Koç University
COMP202
Data Structures and Algorithms
Assignment 3

Instructor: Barış Akgün
Responsible TA: İpek Kızıl
Due Date: April 6 2018, 23:59
Submission Trough: Blackboard
Relevant Programming Session: March 29 2018 17:30-18:45 at SOS180/CASEZ48

This programming assignment will test your knowledge and your implementation abilities of what you have learned in the Trees, Binary Search Trees and Priority Queues parts of the class.

This homework must be completed individually. Discussion about algorithms and data structures are allowed but group work is not. Any academic dishonesty, which includes taking someone else's code or having another person doing the assignment for you will not be tolerated. **By submitting your assignment, you agree to abide by the Koç University codes of conduct.**

Description

This assignment requires you to implement a *Binary Search Tree* to be used as a map data structure with a few additional operations. In addition, you are also asked to implement a *Array Based Heap* and modify a BST, both to be used as a *Priority Queue*. The assignment will only have a single mandatory phase, assigned right now. A bonus phase will follow soon.

You are going to implement the binary search tree which implements a binary search tree interface and a map interface. You are also given an adaptable priority queue interface. This is going to be implemented by an array-based heap and a binary search tree based priority queue, which also extends the previous binary search tree. The details of these interfaces will not be given here due to the comments in the files. **Feel free to contact us if anything is unclear**. You are additionally given additional classes that will help with the implementation.

As mentioned above, yhe code has comments about the assignment and the desired functionality. Majority of what you need to do is actually in these comments so make sure you read them carefully. You are provided with a compressed file which contains following files. Bold ones are the ones you need to modify and they are placed under the *code* folder, with the rest under *given*.

- *iMap.java*: This file defines a simple map interface. Even though you do not need to modify this, make sure you read the comments in detail. It is pretty straight forward.
- *iBinarySearchTree.java*: This file includes the `iBinaryTree` interface, which describes the desired binary search tree ADT. Even though you do not need to modify this, make sure you read the comments in detail.
- *Entry.java*: This file includes the `Entry` class that describes a *Key-Value* pair. Certain useful methods are implemented for you. This will form the basis of your binary tree nodes and be used in the priority queues. You should take a look at its functions but do not modify it.
- ***BinaryTreeNode.java***: This file includes the `BinaryTreeNode` class which is used in the binary tree as nodes. It extends the `Entry` class. You are free to design and implement it however you want.
- ***BinarySearchTree.java***: This is where one of the main data structures of the assignment, the `BinarySearchTree` class, should be implemented. It extends both the `iBinarySearchTree` and the `iMap` interfaces. You should implement this class and the `BinaryTreeNode` class in conjunction.

- *DefaultComparator.java*: This file implements the comparators that are used in the assignment. You do not need to touch this or even go over this unless you are curious.
- *TestBST.java*: A very simple program to test your binary search tree implementation. It is designed to be resemble a primitive autograder. **We might update this!**.
- *iAdaptablePriorityQueue.java*: This file includes the `iAdaptablePriorityQueue` interface, which describes an adaptable priority queue interface. As with the other interface files, make sure to go over the comments. There is one important point, its operations assume that both the keys and the values are unique. This is done to make the assignment easier but it is not very realistic. If you ever need something ike this in rel life, keep this in mind! Another point is that the adaptations require you to find the right entries. This is slightly different than the *Position* concept we have seen, i.e., to change keys or values we find the relevant entry but do not store indiviual entries.
- ***BSTBasedPQ.java***: This file defines the binary search tree based priority queue, namely the `BSTBasedPQ` class. It extends the `BinarySearchTree` class and implements the `iAdaptablePriorityQueue` interface. Do not try to create heap behavior but think about how to use a BST as a priority queue. Hint: There is a $O(log(n))$ way to get to the minimum element!
- ***ArrayBasedHeap.java***: This is where the other main data structures of the assignment, the `ArrayBasedHeap` class, should be implemented. It extendsthe `iAdaptablePriorityQueue` interface. We suggest that you first implement the array based binary tree functionality (no interface given but you can get inspiration from the available interfaces/classes), then add heap functionality. Make sure that this actually behaves as a heap and not a sorted list!
- *testPQ.java*: An implementation of an order-book based on your priority queue implementations. Feel free to ask about its implementation in person but we are not going to delve into it here. However, you can change the `testFolderName` variable to make the problem bigger. I have provided an excel sheet to help you compare your outputs with the desired ones. Everything should be self-explanatory but feel free to ask if needed.

