# Exercises: Binary Search Trees

This document defines the **exercise assignments** for the ["Data Structures" course @ Software University](https://softuni.bg/trainings/1857/data-structures-january-2018). You can submit your **C#** code in the SoftUni Judge System - <https://judge.softuni.bg/Contests/604/Binary-Search-Trees-CSharp-Exercise>. You can submit your **Java** code in the SoftUni Judge System - <https://judge.softuni.bg/Contests/607/Binary-Search-Trees-Java-Exercise>.

# Implement BST Operations

You are given a skeleton, in which you will find implemented the following operations:

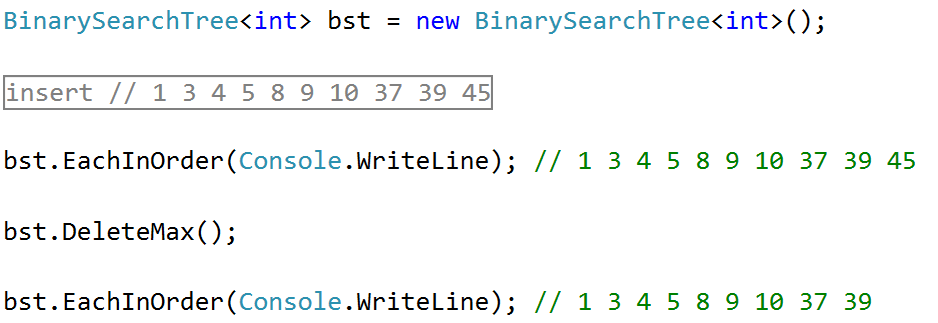
* void Insert(T) – Recursive implementation
* void EachInOrder(Action<T>) – In-Order traversal
* bool Contains(T) – Iterative implementation
* BST<T> Search(T) – Returns copy of the BST
* IEnumerable<T> Range(T, T) – Returns collection with the elements found in the BST. Both borders are **inclusive**.
* DeleteMin() – Deletes the smallest element in the tree. Throws exception if the tree is empty.

You will need to implement the rest of the operations, that are defined below:

|  |  |  |  |
| --- | --- | --- | --- |
| C# Method | Java Method | Return Type | Exception C#/Java |
| **DeleteMax()** | deleteMax() | void | C# - InvalidOperationException Java -IllegalArgumentException |
| **Count()** | size() | int |  |
| **Rank(T)** | rank(T) | int |  |
| **Select(int)** | select(int) | T | C# - InvalidOperationException |
| **Ceiling(T)** | ceiling(T) | T | C# - InvalidOperationException |
| **Floor(T)** | floor(T) | T | C# - InvalidOperationException |
| **Delete(T)** | delete(T) | void | C# - InvalidOperationException Java - IllegalArgumentException |

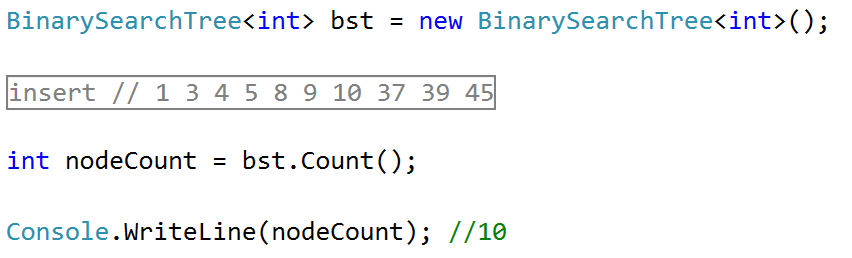
## Delete Max

Implement a **method** which **deletes** the **max** **element** in a BST (Binary Search Tree). If the tree is empty it should throw exception. The logic is similar to the DeleteMin() method, but you need to traverse the tree to the right.



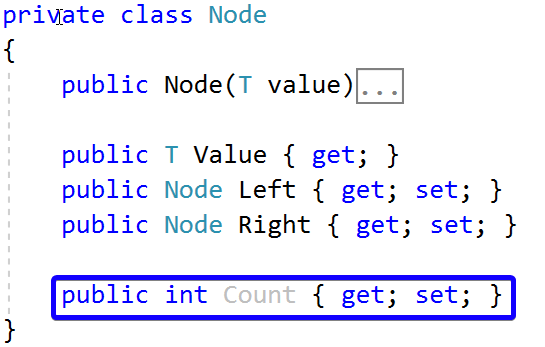
## Count

Implement a **method** which returns the count of elements in the BST.

