Mandatory exercise 2
INF102

Deadline: 20th of October (23:59).

This assignment is an individual task, however you are allowed to collaborate and discuss solutions *as long as you do not share code* (see our policy on Collaboration and Cheating). Similarly, for tasks that are not answered with code, you may freely discuss your answers with your peers; but clearly, blindly transcribing answers from others is not allowed.

The assignment is graded on a scale from 0 to 80, and accounts for 10% of your final grade.

Organizational Instructions

First download the zipfile *Oblig2INF102(H19).zip* from files/Oblig 2 on mittuib. The zipfile contains the classes you are required to implement as well as other classes needed which contains methods for reading input, generating random numbers, and for timing your program.

The answers to code questions should go into the respective classes for each question (*PerfectBST.java* for Task 2, *MedianPQ.java* for Task 3, and *WordCount.java* for Task 4). Note that classes should not be placed in a package.

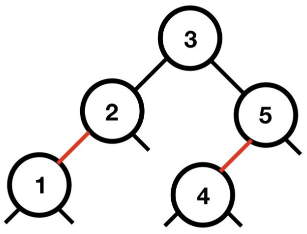
The answers to non-code questions should be submitted in a pdf named {yourid}.pdf, where yourid is your UiB SEBRA account id. For example, if your id is abc123 the pdf should be named abc123.pdf. You can answer questions in either Norwegian or English.

Every file (the pdf + sourcefiles) should be placed in a folder named {yourid} (without the {}). Do not use subfolders, all files should be placed on the first level of your folder. This directory should then be placed in a zipfile named {yourid}.zip. Finally this file should be handed in on mittuib.

If your submission does not follow these instructions (ie. we need to search for your code), you will be deducted up to 10 points.

Task 1 (20 points)

For each of the following input sequences, draw the associated left-leaning red-black binary search tree. Include the coloring of the edges in your figure. As an example, if the input was 4, 3, 1, 5, 2 you would draw:

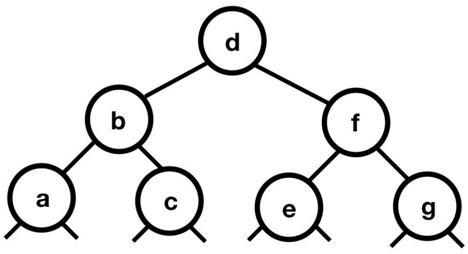


a) 1, 3, 5, 7, 6, 4, 2
b) 4, 6, 8, 3, 1, 5, 7, 2
c) 1, 2, 3, 4, 5, 6, 7
d) 7, 6, 5, 4, 3, 2, 1

Task 2 (20 points)

The class `BST.java` contains an implementation of a binary search tree. The class `PerfectBST.java` extends `BST.java` with one new constructor `PerfectBST(Key k[], Value v[])` that constructs a perfectly balanced binary tree using (`k[i] ,v[i]`) as (key, value) pairs. You can assume that `k[]` is sorted in ascending order and that `k[]` and `v[]` are of the same length. Your task is to implement this constructor in the given file. Your code should have linear running time measured in the number of pairs.

Example:
If `k[]` consists of `[a,b,c,d,e,f,g]` then you should build the following tree (value fields are not shown):



If there are N numbers in `k[]` and N is an even number then you can either choose element N/2 or N/2+1 as the root element. This also applies recursively when selecting subroots.

Task 3 (20 points)

The median of sequence of N numbers is the middle number if the numbers are sorted. If N is even then the median is the average of the two numbers in position N/2 and N/2+1 (if the numbers are sorted).

Example: The median of [1,5,2,3,7,6,4] is 4, while the median of [1,5,2,3,6,4] is 3.5.

Write a class `MedianPQ.java` that supports the two operations insert a floating point number and find median. Insert should run in <u>logarithmic time</u> and find median in <u>constant time</u>. You can use the two classes `MaxPQ.java` and `MinPQ.java` for this assignment.

You can use the class `MedianPQClient.java` to test the program. This program reads floating point values from standard input and inserts these into the `MedianPQ` data structure. For each inserted value the current median value is printed.

Example:

```
1.0
Median = 1.0
5.0
Median = 3.0
2.0
Median = 2.0
3.0
Median = 2.5
```

Task 4 (20 points)

The class `BST.java` contains an implementation of a binary search tree. Write a program `WordCount.java` that uses this to count the number of occurrences of words read from standard input. When the input has been read the program should output each distinct word in alphabetically ascending order, along with the number of times it occurs. If the input consists of N words of which M are distinct, then the program should run in average time N lg M (and not N log N).

Example: If the input is:

```
99 bottles of beer on the wall, 99 bottles of beer
```

Then the output should be:

```
99      2
beer    2
bottles 2
of      2
on      1
the     1
```

wall, 1

Appendix: Style Guide

The following rules applies to all source code that is handed in. These guidelines are loosely based on Google Java Style Guide, though with some minor differences.

Most important

- Make your code easy to read.
    - Comment tricky parts of the code.
    - Use descriptive and logical variable names and function names.
    - Break the code into appropriate functions for readability and code reuse; no function should ideally be more than 30 lines of code (with the exception of extremely monotone code, such as sanity tests).
    - Write javadoc comments for functions that are not self-explanatory
- All files should use UTF-8 character encoding.

Also important

- The only whitespace characters allowed are spaces and newlines (tabs are not allowed).
- Each indentation level increases by 4 spaces.
- No line may exceed 120 characters - lines exceeding this limit must be line-wrapped.
- Some exceptions, e.g. very long URL's that don't fit on one line.
- File name should be the same as the name of the class, written in UpperCamelCase.
- Function names, parameter names and variable names should be written in lowerCamelCase.
- Constants should be written in ALL_CAPS.
- No line breaks before open braces ({).
- Blank lines should be used sparingly, but can be used to separate logic blocks and to increase readability.