Lecture 5: Binary Search Trees and std::set

Activity 1: Imagine you're designing a system for a stock exchange, like for trading Apple stock. Traders are constantly submitting buy and sell orders—buyers say, 'I'll buy 100 shares at $150,' and sellers say, 'I'll sell 50 shares at $150.' The system maintains an order book: a list of all buy orders sorted by price, highest to lowest, so the best offers are matched first. Orders arrive in real-time, and the system needs to quickly search for matches—like finding a buy order at $150 to match a sell order—print the book in price order, and find the best buy price. This happens thousands of times a second, so efficiency is critical!

To manage the buy order book, which data structure should we use? The system needs to handle a stream of new orders, search for prices to match trades, print orders in sorted order, and find the best price—all as fast as possible.

Discuss with your peers and vote for the best option:

A) **Unsorted Vector**: Add orders with push_back, search with std::find, sort when printing.

B) **Sorted Vector**: Add orders, re-sort after each addition, search with std::binary_search.

C) **std::set**: Add orders with insert, search with find, print in order, get best price with begin().

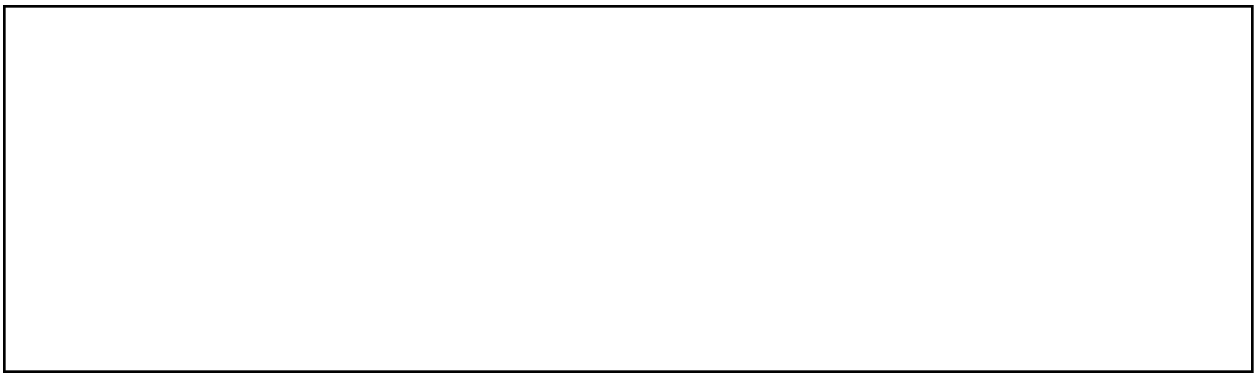
D) **Array**: Fixed-size array, manually manage sorting and searching.

E) More than one option works equally well.

Fill in the blank: Binary Search Tree (BST) is a Binary Tree, where each node satisfies the property:_____

Height of the tree - Length of the longest path from the root to a leaf node.

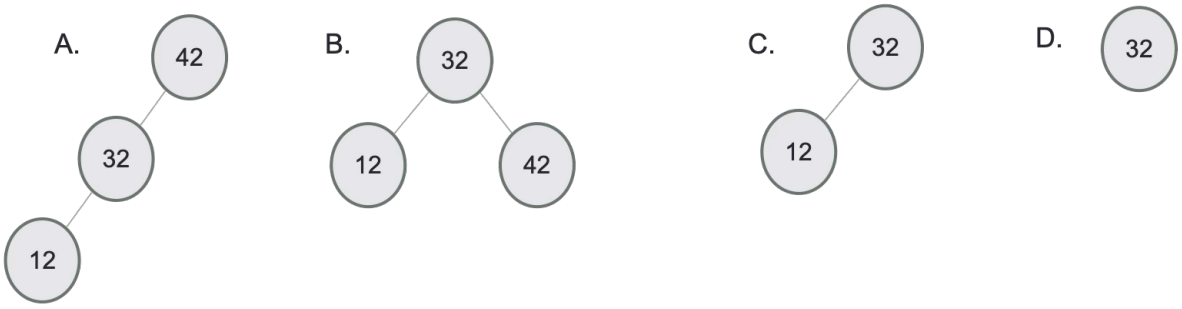
Activity 3: Draw all possible BSTs that contain the keys: 12, 32, 41, 42, 45. Write the height in each case



```
class BST{
  public:
      int getHeight() const; // Implement this function recursively
      //Other member functions
  private:
      struct TreeNode {
                TreeNode* left;
                TreeNode* right;
                TreeNode* parent;
                int const data;
                TreeNode(int d) : data(d) {
                      left = right = parent = nullptr;
                }
      };

      TreeNode* root;
}
```

Activity 4: Describe a recursive approach to calculate the height of a BST.
Think about an empty tree (height = -1), a one-node tree (height = 0)
Use the examples below to figure out the recursive case.



Implement the getHeight() function recursively.