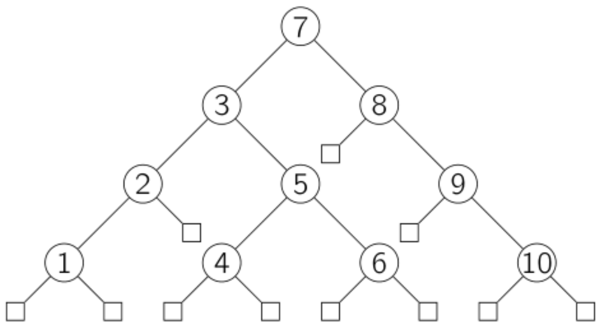
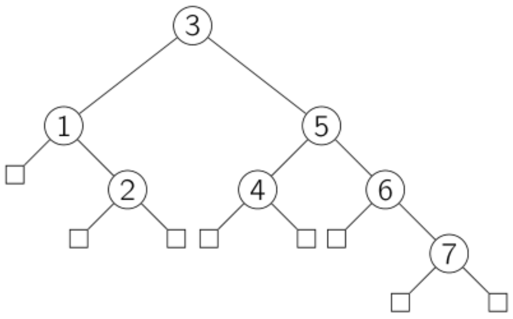
Sample Solution <Unofficial> (Class Test 1 2023)

Question 1





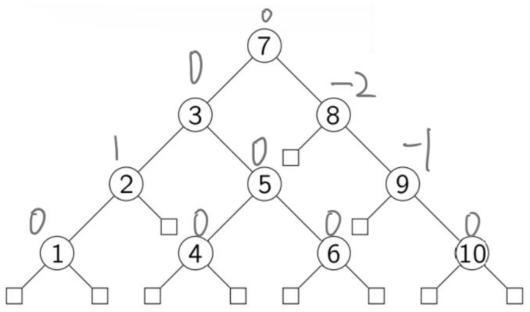
Part 1 Consider the two trees above and answer the following two questions. Briefly motivate your answers.

(a) Determine which of these two trees is perfectly balanced. **[2 marks]**

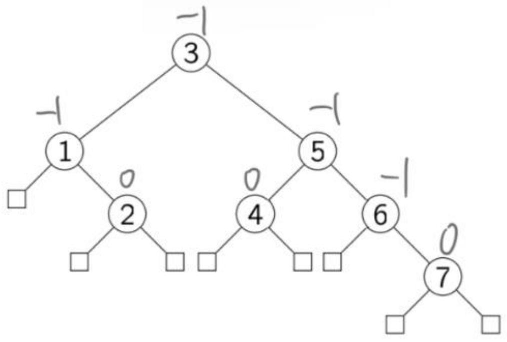
(b) Determine which of these two trees is an AVL tree. **[2 marks]**

Answer (for proof, see the following Trees annotated with balance = Height (L)-Height(R))

(a) Neither of these two trees is perfectly balanced
(b) The second tree (right) is an AVL tree.



Left Tree: Balance of every node



Right Tree: Balance of every node

Part 2 Consider a general variant of binary search trees that only admits nodes with balance -2, -1, 0, 1, or 2. Answer the following questions about the size of such trees.

(c) What is the maximum possible number of nodes of a tree with height 6? **[2 marks]**

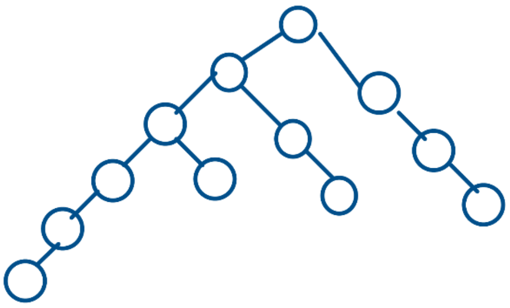
(d) What is the minimum possible number of nodes of a tree with height 6? **[2 marks]**

Answer:

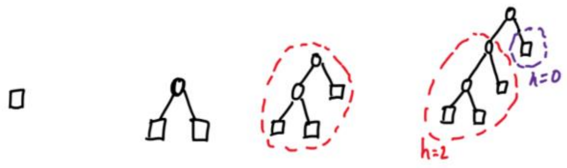
(c) The maximum possible number of nodes can be calculated using $2^h - 1$ where the h is the height. Therefore, for a binary search tree of height 6 the maximum possible nodes will be 2^6-1 which equals 63.

(d) Minimum number of possible nodes of a tree with height 6 is 12. Students can show their answers in various ways. For instance,

Method 1: Draw it in such a way that each node has a balance between -2, -1, 0, 1 or 2. Counting the nodes to show that they are 12.



