**Advanced Algorithms**

**Exercise for Lecture 5**

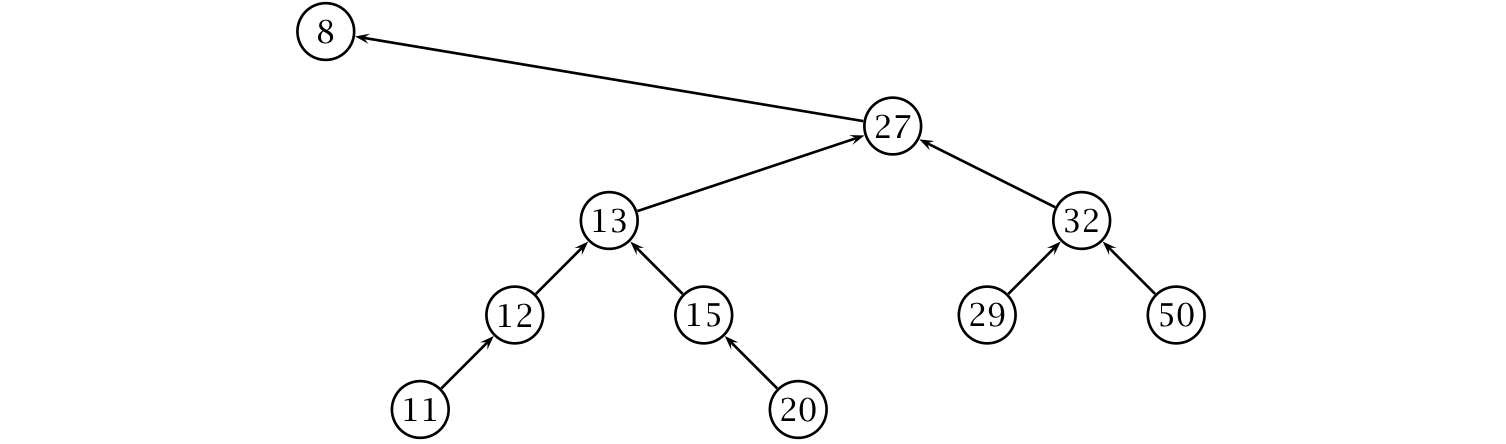
|  |  |  |  |
| --- | --- | --- | --- |
| **Student Name** |  | **Student ID** |  |
| **Problem 1** |  | | |
| **Problem 2** |  | | |
| **Problem 3** |  | | |
| **Problem 4** |  | | |
| **Total Score** |  | | |
| **Notes** | Deadline: **2023-09-26 24:00**  Submission Format: ‘**Lecture5\_Name\_Student ID.docx**’, and please send to: **[algorithms\_23fall@163.com](mailto:algorithms_23fall@163.com)**.  This assignment is meant to be an evaluation of your **individual** understanding coming into the course and should be completed **without collaboration** or outside help. | | |

**Problem 1. [25 points]** Draw a binary search tree containing keys

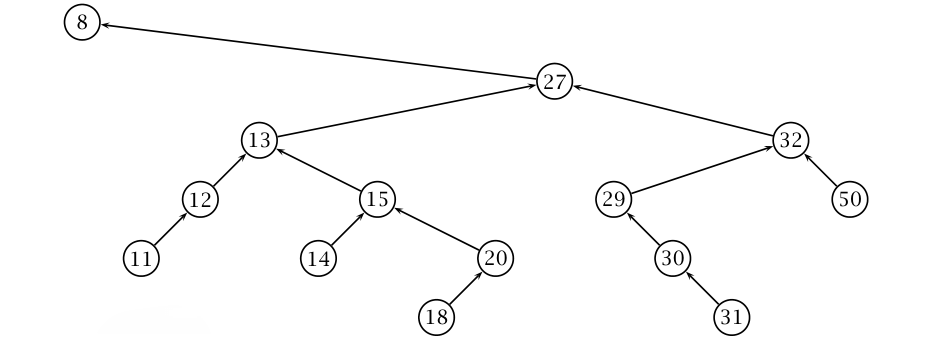
, inserted in this order. Then, add keys , in this order, and again draw the tree. Then delete keys and , in this order, and again draw the tree.

**Solution:**

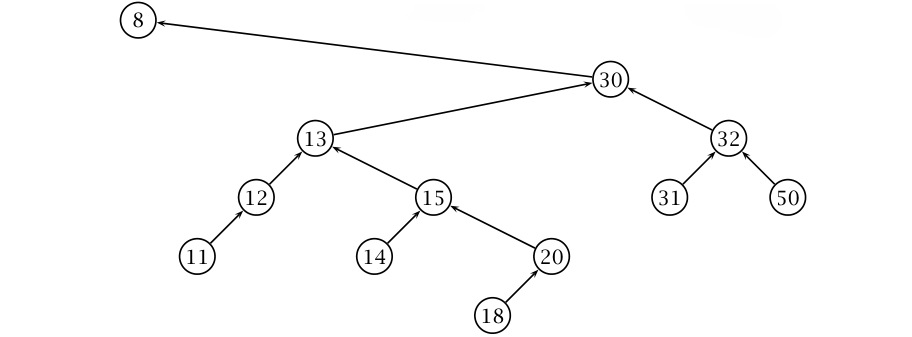
Step 1: [8 points]



Step 2: [7 points]



Step 3: [10 points]

