```
import random

def g_parent(i):
    return (i-1)/2

def g_left(i):
    return i*2 + 1

def g_right(i):
    return i*2 + 2

def check_sorted(s):
    prev = s[0]
    for x in s:
        if x < prev:
            print 'not sorted!'
            return
        prev = x
    #print 'sorted!'

def check_heap(h, i=0):
    if i >= len(h):
        return True
    l = g_left(i)
    r = g_right(i)
    if (l < len(h) and h[i] > h[l]):
        return False
    if (r < len(h) and h[i] > h[r]):
        return False
    return check_heap(h, l) and check_heap(h, r)

def swap(h, i, j):
    temp = h[i]
    h[i] = h[j]
    h[j] = temp

class heap:
    def __init__(self, arr = None):
        self.h = arr or []
        self.index = {} #item --> index
        if arr:
            #fill in inital index
            for i in range(0, len(self.h)):
                self.index[self.h[i]] = i
            self.heapify()

    #heapify array
    def heapify(self):
        #startin from n/2 -1index to 0
        for i in range(len(self.h)/2 - 1, -1, -1):
            self.sift_down(i)

    def size(self):
        return len(self.h)

    def swap_index(self, i, j):
        self.index[self.h[i]] = j
        self.index[self.h[j]] = i
```

```
#h is an array
def insert(self, x):
    self.h.append(x)
    self.index[x] = len(self.h)-1
    self.sift_up(len(self.h)-1)

def get_min(self):
    m = self.h[0]
    self.delete(0)
    return m

#delete element at position i
def delete(self, i):
    del self.index[self.h[i]]
    if i == len(self.h)-1:
        self.h.pop()
    else:
        self.h[i] = self.h.pop()
        self.index[self.h[i]] = i
        p = g_parent(i)
        if p >= 0 and self.h[i] < self.h[p]:
            self.sift_up(i)
        else:
            self.sift_down(i)

def delete_item(self, item):
    index = self.index[item]
    self.delete(index)


#sift up at i if necessary
def sift_up(self, i):
    p = g_parent(i)
    while(p >= 0 and self.h[i] < self.h[p]):
        self.swap_index(i, p)
        swap(self.h, i, p)
        i = p
        p = g_parent(i)

#sift down at i if necessary
def sift_down(self, i):
    right = g_right(i)
    while(right < len(self.h)):
        left = g_left(i)
        min_ind = min((self.h[left],left),(self.h[right],right))[1]
        if self.h[i] > self.h[min_ind]:
            self.swap_index(i, min_ind)
            swap(self.h, i, min_ind)
            i = min_ind
            right = g_right(i)
        else:
            break
    #check remaining possible left child
    left = g_left(i)
    if left < len(self.h) and self.h[i] > self.h[left]:
        self.swap_index(i, left)
```

```
            swap(self.h, i, left)

    def get_index(self, x):
        return self.index[x]

    def check_index(self):
        for i in range(0, len(self.h)):
            if self.index[self.h[i]] != i:
                print 'index faliling!'
                print 'h[%d]= %s has index %d '% (i, self.h[i],
self.index[self.h[i]])
                print 'h length is', len(self.h)
                print 'index is',self.index
                print '------------'
                print 'h is', [(self.h[i],i) for i in range(0, len(self.h))]
                return False
        return True

    def __str__(self):
        return str(self.h)

def test():
    MAX = 10
    h = heap()
    #s = []

    #check heapify
    for t in range(0, 100):
        print 't = %d'%t
        arr = []
        lst = []
        for i in range(0, MAX):
            arr.append(random.randint(-MAX, MAX))
        haha = heap(arr)
        if not check_heap(haha.h):
            print 'heapify failed at trial %d'%t
        while (haha.size() > 0):
            lst.append(haha.get_min())
        check_sorted(lst)




    for i in xrange(0, MAX):
        r = random.randint(1,2*MAX)
        r2 = random.randint(1,2*MAX)
        h.insert((r, r2))
        if not check_heap(h.h):
            print 'heap fail at insert'

        #if not h.check_index():
            #print 'index fail at insert'
            #return
        #insert(h2, random.randint(1,2*MAX))
    #print 'unsorted is',h

    #CHECK delete
```

```
    #c = 0
    #while(h.size() > 0):
        #c += 1
        #r = random.randint(0, h.size()-1)
        #print 'deleting h[%d] = %s' % (r,h.h[r])
        #print 'current h is',h.h
        #h.delete(r)
        #if not check_heap(h.h):
            #print 'heap fail at delete',c
            #if c > 10:
                #return

    #CHECK delete item
    c = 0
    print 'init h is',h.h
    while(h.size() > 0):
        c += 1
        r = random.randint(0, h.size()-1)
        print 'deleting h[%d] = %s' % (r,h.h[r])
        h.delete_item(h.h[r])
        if not check_heap(h.h):
            print 'heap fail at delete',c
            if c > 10:
                return
        print 'after delete h is',h.h


    #CHECK extract min
    #while(h.size() > 0):
        #s.append(h.get_min())
        #if not check_heap(h.h):
            #print 'heap fail at extract min'
        #if not h.check_index():
            #print 'index fail at getmin'
            #return
    #check if s is sorted
    #print 'check if sorted'
    #print 'result is',s
    #check_sorted(s)


if __name__ == '__main__':
    test()
```