```
!date
```

```
Fri Apr 22 03:20:27 UTC 2022
```

Please run the above line to refresh the date before your submission.

CSCI-SHU 210 Data Structures

Recitation 11 Heap/Priority Queue

You should work on the tasks as written in the worksheet.

- For students who have recitation on Wednesday, you should submit your solutions by **Apr 22th** Friday 11:59pm.
- For students who have recitation on Thursday, you should submit your solutions by **Apr 23th** Saturday 11:59pm.
- For students who have recitation on Friday, you should submit your solutions by **Apr 24th** Sunday 11:59pm.

Name: Peter Yuncheng Yao 姚允成

NetID: yy4108

Please submit the following items to the Gradescope:

- Your Colab notebooklink (by clicking the Share button at the top-right corner of the Colab notebook, share to anyone)
- The printout of your run in Colab notebook in pdf format
- No late submission is permitted. All solutions must be from your own work. Total points of the assignment is 100.

1. Heap Priority Queue

This is the original min-heap. Modify it such that it becomes a max-heap.

```
class HeapPriorityQueue:
    """A min-oriented priority queue implemented with a

    class _Item:
        """Lightweight composite to store priority queu
        __slots__ = '_key', '_value'
```

```
 7
 8         def __init__(self, k, v):
 9             self._key = k
10             self._value = v
11
12         def __lt__(self, other):
13             return self._key < other._key    # compare
14
15         def __repr__(self):
16             return '({0},{1})'.format(self._key, self._
17
18     #------------------------------ nonpublic behaviors
19     def _parent(self, j):
20         return (j-1) // 2
21
22     def _left(self, j):
23         return 2*j + 1
24
25     def _right(self, j):
26         return 2*j + 2
27
28     def _has_left(self, j):
29         return self._left(j) < len(self._data)     # in
30
31     def _has_right(self, j):
32         return self._right(j) < len(self._data)    # in
33
34     def _swap(self, i, j):
35         """Swap the elements at indices i and j of arra
36         self._data[i], self._data[j] = self._data[j], s
37
38     def _upheap(self, j):
39         parent = self._parent(j)
40         if j > 0 and self._data[j] > self._data[parent]
41             self._swap(j, parent)
42             self._upheap(parent)             # recur at
43
44     def _downheap(self, j):
45         if self._has_left(j):
46             left = self._left(j)
47             big_child = left                # although r
```

```
48              if self._has_right(j):
49                  right = self._right(j)
50                  if self._data[right] > self._data[left]
51                      big_child = right
52              if self._data[big_child] > self._data[j]:
53                  self._swap(j, big_child)
54                  self._downheap(big_child)    # recur at
55
56      # ## Bottom_Up Heap Construction ##
57      def _heapify(self):
58          start = self._parent(len(self)-1)
59          for j in range(start, -1, -1):
60              self._downheap(j)
61
62      #------------------------------ public behaviors --
63      # def __init__(self):
64      #     """Create a new empty Priority Queue."""
65      #     self._data = []
66
67      # This part is only for testing _heapify function #
68      def __init__(self, contents=[]):
69          """Create a new empty Priority Queue."""
70          self._data = [self._Item(k,v) for k,v in conten
71          if len(self._data)>1:
72              self._heapify()
73
74      def __len__(self):
75          """Return the number of items in the priority q
76          return len(self._data)
77
78      def is_empty(self):
79          return len(self) == 0
80
81      def add(self, key, value):
82          """Add a key-value pair to the priority queue."
83          self._data.append(self._Item(key, value))
84          self._upheap(len(self._data) - 1)             #
85
86      def min(self): # max(self) for the task
87          """Return but do not remove (k,v) tuple with mi
88
```

```
        Raise Empty exception if empty.
        """
        if self.is_empty():
            raise Exception('Priority queue is empty.')
        item = self._data[0]
        return (item._key, item._value)

    def remove_min(self): # remove_max(self) for the ta
        """Remove and return (k,v) tuple with minimum k

        Raise Empty exception if empty.
        """
        if self.is_empty():
            raise Exception('Priority queue is empty.')
        self._swap(0, len(self._data) - 1)           #
        item = self._data.pop()                      #
        self._downheap(0)                            #
        return (item._key, item._value)

```

```
heap = HeapPriorityQueue()
import random
for i in range(10):
  heap.add(random.randint(0, 20), "happy_final!")
print(heap._data)

for i in range(10):
  print("Removing from heap:", heap.remove_min()[0])
```

```
[(19,happy_final!), (19,happy_final!), (15,happy_final!), (16,happy_final!), (
Removing from heap: 19
Removing from heap: 19
Removing from heap: 17
Removing from heap: 16
Removing from heap: 15
Removing from heap: 14
Removing from heap: 13
Removing from heap: 9
Removing from heap: 5
Removing from heap: 0
```

```
## Test _heapify ##
contents=[(16,"Beijing"),(9,"Hangkong"),(15,"Shenzhen")
          (8,"Chengdu"),(11,"Chongqing"),(2,"Suzhou")]
```

```
heap = HeapPriorityQueue(contents)
print(heap._data)
```

```
[(16,Beijing), (9,Hangkong), (15,Shenzhen), (7,Guangzhou), (8,Chengdu), (11,Ch
```