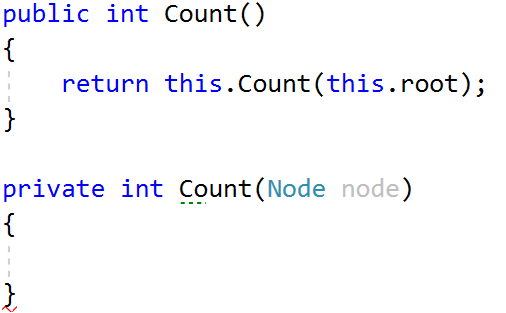


### Hints

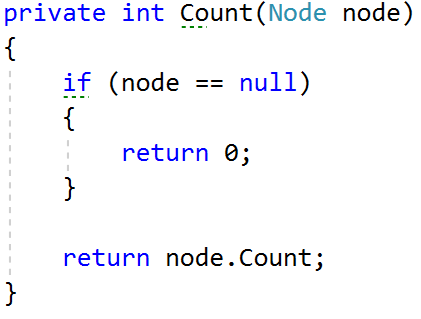
In order to implement the count, we will create a new field in our Node class:



Now we can create new method Count(Node), which will recursively find the count of elements:



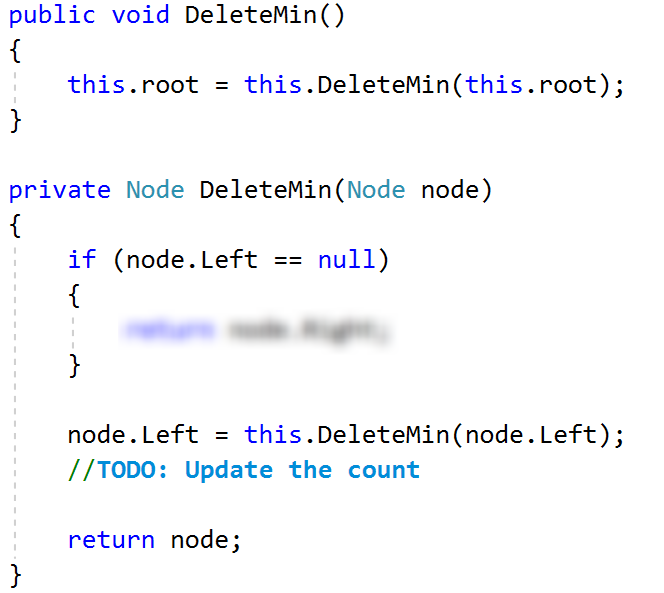
If our current node is null, we will return 0. Otherwise, we will return the count of our current node:



Now we only have to modify our Insert() method. It will set the count of elements of our new node to the count of its children nodes plus itself:



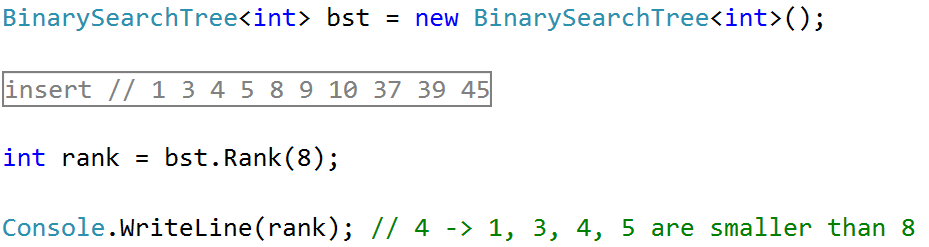
Next, we need to find a way to update the recalculate the count for each node when DeleteMin() is invoked. One way would be to change the DeleteMin() implementation to be recursive:



What will happen if our tree is empty and we call DeleteMin()? **Fix** it. Our count is ready.

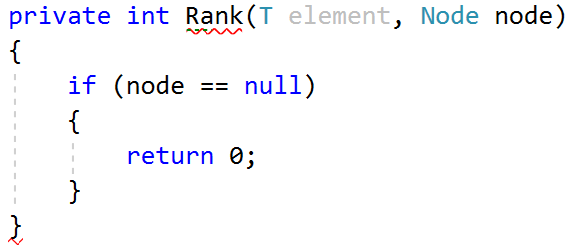
## Rank

Implement a **method** which **returns** the **count** of elements **smaller** **than** a given **value**.

