Implementation of a priority queue with a heap

Variables

heap: array of items
size: integer
invariant: *heap* represents a heap, *heap*[1], ..., *heap*[*size*] are the items of the priority queue.

Initialization

size $\leftarrow$ 0

Algorithms

size():
output: size of priority queue
return *size*

isEmpty():
output: priority queue is empty?
return *size* = 0

bubbleUp(*index*):
input: level number of item to be bubbled up
postcondition: item with level number *index* has been bubbled up
if *index* $\neq$ 1 **then**
    parent $\leftarrow$ $\lfloor$ *index* / 2 $\rfloor$
    if key of *heap*[*index*] < key of *heap*[*parent*] **then**
        swap items of *heap*[*index*] and *heap*[*parent*]
        bubbleUp(*parent*)

bubbleDown(*index*):
input: level number of item to be bubbled down
postcondition: item with level number *index* has been bubbled down
if *heap*[*index*] is not a leaf **then**
    child $\leftarrow$ level number of child of *heap*[*index*] with smallest key
    if key of *heap*[*index*] > key of *heap*[*child*] **then**
        swap items of *heap*[*index*] and *heap*[*child*]
        bubbleDown(*child*)

insertItem(*key*, *element*):
precondition: priority queue is not full
postcondition: item (*key*, *element*) has been inserted in the priority queue
input: item to be inserted
size $\leftarrow$ *size* + 1
heap[*size*] $\leftarrow$ (*key*, *element*)
bubbleUp(*size*)

minElement():
precondition: priority queue is nonempty
output: element with smallest key in priority queue
return element of *heap*[1]

minKey():
precondition: priority queue is nonempty
output: smallest key in priority queue
return key of *heap*[1]

removeMinElement():
precondition: priority queue is nonempty
postcondition: item of returned element has been removed from the priority queue
output: element with smallest key in priority queue
element $\leftarrow$ element of *heap*[1]
swap items of *heap*[*size*] and *heap*[1]
size $\leftarrow$ *size* $-$ 1
bubbleDown(1)
return *element*