L2 task

L2 mandatory part

1. Examine the code in the l2-assignment-binary-search-trees project.

2. (**5 points**) Implement the unimplemented methods and test them (tests can be initiated either through the project user interface or the console).

   The following methods need to be implemented in the BstSet class:

   - remove(E element);
   - removeRecursive(E element, BstNode<E> node) - auxiliary remove method.
   - Iterator method remove();
   - addAll(Set<E> set) - place all elements of *set* into an array, if both arrays have the same element it is not added.
   - containsAll(Set<E> set) - check if all elements in *set* exist in the set.
   - retainAll(Set<E> set) - to select items that are not in *set*.
   - headSet(E e) - returns a subset of the set up to element e.
   - tailSet(E e) - returns a subset of the set from element e.
   - subSet(E element1, E element2) - returns a subset of the set from element1 to element2.

   The following methods need to be implemented in the AvlSet class:

   - remove();
   - removeRecursive(E element, BstNode<E> node) - auxiliary remove method.

3. (**3 marks**) Perform benchmarking (using JMH) with one pair of methods from the table below:

No.	**Method 1**	**Method 2**
1	Class BstSet: remove()	Class AvlSet: remove()
2	Class BstSet: contains()	Class AvlSet: contains()
3	Class BstSet: containsAll()	Class AvlSet: containsAll()
4	Class BstSet: addAll()	Class AvlSet: addAll()
5	Class BstSet: retainAll()	Class AvlSet: retainAll()
6	Class BstSet: remove()	Class java.util.TreeSet<E>: remove()
7	Class BstSet: contains()	Class java.util.TreeSet<E>: contains()
8	Class AvlSet: remove()	Class java.util.TreeSet<E>: remove()
9	Class AvlSet: contains()	Class java.util.TreeSet<E>: contains()

Note: The actual task number will be assigned to you by your lecturer.

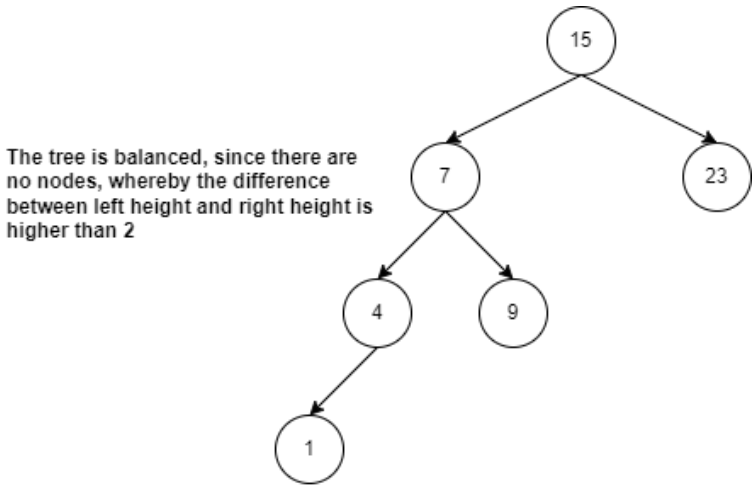
Prepare a free-form report. The report must include:

