UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA
Denkleiers • Leading Minds • Dikgopolo tša Dihlalefi

Department of Computer Science
Faculty of Engineering, Built Environment & IT
University of Pretoria

COS212 - Data structures and algorithms

Practical 6 Specifications - Binary Heap

Release date: 22-04-2024 at 06:00
Due date: 26-04-2024 at 23:59
Total marks: 140

Contents

1 General Instructions

- ***Read the entire assignment thoroughly before you start coding.***
- This assignment should be completed individually; no group effort is allowed.
- **To prevent plagiarism, every submission will be inspected with the help of dedicated software.**
- Be ready to upload your assignment well before the deadline, as no extension will be granted.
- You may not import any of the built-in Java data structures. Doing so will result in a zero mark. You may only use native 1-dimensional arrays where applicable. If you require additional data structures, you will have to implement them yourself.
- If your code does not compile, you will be awarded a mark of zero. Only the output of your program will be considered for marks, but your code may be inspected for the presence or absence of certain prescribed features.
- If your code experiences a runtime error, you will be awarded a zero mark. Runtime errors are considered unsafe programming.
- Read the entire specification before you start coding.
- **Ensure your code compiles with Java 8**
- The usage of ChatGPT and other AI-Related software is strictly forbidden and will be considered as plagiarism.

2 Plagiarism

The Department of Computer Science considers plagiarism a serious offence. Disciplinary action will be taken against students who commit plagiarism. Plagiarism includes copying someone else's work without consent, copying a friend's work (even with consent), and copying material (such as text or program code) from the Internet. Copying will not be tolerated in this course. For a formal definition of plagiarism, the student is referred to http://www.library.up.ac.za/plagiarism/index.htm (from the main page of the University of Pretoria site, follow the Library quick link, and then choose the Plagiarism option under the Services menu). **If you have any form of question regarding this, please ask one of the lecturers to avoid any misunderstanding.** Also note that the OOP principle of code reuse does not mean that you should copy and adapt code to suit your solution.

3 Outcomes

On completion of this practical, you will have gained experience with the following:

- Red-Black Trees, including:
  - Inserting data into Red-Black Trees and subsequent balancing rotations
  - Removing data from Red-Black Trees and subsequent balancing rotations
  - Validating Red-Black Trees in terms of the Red-Black tree properties

4 Introduction

4.1 Binary Heaps

A binary heap is a specialized tree-based data structure that satisfies the heap property. It is commonly used to implement priority queues, which are abstract data types designed to manage a collection of elements subject to priority ordering. The essence of the binary heap lies in its ability to allow quick retrieval and manipulation of the highest (or lowest) priority element.

In a binary heap, the data is organized into a binary tree. This tree is both a complete binary tree, meaning it is filled at every level except possibly the last level, which is filled from left to right, and it maintains the heap property: each node is either greater than or equal to (in a max heap) or less than or equal to (in a min heap) its children. This structure ensures that the root of the tree always contains the highest (or lowest) priority item, making a binary heap an efficient means to implement a priority queue.

4.2 Template Method Design Pattern

The given files are intended to guide you towards an implementation of the template method design pattern. The template method design pattern is a behavioral design pattern that defines the program skeleton of an algorithm in a method, referred to as the template method, which defers some steps to subclasses. This allows subclasses to override certain steps of the algorithm without changing its structure. The provided files serve as a blueprint for implementing this pattern by allowing you to outline the core algorithmic steps in a base class, while allowing customization of specific details (specifically the `compare` method) in derived classes. For more information on the template method design pattern see `https://www.cs.up.ac.za/cs/lmarshall/TDP/Notes/_Chapter3_TemplateMethod.pdf` by Dr Linda Marshall.

5 Tasks

You are tasked with implementing a binary heap. You have been given the skeletons for `AbstractHeap`, `MinHeap` and `MaxHeap` classes. Your implementation must adhere to the given class diagram (Figure 1).

