from priority\_queue\_base import PriorityQueueBase  
from positional\_list import PositionalList  
  
class sortedPriorityQueue(PriorityQueueBase): # base class defines \_Item  
 *"""A min-oriented priority queue implemented with an unsorted list."""* #----------------------------- nonpublic behavior -----------------------------  
  
  
  
 #------------------------------ public behaviors ------------------------------  
 def \_\_init\_\_(self):  
 *"""Create a new empty Priority Queue."""* self.\_data = PositionalList()  
  
 def \_\_len\_\_(self):  
 *"""Return the number of items in the priority queue."""* return len(self.\_data)  
  
 def add(self, key, value):  
 *"""Add a key-value pair."""* newest = self.\_Item(key,value)  
 walk = self.\_data.last()  
 while walk is not None and newest < walk.element():  
 walk = self.\_data.before(walk)  
 if walk is None:  
 self.\_data.add\_first(newest)  
 else:  
 self.\_data.add\_after(walk,newest)  
  
 def min(self):  
 *"""Return but do not remove (k,v) tuple with minimum key.  
  
 Raise Empty exception if empty.  
 """* if self.is\_empty():  
 raise Exception("Priority queue is empty")  
 p = self.\_data.first()  
 item = p.element()  
 return (item.\_key,item.\_value)  
  
 def remove\_min(self):  
 *"""Remove and return (k,v) tuple with minimum key.  
  
 Raise Empty exception if empty.  
 """* if self.is\_empty():  
 raise Exception("Priority Queue is empty.")  
 item = self.\_data.delete(self.\_data.first())  
 return (item.\_key)  
  
 def \_\_iter\_\_(self):  
 *"""Generate iteration of the map's keys."""* for item in self.\_data:  
 yield item # yield the KEY  
 def insertionSort(self,A):  
 for i in A:  
 self.add(i,i)  
 for i in range(len(A)):  
 A[i] = self.remove\_min()  
  
if \_\_name\_\_ == '\_\_main\_\_':  
  
 p = sortedPriorityQueue()  
 A = [6,2,25,88,1,15]  
 p.insertionSort(A)  
 for i in A:  
 print(i)