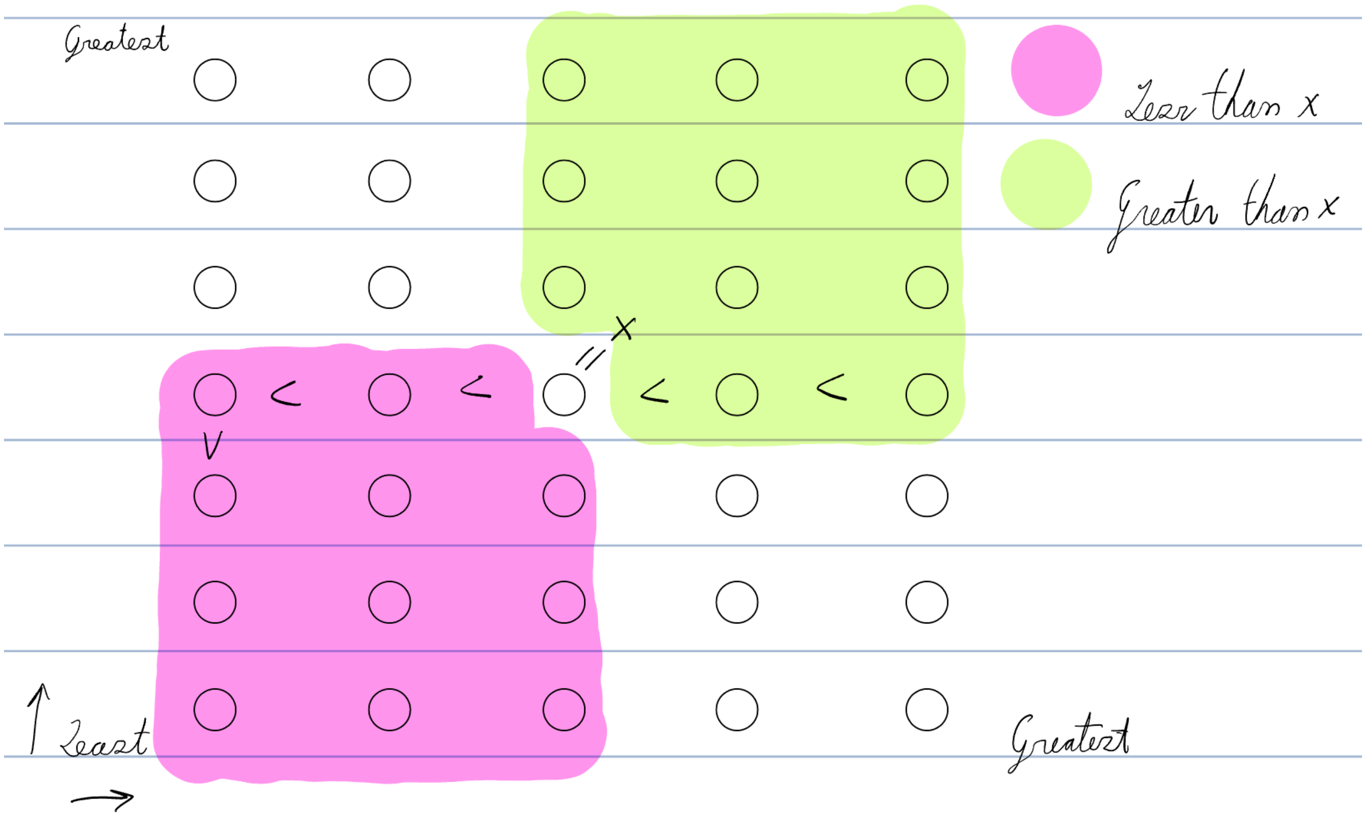
Modified Select, Half Majority, Fast BST

1A

There are at least $\frac{4n}{14}$ elements guaranteed to be $< x$ and $> x$. As seen in the diagram below, each column represents a group of seven elements in sorted order from least to greatest from bottom to top, and the groups are sorted from least to greatest median from left to right. The bottom left group is guaranteed to be $< x$ because the medians of those groups are known to be less than x, and the elements within each group less than the median must also be less than x. They account for $4 \cdot \frac{n}{7} \cdot \frac{1}{2} = \frac{4n}{14}$ elements. There are 4 elements in each group, $\frac{n}{7}$ groups total, and half the groups are to the left containing x. The same argument applies to the elements $> x$ seen in the top-right. We cannot be certain that the elements not highlighted are either $< x$ or $> x$.



1B

Worst-case partition is $\frac{4n}{14}$ elements in one array and $\frac{10n}{14}$ on the other array. Because we know that whatever median of median x we pick, there are at least $\frac{4n}{14}$ elements for which we know how they compare to x, $n - \frac{4n}{14} = \frac{10n}{14}$ elements must be in the other array.

1C

$$T(n) = T\left(\frac{n}{7}\right) + T\left(\frac{10n}{14}\right) + \Theta(n)$$
$$= T(n) = T\left(\frac{12n}{14}\right) + \Theta(n)$$

It takes linear time to find the initial $\frac{n}{7}$ medians, and to find medians recursively takes $T(\frac{n}{7})$ since there are groups of 7, and, in the worst case, the array needed to recurse on will take $T(\frac{10n}{14})$, since that is the maximum number of items one of the arrays can contain (shown in `1B`).

1D

This recurrence solves in linear time because $\frac{12n}{14} < n$. The number of items to recurse on decreases with every iteration. Solving this recursion will result in n multiplied by some geometric sum with a ratio < 1, which is $\Theta(1)$. So, it follows that the recursion is linear.

2

Use `Select` to get the elements with rank $\frac{n}{4}$, $\frac{2n}{4} = \frac{n}{2}$, and $\frac{3n}{4}$. These ranks were determined by dividing the array into quarters and finding the inner posts that delimit the quarters. If there is an element that occurs more than $\lceil \frac{n}{4} \rceil$, it will have a rank that overlaps with one of the previously mentioned ranks, regardless of its value. Then, iterate over the array and count the frequencies of the return values from the three `Select` calls. Then, check if either occurs more than $\lceil \frac{n}{4} \rceil$. This will take $T(n) = 3\Theta(n) + 3\Theta(n) + \Theta(1) = 6\Theta(n)$. Three calls of `Select` run in linear time, counting the frequency of three items takes a linear time for each, and finally checking if either of those items occur the desired amount takes constant time.

3

It is impossible to develop such an algorithm. It is known that the lower bound for any comparison-based sorting algorithm is $\Theta(nlgn)$. This can be demonstrated with a decision tree. Such a data structure would allow sorting in $\Theta(n\sqrt{lgn})$, which is impossible. The algorithm would involve adding all items to the hypothetical BST and then doing an in-order tree traversal, resulting in a faster sorting algorithm than the known lower bound. There will be n inserts that each take $\Theta\sqrt{lgn}$, which is $\Theta(n\sqrt{lgn})$. Then, an in-order traversal, which takes linear time, gives $\Theta(n\sqrt{lgn}+n)=\Theta(n\sqrt{lgn})$ overall. An in-order traversal involves visiting the nodes to the left of the root, then the root, and then the nodes to the right of the root. Since a BST has the property that all elements to the left are less than the root and elements to the right are greater, this traversal traverses the list in sorted order.

#homework