**Binary Search Tree**

A set of nodes T is a binary search tree if either of the following is true:

* T is empty
* If T is not empty, its root node has 2 subtrees, and , such that and are binary search trees and the value in the root node of T is greater than all values in and is less than all values in .
  + = left subtree; = right subtree

There is only 1 instance of every element in the binary search tree

Diagram, radar chart

Description automatically generated

Recursive Algorithm for Searching a Binary Search Tree

Graphical user interface, text, application

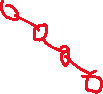
Description automatically generated

*n = 0 is base case, not the best case.*

* best case - target value at the root :
* worst case - traversing until a leaf (depends on height) :

h depends on structure of the tree, not the contents. For example, best case is about the content. h depends on the structure of a tree.

* Best height (lower height) : complete tree 🡪
* Worst height (highest height) :



So if we are talking about worst case, height would be linear so worst case is

T(n) = O(n)

Worst height case is very rare. If we analyze the average case:

h(n) =

So for average: Tav(n) =

**INTERFACE SearchTree<E>**

Table

Description automatically generated

**CLASS BinarySearchTree<E>**

Graphical user interface, text, application

Description automatically generated

Diagram

Description automatically generated  
There are 1 data field (root) in the binary tree.

- find() (You can delegate contains to find)

Text

Description automatically generated

Recursive algorithm for insertion:

Text

Description automatically generated

**- add()**

Text, letter

Description automatically generated

add(root, item) will return the tree that is obtained after the insertion.

add(root, item) returns 2 things:

1. tree after insertion
2. whether the insertion successful or not

addReturn is the side effect of this recursive add method.

Text

Description automatically generated

We always insert as a leaf node.

Number of recursive operations depends height of the tree.

* best case - item is at the root : Tb(n) =
  + *Tree’nin empty olması best case değil, base case!*
* Tw(n) =

--------------------------------------------------------

Generally height is thought as logarithmic:

* + Taverage(n) = (logn)

You cannot say 🡪 T(n) = O(logn) 🡪 THIS IS FALSE

T(n) = O(n) 🡪 THIS IS CORRECT

Removal from a Binary Search Tree

If the item to be removed has no children, simply delete the reference to the item

If the item to be removed has only one child, change the reference to the item so that it references the item’s only child

Diagram, radar chart

Description automatically generated

If we want to remove “rat”, we can put “shaven” to the place of rat because shaven is greater than every node at the left subtree and smaller than every node at the right subtree of rat.

We can also put priest to the place of rat and put morn to the place of priest.

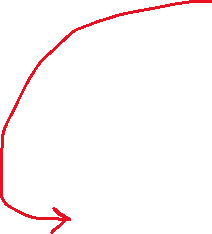
Three cases:

1. Delete a leaf node
2. Delete a node that has 1 child
3. Delete a node that has 2 children
   1. Find smallest element on the right subtree OR greatest element on the left subtree then replace the root node with that value

Recursive delete method will return the tree after deletion and return the element that is removed (field deleteReturn) as a side effect.

Text

Description automatically generated



Shape

Description automatically generated Shape

Description automatically generated



Tb(n) = 🡪 line 12  
Tw(n) = = 🡪 deleting a leaf node, or a node that has 1 child which is leaf. (Tav(n) = )  
Worst case of deleting a node with 2 children :

Text

Description automatically generated

Text, letter

Description automatically generated

Text, letter

Description automatically generated

- findLargestChild()

Text

Description automatically generated

Tb(n) =   
Tw(n) = =

Testing a Binary Search Tree

To test a binary search tree, verify that an inorder traversal will display the tree contents in ascending order after a series of insertions and deletions are performed.

Structural property: being a binary tree  
Order property: node is greater than everything at left, smaller than everything at right

Writing an Index for a Term Paper

Problem: write an index for a term paper

* The index should show each word in the paper followed by the line number on which it occurred
* The words should be displayed in alphabetical order
* If a word occurs on multiple lines, the line numbers should be listed in ascending order:
  + a, 003
  + a, 013
  + are, 003

Analysis

Store each word and its line number as a string in a tree node

For example, two occurrences of “java”: “java, 005” and “java, 010”

Display the words in ascending order by performing an inorder traversal

Design

Use TreeSet<E>, a class based on a binary search tree, provided in the Java API

Write a class IndexGenerator with a TreeSet<String> data fields

Graphical user interface, text, application

Description automatically generated

TreeSet<E> : Balanced binary search tree

Text, letter

Description automatically generatedText, letter

Description automatically generated

You can also use this:

Text

Description automatically generated