COMP3506/7505 – Assignment 1

Due: 5pm 20 Sep 2015 **Revision: 1.2**

Change log from 1.1:

- Now 3 errors in AdsStack
- Added tests for AdsList to help find errors and demonstrate useage
- Updated wording in support document

Exercise 1 – Debugging a linear data structure [4 marks]

Deliverable: edited *AdsList.java* and *AdsStack.java* files.

The package assign1 contains an AdsList class implementing a list, an AdsStack implementing a stack using the AdsList class and a Tester class (see last testing section of the task sheet) containing short procedures making use of the AdsList and AdsStack data structures.

AdsList.java contains 2 errors and AdsStack.java contains 3 errors. ~~contain two errors each~~. Your task is to find these five ~~four~~ errors and correct them. The Tester class is here to help you in this task you are encouraged to add additional tests. The main method in the Tester class should run without error after correction of the four errors. **For each error you should need to modify/add at most 3 lines.**

When you find an error, insert a brief one-line "//" comment at that location, indicating where the error had been found, and why it occurred. You may not modify the code other than to fix the 5 ~~2~~ ERRORS and insert the required comments.

Exercise 2 – A sorting algorithm using a tree [5 marks]

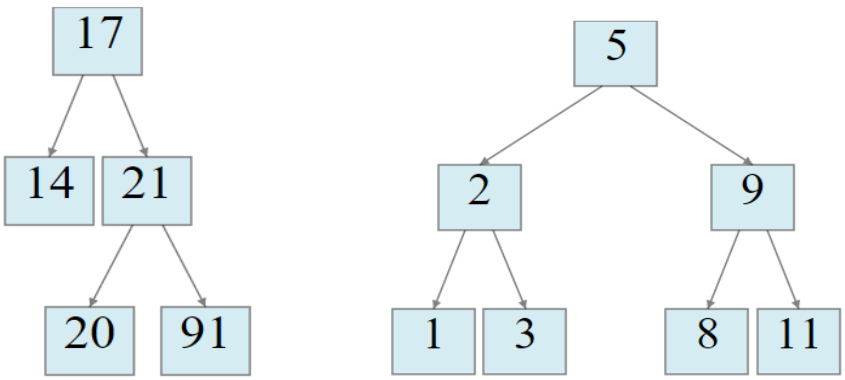
Deliverable: a pdf file containing your answers for questions (a) to (e). You will also write your answer to question 3(a) on this same document. For the insertion of images of trees, the webpage http://jimblackler.net/treefun/index.html can be used.

A binary search tree (BST) is a binary tree whose internal nodes each store a key, and each have two distinguished sub-trees, commonly denoted left and right. The tree additionally satisfies the binary search tree property, which states that the key in each node must be greater than all keys stored in the left sub-tree, and smaller than all keys in right sub-tree.

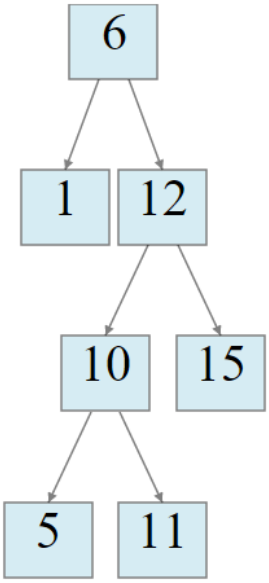
The major advantage of binary search trees over other data structures is that the related sorting algorithms and search algorithms such as in-order traversal can be very efficient; they are also easy to code. Binary search trees are a fundamental data structure used to construct more abstract data structures such as sets, multisets, and associative arrays.

Source: https://en.wikipedia.org/wiki/Binary_search_tree

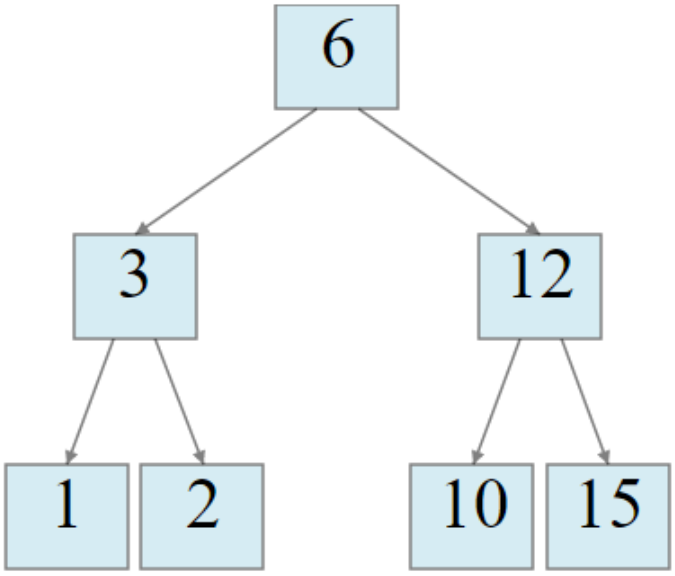
Examples of valid BSTs:



Examples of invalid BSTs:



invalid, because 5<6



invalid, because 2 < 3

To insert an element in a BST, we compare it with the root of the tree, and insert it to the left sub-tree if it strictly smaller, the right sub-tree otherwise. We repeat this process until a free spot is found.

To remove an element in a BST, we delete it, and if it is an internal node, we replace it with either the highest number of the left sub-tree, or the smallest number of the right sub-tree. Then we repeat the process on the spot that the replacing node left vacant, until we have a valid tree.