****

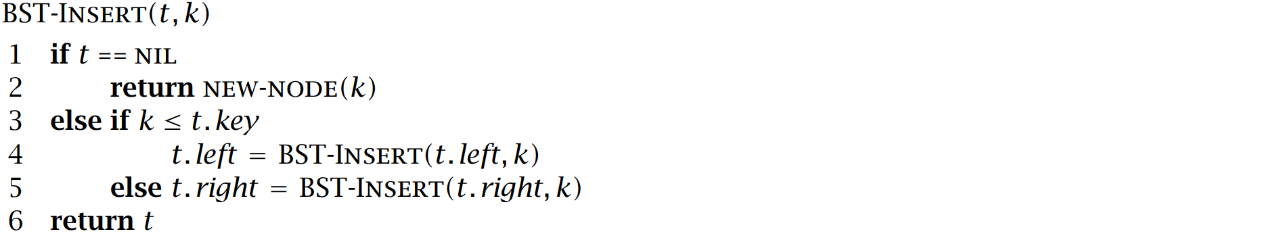
**Problem 2. [20 points]** Given a collection of numbers and a number , the upper bound of in is the minimal value such that , or NULL if no such value exists. For example, given , the upper bound of is , while the upper bound of is and the upper bound of is NULL. Write an algorithm UPPER-BOUND-BST() that returns the upper bound of in a binary search tree . Analyze the complexity of UPPER-BOUND-BST.

**Solution:**

UPPER-BOUND-BST() [18 points]

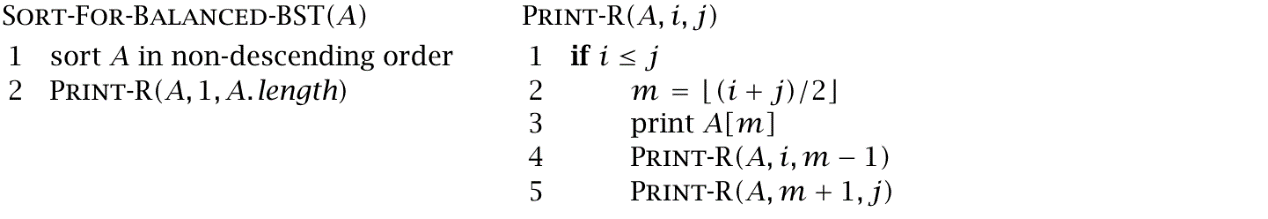
1. **while** NULL
2. **if**
4. **else** **while** NULL **and**
6. **return**
7. **return** NULL

The complexity is where is the height of the input tree. [2 points]

**Problem 3. [25 points]** Consider the following classic insertion algorithm for a binary search tree:

Write an algorithm SORT-FOR-BALANCED-BST() that takes an array of numbers , and prints the elements of in a new order so that, if the printed sequence is passed to BST-INSERT, the resulting BST would be of minimal height. Also, analyze the complexity of your solution.

**Solution:**

In this exercise, randomization or rotations cannot be used to balance the height of the BST. So, input sequence must be pre-sorted so that, inserting elements in the tree in the new order, the resulting BST has still minimal height, , even using the classic insertion algorithm (that could potentially result in unbalanced trees). Intuitively, this is possible by inserting elements in this order: *median(1, n)*, *median(1, n/2)*, *median(n/2, n)*, *median(1, n/4)*, *median(n/4, n/2)*, *median(n/2, 3n/4)*, *median(3n/4, n)*. Or, equivalently, *median(1, n)*, *median(1, n/2)*, *median(1, n/4)*, *median(n/4, n/2)*, *median(n/2, n)*, *median(n/2, 3n/4)*, *median(3n/4, n)*. The input array can be sorted in this order by using the functions below: [20 points]

PRINT-R runs in , since it simply prints one element—the median element, since the input is sorted—and then recurses on the left and side parts by excluding the element it just printed. In the end, PRINT-R runs (recursively) exactly once for each element of the array. So, the complexity of PRINT-R is and the dominating cost for SORT-FOR-BALANCED-BST is the cost of sorting, which can be done in . [5 points]

**Problem 4. [30 points]** Consider an algorithm BST-FIND-SUM() that, given a binary search tree containing distinct numeric keys, and given a target value , finds and returns two nodes in whose keys add up to . The algorithm returns NULL if no such keys exist in . BST-FIND-SUM may not modify the tree, and may only use a constant amount of memory.

1. Write BST-FIND-SUM(). You may use the basic algorithms that operate on binary search trees (TREE-MINIMUM, TREE-SUCCESSOR, TREE-SEARCH, etc.) without defining them explicitly.
2. Write a variant of BST-FIND-SUM() that works in time. If your solution to *Exercise a* already has this complexity bound, then simply say so.

**Solution:**

1. BST-FIND-SUM() [8 points]
2. TREE-MINIMUM()
3. **while** NULL
4. TREE-SEARCH()
5. **if** NULL
6. **return**
7. **else** TREE-SUCCESSOR()
8. **return** NULL

The time complexity is . [2 points]

1. BST-LOWER-BOUND()

// rightmost element whose key is , or NULL

1. **while** NULL
2. **if**
4. **elseif**  NULL **and**
6. **else** **return**
7. **return** NULL

BST-FIND-SUM() [18 points]

1. BST-LOWER-BOUND()
2. TREE-SUCCESSOR()
3. **while** NULL **and** NULL
4. **if**
5. **return**
6. **elseif**
7. TREE-SUCCESSOR()
8. **else** TREE- PREDECESSOR()
9. **return** NULL

The time complexity is . [2 points]