Exercise 1: Priority Queues

Consider the following priority queue stored in an array:

\[
H = [(3, L), (10, D), (8, E), (12, C), (13, B), (23, R), (9, F), (17, S), (14, M)]
\]

Execute the following operations on \( H \): \( H.insert((7, N)) \), \( H.deleteMin() \), \( H.changeKey((13, B), 9) \).

Write down \( H \) after each operation including the repairing process. It may help if you draw \( H \) as a binary tree.

Exercise 2: Amortized Analysis

Consider the data structure \textit{stack} in which elements can be stored in a ‘last in first out’ manner. For a stack \( S \) we have the following operations:

- \( S.push(x) \) puts element \( x \) onto \( S \).
- \( S.pop() \) deletes the topmost element of \( S \). Calling \( pop \) on an empty stack generates an error.
- \( S.multipop(k) \) removes the \( k \) top objects of \( S \), popping the entire stack if \( S \) contains fewer than \( k \) objects.

Assume the costs of \( S.push(x) \) and \( S.pop() \) are 1 and the cost of \( S.multipop(k) \) is \( \min(k, s) \) where \( s \) is the current number of elements in \( S \).

Use the bank account paradigm to show that we can assign all three operations constant amortized costs.