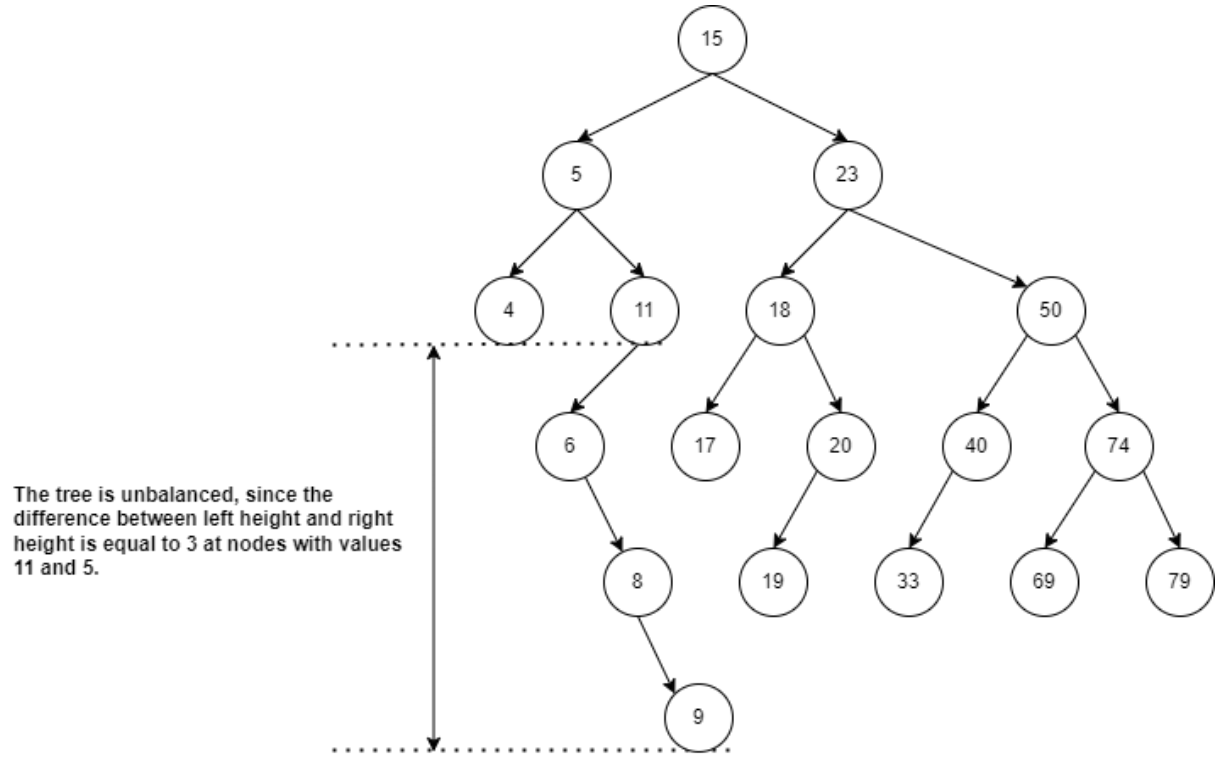
- Description the benchmarked methods.
- The computational complexity of the benchmarked methods.
- The description of benchmark methodology (How benchmarking was carried out).
- The Graph whereby the dependence of the execution time and the input data size is depicted.
- Conclusions. The conclusions, among others, should answer the following questions in free form:
    - Whether the experimentally determined dependence of the execution time and the input size is consistent with the computational complexity of the algorithm/method. If not, why?
    - Which of the methods studied is better in terms of the runtime. Why?

L2 optional part

Solve the following problems:

