You're expected to work on the problems before coming to the lab. Discussion session is not meant to be a lecture. TAs will guide the discussion and correct your solutions if needed. We may not release solutions for all problems. If you're better prepared for discussion, you will learn more. TAs will record names of the students who actively engage in discussion and report them to the instructor. The instructor will factor in participation in final grade.

1. (Basic) Consider the BST in Fig 12.1.(b). Print out all the keys in the BST in preorder, then postorder.
   Sol. preorder: 2 5 7 6 5 8
   postorder: 5 6 8 7 5 2

2. (Basic) Suppose that we have numbers between 1 and 1000 in a binary search tree, and we want to search for the number 363. Which of the following sequences could not be the sequence of nodes examined?

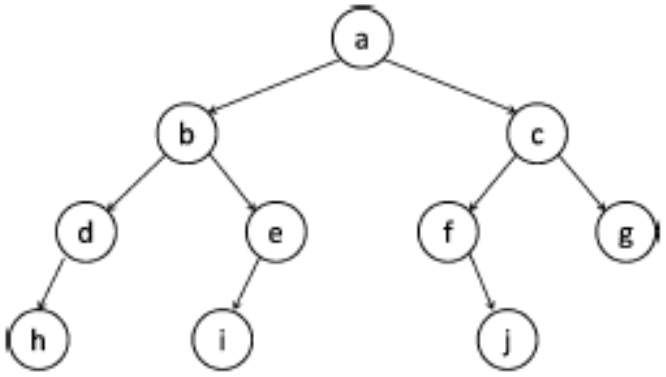
   (a) 2, 252, 401, 398, 330, 344, 397, 363.
   (b) 924, 220, 911, 244, 898, 258, 362, 363.
   (c) 925, 202, 911, 240, 912, 245, 363.
   (d) 2, 399, 387, 219, 266, 382, 381, 278, 363.
   (e) 935, 278, 347, 621, 299, 392, 358, 363.

   Sol. Check if the binary-search-tree property is satisfied.
   For example, for (a), all 2nd to the last keys are in the right subtree of the first key. And they are greater than 2. $2 < 252, 401, 398, 330, 344, 397, 363$.
   $252 < 401, 398, 330, 344, 397, 363$.
   $398 > 330, 344, 397, 363$
   $330 < 344, 397, 363$.
   $344 < 397, 363$.
   $397 >, 363$.
   so, yes for (a).

   But not (c).
   $240, 912, 245, 363$ are in the left subtree of 911. But $912 > 911$.

3. (Basic) Consider the following binary search tree where nodes are labeled by alphabets. Here the keys are *not* shown in the picture; $a, b, c, \cdots$ are node labels, not keys. Assume that all keys have distinct values.

   (a) What is the node with the min key value? **Sol.** h
   (b) What is the node with the max key value?**Sol.** g
   (c) What is the node with the upper median key value? **Sol.** a
   (d) What is the node with the lower median key value? **Sol.** e
   (e) What is e's successor? **Sol.** a
   (f) What is i's successor? **Sol.** e
   (g) What is g's successor? **Sol.** NIL
   (h) What is j's predecessor? **Sol.** f



4. (Basic) Consider the BST in the above problem.

   (a) Delete node b and show the resulting BST.

   (b) Continue to delete another node c and show the resulting BST.

   (c) Continue to delete another node h and show the resulting BST.

   Sol. See the separate pdf file.

5. (Advanced) We would like to add a new operation to binary search tree, namely AD (Ancestor-Descendant), which takes as input two nodes x and y and outputs true if x is y's ancestor, otherwise false. You can assume that $x \neq NIL$ and $y \neq NIL$. For simplicity, let's say that AD(x,x) returns true. You algorithm must run in O(the tree's height).
   Sol.

```
bool AD(x,y)
1.  if x == false or y == false, then return false;
2.  if x == y return true;
3.  else return AD(x, y.parent);
```

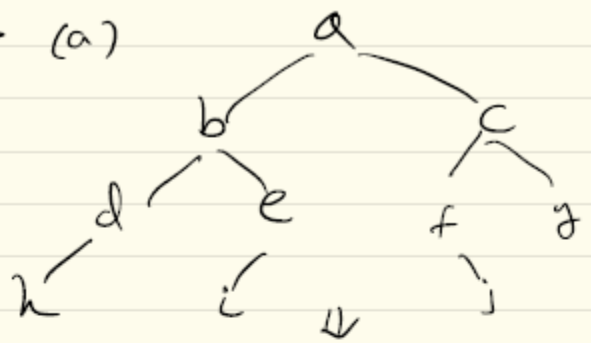
6. (Advanced) We would like to add a new operation to binary search tree, namely Sum, which takes as input a node x (or more precisely the pointer to the node), and outputs the sum of the key values of all nodes in the tree rooted at x. Your algorithm must run in $O(n)$ time. Give a pseudocode for Sum.
   Sol.

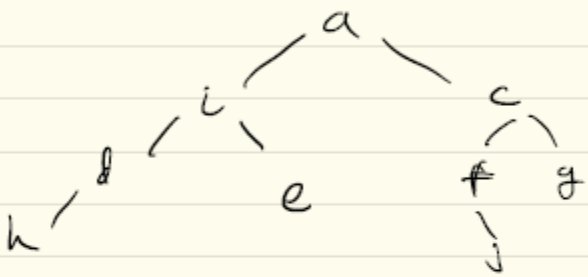
```
int Sum(x)
1.  if x == NIL return 0;
2.  else return Sum(x.left) + Sum(x.right) + x.key;
```

7. (Advanced) You're given a *sorted* array $A[1 \cdots n]$. Give a linear time algorithm that constructs a BST of height $O(\log n)$ with $A[1 \cdots n]$ as key values.
   Sol. Omitted.

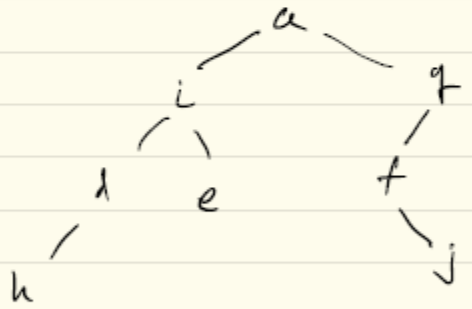
4)

Q. (a)

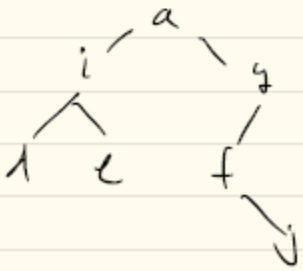




(b)



(c)



7)

// PsuedoCode

SortedArrayToBST(arr, start, end){

if start > end

return null

mid = (start + end) / 2

root = newNode(arr[mid]);

root.Left = SortedArrayToBST(arr, start, mid - 1)

root.Right = SortedArrayToBST(arr, mid + 1, end)

return root;

}

// Algorithm

1) Get the mid of the array and make it root of the tree.

2) Recursively do same for left half of the array and right half of the array.

a) Get the middle of left half of the array and make it left child of the tree root

created in step 1.

b) Get the middle of right half of the array and make it right child of the tree

root created in step 1.