A decision tree for the problem can be as follows:

Here, each internal node represents a key-value comparison. Every internal-node has three children, for =, <, and >. A NO-node represents X not occurring in A. The = path immediately leads to the solution and therefore stops. Let us call this a solution-node.

- Since all values are distinct, and the key can be any value, the solutions nodes are atleast n.
- Each solution-node has a corresponding internal-node.
- Also, each internal-node has only two edges that hold other internal nodes.
- Given height h, the maximum number of internal-nodes you can pack into the tree is \(2^0 + 2^1 + 2^2 + \ldots + 2^h = 2^{h+1} - 1\).
• Clearly

\[ 2^{h+1} - 1 \geq n + 1 \]

which gives us

\[ h \geq (\log_2 n + 2) - 1 \]

• A search, in the worst case, should traverse the height of the decision tree. Therefore, search has a lower bound of \( \Omega(\log n) \).

**Grading**

Upto 6 points:
- Using binary search algorithm as the argument for proving the lower bound.

The solutions that talk about binary search fail to point out that the \( \log n \) lower bound comes from having \( O(n) \) distinct elements.

Using such binary search argument it is possible to show that search in an array with all elements having same value (for which we can come up with a \( O(1) \) algorithm, assuming we know that information before hand) has a lower bound of \( \log n \).

- Using some form of reduction, in the right direction.

From 8 to 14 points:
- Using height of the decision tree for lower bound, but a mistake in the argument.

From 16 to 20 points:
- Mostly precise proof about height of the decision tree.