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 \( H \): \( H.\text{insert}(7, N) \), \( H.\text{deleteMin}() \), \( H.\text{changeKey}(13, B, 9) \). Write down \( H \) after each operation including the repairing process. It may help if you draw \( H \) as a binary tree.

**Sample Solution**

After \( H.\text{insert}(7, N) \):

\[ H = [(3, L), (7, N), (8, E), (12, C), (10, D), (23, R), (9, F), (17, S), (14, M), (13, B)] \]

After \( H.\text{deleteMin}() \):

\[ H = [(7, N), (10, D), (8, E), (12, C), (13, B), (23, R), (9, F), (17, S), (14, M)] \]

After \( H.\text{changeKey}(13, B, 9) \):

\[ H = [(7, N), (9, B), (8, E), (12, C), (10, D), (23, R), (9, F), (17, S), (14, M)] \]

Exercise 2: Amortized Analysis

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

- \texttt{S.push}(x) puts element \( x \) onto \( S \).
- \texttt{S.pop()} deletes the topmost element of \( S \). Calling \texttt{pop} on an empty stack generates an error.
- \texttt{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 \texttt{S.push}(x) and \texttt{S.pop()} are 1 and the cost of \texttt{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.
Sample Solution

Define the amortized costs of the operations as follows:

- \text{S.push}(x) \quad 2
- \text{S.pop()} \quad 0
- \text{S.multipop}(k) \quad 0

For a sequence of \( n \) operations let be \( c_i \) the actual cost and \( a_i \) the amortized cost of operation \( i \leq n \). The total actual costs equals the number of \text{push} operations plus the number of \text{pop} operation, including calls within \text{multipop}. But there can be at most as many \text{pop} operations as \text{push} operations when the stack is initially empty, so the actual costs are at most twice the number of \text{push} operations, i.e.,

\[
\sum_{i=1}^{n} c_i \leq 2 \sum_{i=1}^{n} a_i.
\]