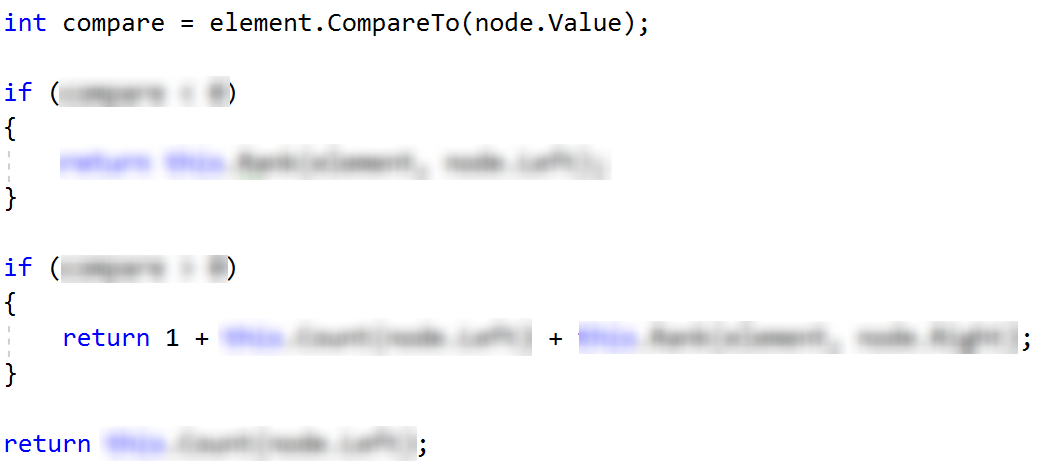


### Hints

Create a new recursive method that will return 0 if the node is null:



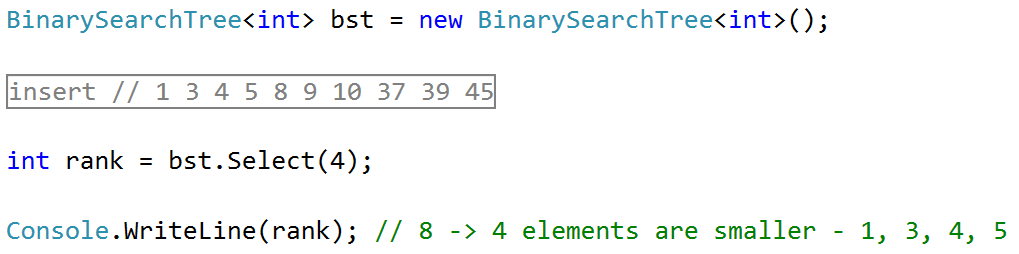
Then, we need to **compare the element** with the value of the node we are currently looking at. If the element is **smaller**, we can **go to the left**. If its **larger**, we need to **get the count of the left** elements and **go to the right**. If we **find the element**, we will return the **count of elements**, **smaller** than it.



You can try it out, it should work as expected.

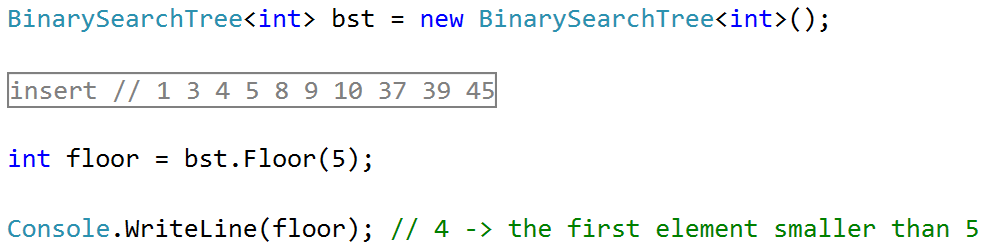
## Select

Implement a **method** which accepts a number (**n**) and **returns** the first **element** which has exactly **n** elements **smaller** than it. Use the logic from Count() and Rank() to implement it.



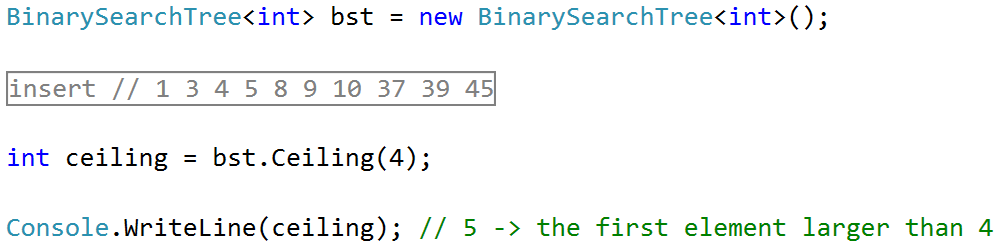
## Floor

Implement a **method** which **finds** (returns) the **nearest** **smaller** **value** than given in the BST. This operation is similar to DeleteMin().



## Ceiling

Implement a **method** which **finds** (returns) the **nearest** **larger** **value** than given in the BST. This operation is similar to Floor() and DeleteMax().



## Delete\*

Implement a **method** which deletes a node with given value.