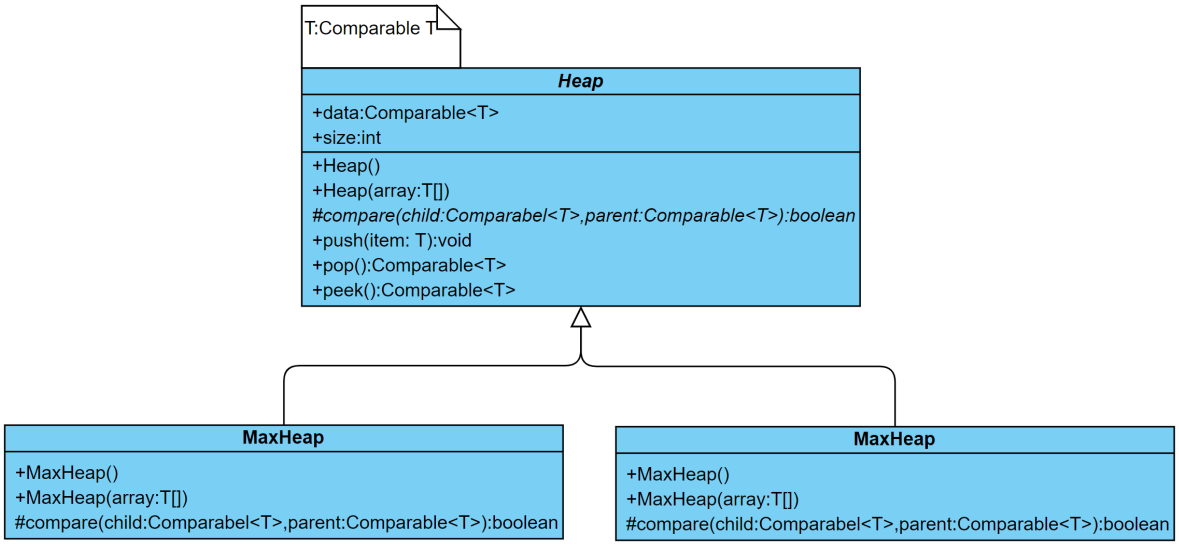

Figure 1: Heap hierarchy UML class diagram

5.1 Heap<T extends Comparable<T> >

- Members:
  - data: Comparable<T>[]
    - The array that holds all the elements in the heap.
  - size: int
    - The number of elements in the heap
    - This is not necessarily equal to the length of the data array.
    - The length of the data array may be larger than the number of elements in the heap. However, the length of the data array may never be smaller than the number of elements in the heap.
- Methods:
  - Heap()
    - Default constructor to initialize the heap with a data array of length 2 and a size of 0.
  - Heap(array:T[])
    - A constructor which takes an array of type T
    - This constructor should use `FloydAlgorithm` to turn the given array into a valid heap.
    - You may assume the array is never empty.
  - compare(child:Comparable<T>,parent:Comparable<T>):boolean
    - Returns true if the child and the parent should be swapped and false if not.
    - This should return false if the parent and the child are equal
    - The implementation of this method will differ between `MinHeap` and `MaxHeap`. Therefore the implementation is left to the sub-classes using the template method design pattern.
  - push(data: T): void
    - Adds the given data to the heap and then maintains the heap property by performing the necessary swaps.
    - If the data array is full before an insert (`size==data.length`) then the contents of the current data array should be copied to a new array of twice the size of the current one. This new array should then be used as the data array.
  - pop(): Comparable<T>
    - Removes the root of the heap which should be the maximum or minimum element and returns it.
    - After a pop the heap property should be restored and the next largest (or smallest) element should become the root of the heap.
  - peek(): Comparable<T>
    - Returns the root of the heap which should be the maximum or minimum element without removing it.
  - toString():String
    - Returns a string representation of the tree
    - This has been implemented for you. Please do not change it as we may test against it on FitchFork.

5.2 MaxHeap<T extends Comparable<T> >

- Methods
  - MaxHeap()
    - The default constructor for a max-heap. It calls the corresponding base class constructor.
  - MaxHeap(array:T[])
    - The parameterised constructor for a max-heap. It calls the corresponding base class constructor.
  - compare(child:Comparable<T>,parent:Comparable<T>):boolean
    - Returns true if the child and the parent should be swapped and false if not.
    - This should return false if the parent and the child are equal
    - The implementation in this method should be specific to a max-heap. Therefore true should be returned if the child is greater than the parent.

5.3 MinHeap<T extends Comparable<T> >

- Methods
  - MinHeap()
    - The default constructor for a min-heap. It calls the corresponding base class constructor.
  - MinHeap(array:T[])
    - The parameterised constructor for a min-heap. It calls the corresponding base class constructor.
  - compare(child:Comparable<T>,parent:Comparable<T>):boolean
    - Returns true if the child and the parent should be swapped and false if not.
    - This should return false if the parent and the child are equal
    - The implementation in this method should be specific to a min-heap. Therefore true should be returned if the child is smaller than the parent.

6 Testing

As testing is a vital skill that all software developers need to know and be able to perform daily, approximately 10% of the assignment marks will be assigned to your testing skills. To do this, you will need to submit a testing main (inside the Main.java file) that will be used to test an Instructor-provided solution. You may add any helper functions to the main.cpp file to aid your testing. In order to determine the coverage of your testing the jacoco tool. The following set of commands will be used to run jacoco:

```
javac *.java
rm -Rf cov
mkdir ./cov
java -javaagent:jacocoagent.jar=excludes=org.jacoco.*,destfile=./cov/output.exec
    -cp ./ Main
mv *.class ./cov
java -jar ./jacococli.jar report ./cov/output.exec --classfiles ./cov --html
    ./cov/report
```

This will generate output which we will use to determine your testing coverage. The following coverage ratio will be used:

$$\frac{\text{number of lines executed}}{\text{number of source code lines}}$$

and we will scale this ratio according to the size of the class.
The mark you will receive for the testing coverage task is determined using table 1:

Coverage ratio range	% of testing mark
0%-5%	0%
5%-20%	20%
20%-40%	40%
40%-60%	60%
60%-80%	80%
80%-100%	100%

Table 1: Mark assignment for testing

Note the top boundary for the Coverage ratio range is not inclusive except for 100%. Also, note that only the function stipulated in this specification will be considered to determine your mark. Remember that your main will be testing the instructor-provided code and as such it can only be assumed that the functions outlined in this specification are defined and implemented.

7 Upload checklist

The following files should be in the root of your archive

- Main.java
- Heap.java
- MaxHeap.java
- MinHeap.java
- Any other .java files that your solution requires
- Any .txt files you use for testing

8 Allowed libraries

- None, sorry :(

9 Submission

You need to submit your source files on the FitchFork website (https://ff.cs.up.ac.za/). All methods need to be implemented (or at least stubbed) before submission. Place the above-mentioned files in a zip named uXXXXXXXX.zip where XXXXXXXX is your student number. Your code must be able to be compiled with the Java 8 standard.

For this practical, you will have 5 upload opportunities and your best mark will be your final mark. Upload your archive to the appropriate slot on the FitchFork website well before the deadline. **No late submissions will be accepted!**