```
# Tora De Boer
# toboe19@sdu.student.dk
# Camilla Marie Bach
# cabac19@sdu.student.dk

import math

def min_heapify(A, i):
    """Method that restores the properties of a min heap by
recursively calling itself on the list"""
    l = left(i)
    r = right(i)
    # Checks if i's left child is larger than the value of i and a
valid node in the list and stores the smallest values index.
    if l <= len(A)-1 and A[l] < A[i]:
        smallest = l
    else:
        smallest = i
    # Checks if i's right child is in the list and larger than the
value at index i. Stores the smallest value.
    if r <= len(A)-1 and A[r] < A[smallest]:
        smallest = r
    # If i is not the smallest, then values are swapped and calls
itself on the rest of the list, to restore min heap properties.
    if smallest != i:
        temp = A[smallest]
        A[smallest] = A[i]
        A[i] = temp
        min_heapify(A, smallest)

def extractMin(A):
    """Methods that extracts and returns the minimum value in the
heap and restores min heap properties"""
    min = A[0]
    # Saves the last value of the list in the root node.
    A[0] = A[len(A)-1]
    # Removes the last value from the last node. Calls min_heapify
to restore min heap properties.
    A.pop(len(A)-1)
    min_heapify(A, 0)
    return min

def heap_min(A):
    """Method that returns the minimum value, stored in the
rootnode"""
    return A[0]

def insert(A, e):
    """Method that inserts an element to the end of the list A, then
swaps indeces around to restore the heap"""
    # Appends the element e on the last index in the list.
    A.append(e)
    i = len(A)-1
    # Swaps indexes around, if there are more elements than the root
```

```
node and restores min heap properties.
    while i > 0 and A[parent(i)] > A[i]:
        temp = A[parent(i)]
        A[parent(i)] = A[i]
        A[i] = temp
        i = parent(i)

def build_min_heap(A):
    """Method that builds a min heap by recursively calling
min_heapify on an unordered list"""
    # Calculates the middle index, rounded down if necessary.
    m = math.floor((len(A)-1) / 2)
    # Calls min_heapify on the list from the middle index and down
to the start.
    for i in range(m, -1, -1):
        min_heapify(A, i)

def parent(i):
    """Getter for the parent node of i"""
    return math.floor((i-1)/2)

def left(i):
    """Getter for the left child node of i"""
    return 2*i + 1

def right(i):
    """Getter for the right child node of i"""
    return 2*i + 2
```