1 Introduction and Objective

The purpose of this tutorial is to reinforce the concepts of Binary Search Tree (BST) and the importance of having a balanced BST. In CS2010, we learn Adelson-Velskii Landis (AVL) Tree as one such possible balanced BST implementation.

We will also discuss various techniques that may be useful to handle Subtask C+D of PS2 during this tutorial.
2 Tutorial 03 Questions

Basic Operations of (balanced) Binary Search Tree: AVL Tree

Q1. We will start this tutorial 03 with a quick review of basic BST operations, but on a balanced BST: AVL Tree. Tutor will first open [http://visualgo.net/bst.html?mode=AVL](http://visualgo.net/bst.html?mode=AVL) click Create → Random. Then, the tutor will ask students to Search for some integers, find Successor of existing integers, perform Inorder Traversal, Insert a few random integers, and also Remove existing integers.

This part is open ended, up to the tutor, and can be shortened/lengthened if the tutor feels the need to do so. The basic operations can be done quickly and the tutor has to be creative enough to ask for various corner cases, starting from operations that do not change the underlying BST:

1. Search, vary the request between existing versus non-existing integer, and close to root versus as far as possible.

2. Successor, actually the tutor will ask students to answer Predecessor operation instead to verify understanding of this mirror operation of Successor. The tutor will vary the request between vertex that has Predecessor or not (has left child versus has no left child) and the tutor will also ask about the minimum element (it has no Predecessor).

3. Inorder traversal, trivial. Show them shortcut that just echo the content of the BST in sorted order instead of manually performing Inorder traversal. The output of Inorder traversal of a BST is guaranteed to be sorted.

Then tutor will ask about two operations that modify the content of the bBST that may trigger rotation(s) and explain those rotation(s) cases accordingly:

1. Insert, vary the request between inserting to new leaf close to root versus as far as possible and whether such insertion triggers any rotation (one of the four cases).

2. Delete/Remove, vary the request between the three deletion cases (leaf vertex, vertex with one child, vertex with two children), but defer asking deletion that causes rotation to Q2 below that is specifically asked so that students put more thoughts on such delete operation.

Q2. Draw a valid AVL Tree and nominate a vertex to be deleted such that if that vertex is deleted:

a). No rotation happens
b). Exactly one of the four rotation cases happens
c). Exactly two of the four rotation cases happens (you cannot use the sample given in Lecture 04, which is [http://visualgo.net/bst.html?mode=AVL&create=8,6,16,3,7,13,19,2,11,15,18,10](http://visualgo.net/bst.html?mode=AVL&create=8,6,16,3,7,13,19,2,11,15,18,10), delete vertex 7)

This part is also open ended and there are many possible answers. The tutor will help students construct the answer.
Extra BST Operations

Q3. There are two important BST operations: Select and Rank that are not included in VisuAlgo yet but useful for PS2. Those operations are discussed very briefly at the end of Lecture 04. Please discuss on how to implement these two operations efficiently.

In pseudo code so that students still have to work this out for PS2 as some very minor detail involving null node and value v that does not exist in the bBST are not considered yet.

```java
int rank(node, v) { // assume that v exists in the BST and size attribute is there
    if (node.key == v) return node.left.size + 1; // this is discussed in Lec04
    else if (node.key < v) return rank(node.left, v); // v must be on the left
    else return node.left.size+1 // v is > node’s left and the node
        + rank(node.right, v); // and plus this rank
}
```

// select is very similar to rank and similar with QuickSelect from tut01... :O

```java
int select(node, k) { // assume size attribute is there
    int q = node.left.size;
    if (q+1 == k) return node.key; // this node has rank k
    else if (q+1 > k) return select(node.left, k); // rank k is in the left subtree
    else return select(node.right, k-q-1); // do you understand why?
}
```

Q4. What sequence does a preorder traversal of the BST in Figure 1 yield?
Preorder traversal is very similar to Inorder traversal that we have seen earlier in Lecture 03.
Preorder traversal is just like this:

```java
PreOrder(T)
    if T is null, stop
    Visit/Process T (see, the visitation is done first)
    PreOrder(T.left)
    PreOrder(T.right)
```

What about a postorder traversal of the same BST?
Postorder traversal is just like this:

```java
PostOrder(T)
    if T is null, stop
    PostOrder(T.left)
    PostOrder(T.right)
    Visit/Process T (see, the visitation is done last)
```

Preorder traversal = \{6, 2, 1, 4, 3, 5, 7, 9, 8\}.
Postorder traversal = \{1, 3, 5, 4, 2, 8, 9, 7, 6\}.
Think Carefully

Q5. What is the minimum number of vertices in an AVL Tree of arbitrary height $h$?
Note that this question has been integrated in VisuAlgo Online Quiz, so it may appear in future Online Quizzes :). 

Look at Lecture 04 carefully about $n_h$, the minimum number of vertices in an AVL Tree of height $h$. The base case is $n_0 = 1$ (mentioned in Lecture 04).
But there is one other base case, which is $n_1 = 2$ (draw a line with 2 vertices, that is an AVL tree with height 1).
The recursive case is $n_h = 1 + n_{h-1} + n_{h-2}$ for $h \geq 2$.
Later in Lecture 10-11-12, you will learn about Dynamic Programming techniques to compute this formula efficiently.
But for now, you can plug in that formula for $n_2 = 1 + n_1 + n_0 = 1 + 2 + 1 = 4$.
Then $n_3 = 1 + n_2 + n_1 = 1 + 4 + 2 = 7$.
And so on...
Note: Tutor, take your own risk if you want to explain DP now as most students may not hear about it before.

Problem Set 2
Discussion of PS2 subtask C+D.

Tutor will use this opportunity to link all the discussed items above to guide students for the (near full) solution of PS2 Subtask C+D. Tutor will explain that PS2 Subtask C+D is basically a range query. We want to find out the number of vertices in a balanced BST (99.99% of the students will implement AVL Tree) that are within the [START..END) range. This can be easily found if we already know the rank operation. How? This is discussed in class and not written in this document on purpose (so that your juniors cannot just Google this file) :). Tutor will remind students that the hardest part of PS2 is the implementation... especially the insertion/deletion cases that requires rotation(s). Students who have not attempted PS2 should do so ASAP or risk not sleeping on Friday, 11 September night/Saturday, 12 September early morning.