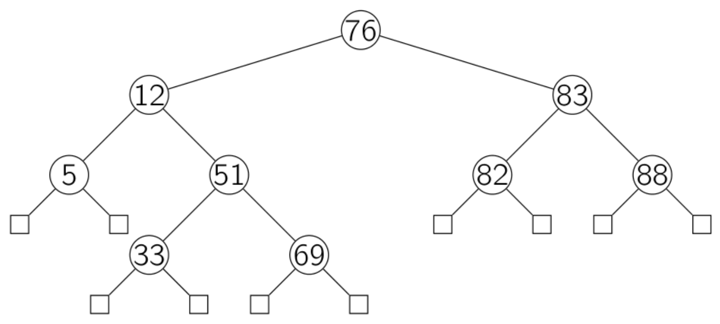
Method 2: Recording the arrangement of the initial four smallest trees that fall within the balance range of -2 to 2:



	h=0	h=1	h=2	h=3
Balance	0	0	1	2
Nodes	0	1	2	3 (1+2+0) $1+h_2+h_0$

Height	Formula	Nodes
0		0
1		1
2		2
3	$1+h_2+h_0$	3
4	$1+h_3+h_1$	5
5	$1+h_4+h_2$	8
6	$1+h_5+h_3$	12

Question 2



(a) Consider the AVL tree above and delete the element with value 76. **[2 marks]**

(b) Consider the AVL tree above and insert an element with value 40 **[2 marks]**

Answer

(a) We replace 76 with 82 (left most node of right subtree)

```
        82
       /  \
     12    83
    /  \     \
   5    51    88
       /  \
      33   69
```

Check Balance (82) = 3-2 = 1

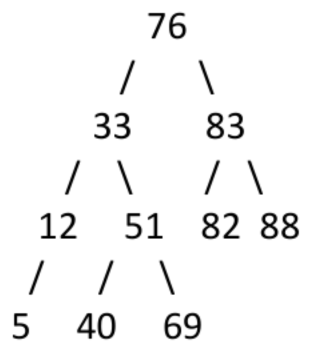
Check Balance (12) = 1-2 = -1

Check Balance (83) = 0-1 = -1

(b) After inserting 40, tree will become

```
        76
       /   \
     12     83
    /  \    / \
   5   51  82  88
       / \
     33   69
      \
      40
```

The node 12 is now unbalanced, so a right rotation at node 51 and a left rotation at node 12.



Check Balance (76) = 3-2 = 1

Check Balance (33) = 2-2 = 0

Check Balance (83) = 1-1 = 0

Question 3

Consider a procedure that visits the elements of a tree and constructs a list with these elements. Let us define a function `visit`:

$$\begin{aligned} \texttt{visit}(\square) &= [\,] \\ \texttt{visit}(\mathsf{Fork}(x, l, r)) &= \texttt{visit}(r) + [x] + \texttt{visit}(l) \end{aligned}$$

where $[\dots]$ denotes a list of elements. In particular, the special case $[\,]$ denotes the empty list and operation $+$ denotes concatenation of lists. For example, $[3, 4, 1] + [2, 3, 5]$ results in $[3, 4, 1, 2, 3, 5]$. Consider a function named `doSomething` that takes an array of integers a with n elements (that is, `a.length` $= n$) and first inserts these elements in a binary search tree and then constructs a list using the function `visit`.

```
List doSomething(int [] a) {
  Tree T = □;
  for (int i = 0; i < a.length; i++) {
    assert(#nodes(T) == i); // invariant
    T = insert(T, A[i]);
  }
  List L = visit(T);
  assert(length(L) == a.length); // postcondition
  return L;
}
```

(a) Briefly describe the task this algorithm performs. Be precise about the content of the output list `L` in relation to the input array `a`. **[2 marks]**

(b) The invariant and the postconditon given in this program do **not always** hold true. Under what condition would they hold true? **[3 marks]**

(c) Give the wost case time complexity of `doSomething` in term of the input size n = `a.length`. Be precise about the kind of binary search tree you refer to when motivating your answer. **[3 marks]**

Answer:

(a) The algorithm inserts all elements from the input array into a binary search tree and then generates a list of these elements in descending order, effectively sorting the input elements and removing duplicates.

(b) If the algorithm is given an array with duplicate values i.e., [1, 2, 2, 3, 4], the invariant would fail during the 3rd iteration when it omits adding the second 2 to the tree. This failure arises because the number of nodes in the tree would be 2, yet the current iteration count would be 3. Similarly, the postcondition would also fail due to the comparison between the array length and the list length, which reflects the number of tree nodes and excludes duplicate values. Nevertheless, both the invariant and postcondition would remain valid if the array had no duplicates or if the insert function did not disregard existing elements in the tree.

(c) The worst-case time complexity of the function doSomething is O(n^2) if it operates on a regular binary search tree that isn't perfectly balanced. This arises because it initially iterates through array a, which has a worst-case time complexity of O(n). Moreover, the insert function within the for loop also exhibits a worst-case time complexity of O(n) as it inserts elements one by one into the binary search tree T. Therefore, this results in a total complexity of n x n, which equals n^2.