Algorithms and Data Structures (ADS2)

Assessed Exercise 2

This exercise is for submission using Moodle and counts for 10% of the total assessment mark for this course.

This exercise is worth a total of 30 points.

The deadline for submission is Monday 21 March 2022 at 4:30pm.

Exercise

This exercise has two parts. The first involves implementing in Java the Priority Queue abstract data type using two different data structures. In the second part you are asked to use a Priority Queue to implement an algorithm for a practical problem. Note you are not allowed to rely on Java library classes in your implementation.

Submission

Submit a zip archive containing the Java sources of your implementations **and** a short (max 3 pages) report describing what you have done in each part of the exercise. Your report should include a heading stating your full name and matriculation number and clear instructions on how to run your code.

Make sure the report is in pdf format and your sources are not password protected.

Part 1

The *Min-priority Queue* is an abstract data type (ADT) for maintaining a collection of elements, each with an associated value called a *key*. The ADT supports the following operations:

- INSERT(Q,x): insert the element *x* into the queue *Q*.
- MIN(Q): returns the element of *Q* with the smallest key.
- EXTRACT-MIN (Q): removes and returns the element of *Q* with the smallest key.

Implement in Java the Min-priority Queue ADT defined above using

a) an array based binary heap similar to the one introduced in Lecture 8, [6]
b) a binary search tree. [6]

Observe that the ADT implementation operations should be in the form `q.insert(x)`, `q.min()`, etc. Explain in the report your implementation, noting the running time (using big-Oh notation) of each operation in both implementations.

c) What are the worst-case running times of the three ADT operations when the underlying BST is self-balancing? Briefly explain your answer. [3]
d) Implement and extension of BST that allows MIN and EXTRACT-MIN operations in O(1). Briefly describe your implementation in the report. Hint: maintain an extra pointer attribute in each node. [6]

Part 2

Implement an efficient algorithm in Java to solve the following problem:

> *You are given n ropes of different lengths (expressed as integers), and you are asked to connect them to form a single rope with the minimum cost. The cost of connecting two ropes is equal to the sum of their lengths.*

Given a sequence of rope lengths, the expected outputs are a sequence of rope connection operations and the total cost. Use one of your implementations of the Min-priority Queue ADT in your solution.

a) Give a brief description of your implementation, explaining why a priority queue is needed for an efficient algorithm. [8]

b) What is the output for this instance of the problem 4,8,3,1,6,9,12,7,2? [1]