ldots, l^v_{j_v})$ and $R_v = (r^v_{i_v}, r^v_{i_v+1}, \ldots, r^v_{j_v})$. For $i_v \leq h \leq j_v$, $l^v_h = \min_{h \leq h' \leq j_v} x_{h'}$ and $r^v_h = \min_{i_v \leq h' \leq h} x_{h'}$.

The first two parts of the data structure clearly use linear space because $T$ stores $n$ elements and uses $O(n)$ space to do so, and $D$ has linear size in the size of $T$, that is, also has size $O(n)$. For the third part, observe that the total size of the lists $L_v$ and $R_v$, for all $v \in T$, is twice the size of the lists $S_v$, for all $v \in T$. The latter have size $O(n \lg n)$ because every element $x_i$ belongs to exactly the lists $S_v$ such that $v$ is an ancestor of $x_i$ in $T$; since $T$ is perfectly balanced, there are only $O(\lg n)$ such ancestors for each $x_i$, that is, each element $x_i$ is stored in $O(\lg n)$ lists.

We can build $T$ in linear time: Scan the list $S$ and make pairs of consecutive elements children of the same parent. This gives us a list $S_1$ of $\lceil n/2 \rceil$ nodes that are parents of leaves. To create the list $S_2$ of their parents, of which there are $\lceil n/4 \rceil$, we apply the same procedure to $S_1$. We repeat this $h := \lfloor \lg n \rfloor$ times until the list $S_h$ has size $|S_h| = 1$. The LCA data structure $D$ for $T$ can be built in linear time. To construct the lists $L_v$ and $R_v$, we apply a post-order traversal of $T$, that is, we process every node after its children. During this traversal, we do in fact also compute $S_v$. For a leaf $x_i$, $S_{x_i} = L_{x_i} = R_{x_i} = (x_i)$. For a node $v$ with children $u$ and $w$, we have $S_v = S_u \circ S_w$, where $\circ$ is the concatenation operation. Even if we represent these lists as arrays,
then creating $S_v$ requires allocating an arrays of size $|S_v|$ and copying the elements in $S_u$ and $S_v$ into this array. This takes $O(|S_v|)$ time. $L_v$ can be computed from $S_v$ in $O(|S_v|)$ time by scanning $S_v$ from right to left, maintaining the minimum element seen so far, and storing this element in the slot of $L_v$ corresponding to the current element of $S_v$. To construct $R_v$, we scan $S_v$ from left to right. In total, $S_v$, $L_v$, and $R_v$ can be constructed in $O(|S_v|)$ time, given the lists associated with $v$’s children. Since we already argued that the total size of the lists $S_v$, for all $v \in T$, is $O(n \lg n)$, the construction of these lists for all nodes of $T$ thus takes $O(n \lg n)$ time.

Now, let $i \leq j$ be a pair of indices for which we want to answer a range minimum query. If $i = j$, we simply report $x_i$, which is obviously correct and obviously takes constant time. If $i < j$, we use $D$ to find the LCA $v$ of $x_i$ and $x_j$ in $T$ in constant time. Let $u$ and $w$ be $v$’s children. Then, since $v$ is the LCA of $x_i$ and $x_j$, $x_i \in T_u$ and $x_j \in T_w$. In particular, $i_u \leq i \leq j_u$, $i_w \leq j \leq j_w$, and $i_w = j_u + 1$. Thus, $\text{RangeMin}(i, j) = \min(\text{RangeMin}(i, j_u), \text{RangeMin}(i_w, j)) = \min(l^u_i, r^w_j)$. Given $v$, $u$ and $w$ can be found in constant time. Looking up $l^u_i$ and $r^w_j$ in $L_u$ and $R_w$ also takes constant time. Thus, $\text{RangeMin}(i, j)$ can be computed in constant time.

**Question 2**

An in-order traversal of a tree $T$ with root $r$ reflects the order in which the nodes of $T$ are visited by depth-first search, including repeated visits caused by backtracking from children to parents. Formally, this traversal defines a node sequence $S = S_r$, which is defined recursively as follows: For a node $v$ with children $w_1, w_2, \ldots, w_k$, we have $S_v = \langle v \rangle$ if $k = 0$, that is, $v$ is a leaf. Otherwise, $S_v = \langle v \rangle \circ S_{w_1} \circ \langle v \rangle \circ S_{w_2} \circ \cdots \circ \langle v \rangle \circ S_{w_k} \circ \langle v \rangle$. For a node $v$ in $T$, its depth is the number of edges on the path from $r$ to $v$. Let $D$ be a sequence of length $|S|$ whose $i$th entry is the depth of the $i$th node in $S$. Our data structure to answer LCA queries on $T$ has four parts:

- Every node $v \in T$ stores an index $i_v$ such that the $i_v$th entry in $S$ is $v$. (There may be more than one such index. It is irrelevant which one $v$ stores.)
- The sequence $S$.
- The sequence $D$.
- A range-minima data structure $M$ for $D$.

Clearly, each of these parts uses $O(n)$ space if $S$ has size $O(n)$. If this is true, it also follows that the DFS traversal that constructs $S$ (and which can be used to compute the depth of each node, that is, to construct $D$ along with $S$) takes $O(n)$ time. The construction of $M$ from $D$ takes $O(|D|) = O(n)$ time in this case. So we need to argue that $|S| \in O(n)$. We observe that every node $v \in T$ occurs $d_v + 1$ times in $S$, where $d_v$ is the number of children of $v$. Thus, every
non-root node contributes one occurrence of itself and one occurrence of its parent to $S$. This shows that, if $T$ has $n$ nodes, then $|S| = 2n - 1$.

Now, to answer an LCA query with nodes $u$ and $w$, we ask a range-minimum query on $M$ with indices $i_u < i_w$ as arguments (if $i_u > i_w$, we simply swap $u$ and $w$). This returns an index $j$ such that $D[j]$ is the smallest entry in $D[i_u, i_w]$. We return $S[j]$ as the query answer. This procedure clearly takes constant time. We have to argue that it gives the right answer, that is, that the node with minimum depth in $S[i_u, i_w]$ is indeed the LCA of $u$ and $w$.

Let $v$ be the LCA of $u$ and $w$. Then every occurrence of $u$ and $w$ in $S$ belongs to $S_v$. Since all nodes in $S_v$ other than $v$ have a depth greater than $v$’s, this implies that all nodes in $S[i_u, i_w]$ have depth no less than $v$’s depth, no matter which occurrences of $u$ and $w$ we choose to define $i_u$ and $i_w$. Thus, to prove that the query procedure returns $v$, it suffices to show that $v$ occurs at least once in $S[i_u, i_w]$. If $v = u$ or $v = w$, this is true because $S[i_u] = u$ and $S[i_w] = w$, so assume $u \neq v \neq w$. Then let $u'$ and $w'$ be the children of $v$ such that $u$ is a descendant of $u'$ and $w$ is a descendant of $w'$. In this case, every occurrence of $u$ belongs to $S_{u'}$, every occurrence of $w$ belongs to $S_{w'}$, and there is at least one occurrence of $v$ in $S_v$ between $S_{u'}$ and $S_{w'}$. Thus, $v \in S[i_u, i_w]$. 