From more detailed explanations, you can refer to the video *Binary Search Trees (BSTs) - Insert and Remove Explained* on YouTube: https://www.youtube.com/watch?v=wcIRPqTR3Kc

(a) (0.5 Mark) Draw the BST obtained after the insertion of the following numbers, in this order, to an initially empty tree: 30, 40, 24, 58, 48, 26, 11, 13, 35, 36.

(b) (1 Mark) Give and justify the complexity (function of n, the number of nodes) of the operation of insertion in a BST in the best case and in the worst case.

(c) (0.5 Mark) Draw the BST obtained after the deletion of the following numbers, in this order, from the tree obtained in question (a): 13, 58, 30.

(d) (1 Mark) Give and justify the complexity of the operation of deletion of the root of a BST in the best case and in the worst case.

(e) (2 Marks) Suggest a sorting algorithm that takes a list of elements to sort as an input and that outputs the list in order, using a BST in the process. Give (and justify) a big-O and a big-Omega characterisation of its complexity.

Exercise 3 – implementation of expression trees in Java [11 marks]

Deliverable: edited *Tree.java* file.

Three new expressions are defined as:

Expression 1: **A ? B = max (A, B)**

Example 1: 1 ? 2 = max (1, 2) = 2

Exemple 2: 5 ? (2 ? 3) = max (5, max (2, 3)) = max (5, 3) = 5

Expression 2: **A % B = A mod B**

(refer to https://en.wikipedia.org/wiki/Modulo_operation)

Example 1: 1 % 2 = 1 mod 2 = 1

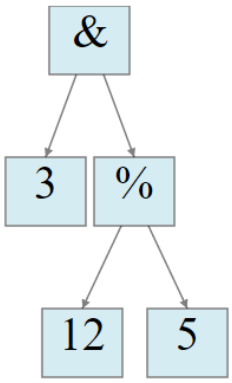
Example 2: 12 % 5 = 12 mod 5 = 2

Example 3: 12 % (2 % 3) = 12 mod (2 mod 3) = 12 mod 2 = 0

Expression 3: **A & B = A + B**

Example 1: (4 & 6) % (2 ? 3) = 10 mod 3 = 1

In this exercise, we use trees to represent expressions involving these three operators and positive integers. The following rules apply to the expressions: All operations are wrapped in parentheses, except the whole expression. There is a single white space around each operator.

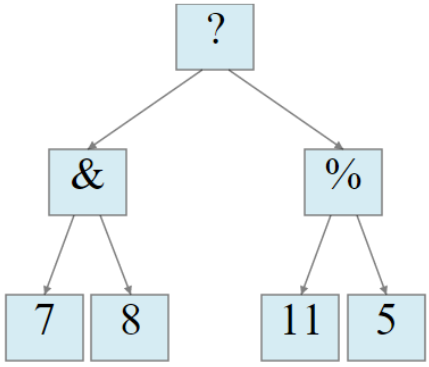


Example 1:

3 & (12 % 5)

= 3 & 2

= 5



Example 2:

(7 & 8) ? (11 % 5)

= 15 ? 1

= 15

You will need to edit the Tree class that you will find in the *assign1* package, in order to add the following methods. Do not edit the other classes or the other methods.

(a) **(3 marks)** Write a constructor for the class that creates a tree from an arithmetic expression given as a String (e.g. "`(4 & 6) % (2 ? 3)`"). The method should print an error message if the expression is invalid, e.g. (4 & &) % 2 7. The possible errors are:

- Wrong number of argument for an operator;
- Non-positive number;
- Non-matching parenthesis;
- Missing operator.

An invalid input should not lead to the construction of a Tree object.

What is the complexity of your algorithm? Justify it in your PDF report.

(a) **(0.5 mark)** Write a method `getHeight()` that returns the height of the tree.

(b) **(0.5 mark)** Write a method `nbLeaves()` that returns the number of leaves (numbers) of the tree.

(c) **(2 marks)** Write a method `getResultRec()` that computes the result of the expression using recursion. If the expression is invalid (If it contains a zero on a right side of a *mod* operation), the method should return -1.

(d) **(3 marks)** Write a method `getResultBySteps()` that returns a List<String> of the tree containing an element for every step of the calculation:

```
(4 & 6) % (2 ? (1 & 2))
(4 & 6) % (2 ? 3)
10 % (2 ? 3)
10 % 3
1
```

The first element of the List should be the tree itself without any changes, the last element should be a result. You should be able to make a tree from any element in this array (they should all be in a valid form). If the expression is invalid, the computation should stop as soon as the error is detected and return the list in its current state, with a "-1" at the end to indicate the invalidity. Example:

```
12 & (3 % (12 % 2)
12 & (3 % 0)
-1
```

(e) **(2 marks)** Write a method `getResultIt()` that computes the result of the expression without using recursion. That is, the body of the method `getResultIt()` cannot contain any calls to `getResultIt()`. If the expression is invalid, the method should return -1.

Running the tests

To help validate your assignment solutions some JUnit tests have been provided. Your Java Build Path will need to include the JUnit 4 Library to run the JUnit tests. Being able to test your solutions will help ensure you don't accidently break one method when fixing another. Usually developers will write tests before they write their actual solution so they know what output they are expecting and that the final system has the behaviour stated at the start.

The tests you have been provided with are a subset of the tests that will be used to assign a final mark to your assignment. You are encouraged to develop more tests on your own to validate your solution before submission. You do not need to submit your own tests when submitting the assignment.

Getting help

Cheating is not tolerated in UQ courses, if you are having trouble completing the assignment speak to teaching staff. All cases of academic misconduct will be dealt with as per UQ policy.

You are encouraged to ask questions/discuss the assignment on the course newsgroup:
https://student.eait.uq.edu.au/infrastructure/newsgroups.html