# *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 20 points.**

***The deadline for submission is Monday 24 March 2025 at 4:30pm.***

## Exercise

This exercise has two parts. In the first part, you are asked to implement in Java an ADT, and define an efficient algorithm to solve a practical problem in the second part.

## Submission

Submit the Java sources of your implementations and a short (maximum 3 pages) report briefly describing what you have done in each part of the exercise. Your report should include clear instructions on how to run your code.

### Part 1

The *Dynamic Set* is an abstract data type (ADT) that can store distinct elements, without any particular order. There are five main operations in the ADT:

- ADD(S,x): add element x to S, if it is not present already
- REMOVE(S,x): remove element x from S, if it is present
- IS-ELEMENT(S,x): check whether element x is in set S
- SET-EMPTY(S): check whether set S has no elements
- SET-SIZE(S): return the number of elements of set S

Additionally, the Dynamic Set ADT defines the following set-theoretical operations:

- UNION(S,T): return the union of sets S and T
- INTERSECTION (S,T): return the intersection of sets S and T
- DIFFERENCE(S,T): returns the difference of sets S and T
- SUBSET(S,T): check whether set S is a subset of set T

a) Implement in Java the Dynamic Set ADT defined above using a binary search tree. Explain in the report your implementation, noting the running time of each operation. You can use a self-balancing binary tree but no extra marks will be awarded. Also, you are not allowed to rely on Java library classes in your implementation. [8]

b) A naïve definition of UNION(S,T) for a BST-based implementation of the Dynamic Set ADT consists in taking all elements of BST S one by one, and inserting them into BST T. Describe in the report an implementation with a better running time. [2]

### Part 2

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.

a) 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 your implementations of the Min-priority Queue ADT in your solution. [7]

b) Give a brief description of your implementation, explaining why a priority queue is needed for an efficient algorithm. [2]

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