Warning: The *not implemented yet* warnings are removed since it confused some of you. However, this means that it is up to you to make sure everything is implemented and not left empty!

Hint: If certain operations end up being $O(n)$, it is fine!

Bonus

You are asked to implement a phone book in the bonus part of the assignment. The phone book stores contact information and is efficiently searchable (In this context, efficient means not iterating over all the entries!) using two keys, contact name and contact number. In addition, it should be searchable by e-mail (not necessarily efficiently). We have gone over how to do it in the class on April 2^{nd} by using multiple maps. **You are only allowed to use your own tree implementations in this bonus part!**. My suggestion is for you to use two binary search trees, one for the contact name and the other for the contact number. All the rest are in the comments. The added files are:

- *ContactInfo.java*: This file has the `ContactInfo` class that you should store as your values. You do not need to touch this but should go over it.
- ***PhoneBook.java***: This file includes the `PhoneBook` class which you need to complete based on the description here and the comments in the file. To make things easier, you can assume that the name and numbers will be unique. Note that we have gone over how to handle non-unique keys in April 2^{nd}'s class. Other than only being allowed to use your own tree data structures, feel free to implement this class in any way you want.
- *TestPhoneBook.java*: The file containing the main function to test your phone-book implementation. We are not going to delve into it here and you will need to figure out how it works if needed. We have provided a text file named *bonus_output.txt* to help you compare. Feel free to create your own test files if the given ones are too big and complex to start with.

The input files for both testing the phonebook and the priority queues are generated randomly, based on certain rules. We have used you a mix of your names and surnames for the phonebook and created random phone numbers, e-mails and addresses. Please do not take any of the resulting names seriously.

Grading

We are likely not going to be able to provide an Autograder for you due to resource constraints. The responsible TA will grade your homework manually or with semi-automated tools. You will have a chance to object. In the event that there is an autograder, we are going to share it with you ASAP.

Still do not send any code that does not compile. You will be heavily penalized and might even get no credit.

Bonus

The bonus will also be graded semi-automatically or manually. It will be worth an extra 25%.

Submission

You are going to submit a compressed archive through the blackboard site. The file can have *zip*, *tar*, *rar*, *tar.gz* or *7z* format. This should extract to a folder with your student ID without the leading zeros. This folder should only contain files that were in **boldface**. Other files will be deleted and/or overwritten.

Submission Instructions

- You are going to submit a compressed archive through the blackboard site.
- The file can have *zip*, *tar*, *tar.gz* or *7z* format.
- This compressed file should extract to a folder with your student identification number with the two leading zeros removed which should have 5 digits. Multiple folders (apart from operating system ones such as MACOSX or DS Store) greatly slows us down and as such will result in penalties
- Code that does not compile will receive 0 credits!
- Do not trust the way that your operating system extracts your file. They will mostly put the contents inside a folder with the same name as the compressed file. We are going to call a program (based on your file extension) from the command line. The safest way is to put everyting inside a folder with your ID, then compress the folder and give it your ID as its name.
- One advice is after creating the compressed file, move it to your desktop and extract it. Then check if all the above criteria is met.
- Once you are sure about your assignment and the compressed file, submit it through Blackboard.
- **DO NOT SUBMIT CODE THAT DOES NOT TERMINATE OR THAT BLOWS UP THE MEMORY.**

Let us know if you need any help with setting up your compressed file. This is very important. We will put all of your compressed files into a folder and run multiple scripts to extract, cleanup, grade and do plagiarism checking. If you do not follow the above instructions, then scripts might fail. This will lead you to get a 0. Such structured submissions greatly help manual grading as well.