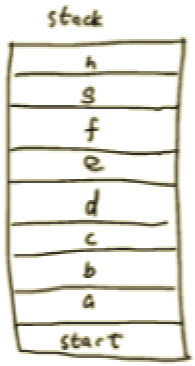
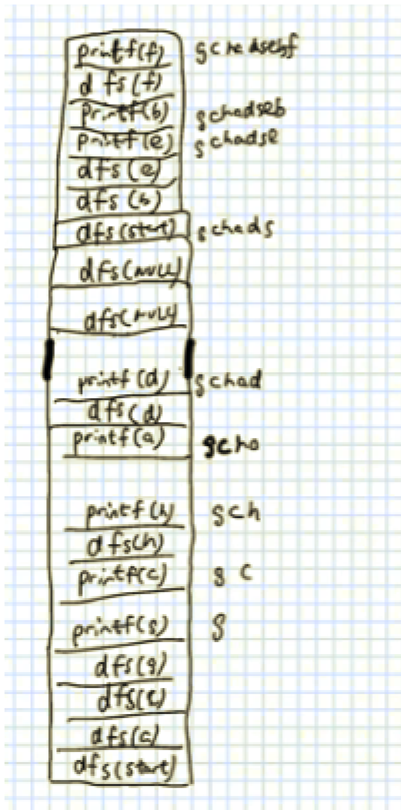
1. Draw a box to represent the stack. Add boxes that the order in which the nodes are added to the tree when they are initially malloc'd on the heap and added to the tree.



2. Draw a large box to represent the stack. Add smaller boxes that represent the recursive function calls to each node in the in your recursive implementation of `dfs` in the lab. Label each box with the letter of the node that is pushed onto the stack to be read. When a node is printed, you should draw it as coming off the stack on the right. **Your drawing for this step should show the maximum number of calls that are on the stack at one time in your recursive implementation of `dfs` in the lab. , along with which nodes they are**. Function calls that have been completed should be shown to the right of the stack.



3. State in words what the maximum number of nodes on the stack will be in your recursive implementation of `dfs` in the lab.

Ans: maximum number of nodes on stack in this recursive implementation of dfs is the depth of the tree, in this lab, the depth of the example tree is 3.

4. Describe the difference between a depth-first search of a binary tree and searching for a value in a binary search tree.

Ans: DFS is an algorithm that traverses each node in a tree, it goes from root node to leaf until it hits NULL and explores other paths, and all tree nodes will be gone through and returned. There are three ways of doing DFS: pre-order, in-order, post-order.

Searching a value is BST is done by comparing current value between the target and current node, if the target is smaller than the current node, we search the left sub tree, else we search the right subtree. It is efficient because it takes advantage of properties of binary search tree. If the target is not found the return will be NULL.