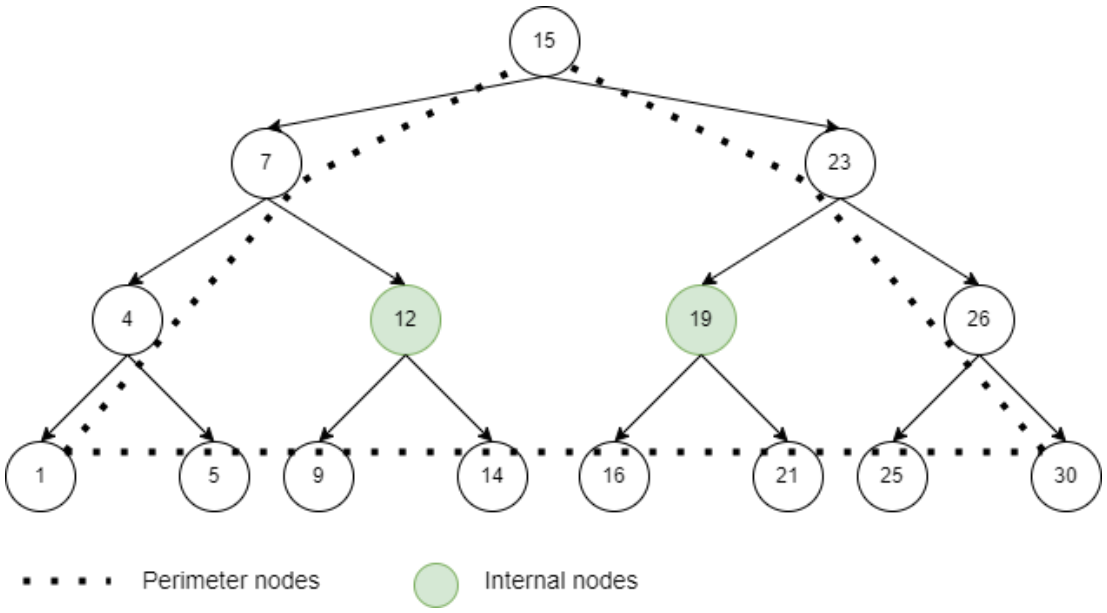
1. (**1 point**) We have a binary search tree with the data of your choice (the object stored in the tree must implement the Comparable interface). Check that the balance of branches of the tree does not exceed an integer k (k ≥ 0). Consider the following examples when k = 2:





The input parameters of an algorithm must comprise the number k and the binary search tree. Implement an algorithm to determine whether the given binary search tree is balanced with respect to parameter k. The computational complexity of the algorithm must be O(n).

2. (**1 point**) We have a complete binary tree. Write an algorithm to find the internal nodes of the tree, i.e. the nodes inside the perimeter of the tree. Consider the following example:



L2 Defence

During the defence, lecturers may ask students to complete the following tasks and answer the questions below. The list of questions and exercises is not exhaustive - lecturers may ask questions or practical exercises related to the topic during the defence that are not included in this list.

Possible practical exercises from the L2 mandatory part

Implement one or more of the methods from the following list:

Value returned	**Description of the method**
boolean	removeAll(BstSet<?> c) Removes from this set all of its elements that are contained in the specified collection.
E	ceiling(E e) Returns the least element in this set greater than or equal to the given element, or null if there is no such element.
E	floor(E e) Returns the greatest element in this set less than or equal to the given element, or null if there is no such element.
E	higher(E e) Returns the least element in this set strictly greater than the given element, or null if there is no such element.
E	last() Returns the last (highest) element currently in this set.
E	lower(E e) Returns the greatest element in this set strictly less than the given element, or null if there is no such element.
E	pollFirst() Retrieves and removes the first (lowest) element, or returns null if this set is empty.
E	pollLast() Retrieves and removes the last (highest) element, or returns null if this set is empty.
SortedSet<E>	copyOf(BstSet<? extends E> set) Returns a Set containing the elements of the given set

Possible practical exercises from the L2 optional part

1. Write an algorithm to check if the given binary tree is a binary search tree.
2. Write a method to convert the array into a balanced binary search tree.

3. Write a method to convert a binary search tree into a linked list.
4. Write a method to iterate through the tree along its levels.
5. Write a method to iterate through the tree in vertical layers from top to bottom
6. Print out the leaves of the binary search tree.
7. Given two nodes in a tree, find and print the shortest path between these nodes.
8. Find and print all paths of the binary search tree from root to leaf.
9. Given a number, determine whether a path exists in the binary search tree whose sum of node keys is equal to the given number.
10. Rewrite one of the basic tree operations without recursion.

Possible theoretical questions

1. What is the computational complexity of various operations implemented in the lab?
2. What are the advantages and disadvantages of a binary search tree and a balanced binary search tree?
3. Be able to explain the various nuances involved in the programming code.