CS 519-005, Algorithms (MS/MEng-level), Winter 2018
HW4 - Priority Queue and Heaps

Due via submit program on Monday Feb 5, 11:59pm.
No late submission will be accepted.

Need to submit: report.txt, nbest.py, kmergesort.py, datastream.py.
DO _NOT_ ZIP YOUR SUBMISSION.
datastream.py will be graded for correctness (1%).

Textbooks for References:
[1] CLRS Ch. 6
[2] Python heapq module

0. There are two methods for building a heap from an unsorted array:

```
   (1) insert each element into the heap  --- O(nlogn)
   (2) heapify (top-down)                 --- O(n)
```

   (a) Derive these time complexities.
   (b) Use a long list of random numbers to show the difference in time. (Hint: random.shuffle)
   (c) What about sorted or reversely-sorted numbers?

1. (taken from my first paper: see "Algorithm 1" in Huang and Chiang (2005).)

   Given two lists A and B, each with n integers, return
   a sorted list C that contains the smallest n elements from AxB:

```
     AxB = { (x, y) | x in A, y in B }
```

   i.e., AxB is the Cartesian Product of A and B.

```
   ordering:  (x,y) < (x',y') iff. x+y < x'+y' or (x+y==x'+y' and y<y')
```

   You need to implement three algorithms and compare:

   (a) enumerate all n^2 pairs, sort, and take top n.
   (b) enumerate all n^2 pairs, but use qselect from hw1.
   (c) Dijkstra-style best-first, only enumerate O(n) (at most 2n) pairs.
       Hint: you can use Python's heapq module for priority queue.

   Q: What are the time complexities of these algorithms?

```
   >>> a, b = [4, 1, 5, 3], [2, 6, 3, 4]
   >>> nbesta(a, b)   # algorithm (a), slowest
   [(1, 2), (1, 3), (3, 2), (1, 4)]
   >>> nbestb(a, b)   # algorithm (b), slow
   [(1, 2), (1, 3), (3, 2), (1, 4)]
   >>> nbestc(a, b)   # algorithm (c), fast
   [(1, 2), (1, 3), (3, 2), (1, 4)]
```

   Filename: nbest.py

2. k-way mergesort (the classical mergesort is a special case where k=2).

```
   >>> kmergesort([4,1,5,2,6,3,7,0], 3)
   [0,1,2,3,4,5,6,7]
```

   Q: What is the complexity? Write down the detailed analysis in report.txt.

   Filename: kmergesort.py

3. [WILL BE GRADED]

   Find the k smallest numbers in a data stream of length n (k<<n),
   using only O(k) space (the stream itself might be too big to fit in memory).

```
   >>> ksmallest(4, [10, 2, 9, 3, 7, 8, 11, 5, 7])
   [2, 3, 5, 7]
```

```
    >>> ksmallest(3, range(1000000, 0, -1))
    [1, 2, 3]
```

Note:
a) it should work with both lists and lazy lists
b) the output list should be sorted

Q: What is your complexity? Write down the detailed analysis in report.txt.

Filename: datastream.py

4. (optional) Analyze the time complexities of the two "slow" solutions in HW3 we provided for the closest_sorted problem.

5. (optional) Summarize the time complexities of the basic operations (push, pop-min, peak, heapify) for these implementations of priority queue:

unsorted array,
sorted array (highest priority first),
sorted array (lowest priority first),
linked list,
binary heap

Debriefing (required!): --------------------------

0. What's your name?
1. Approximately how many hours did you spend on this assignment?
2. Would you rate it as easy, moderate, or difficult?
3. Did you work on it mostly alone, or mostly with other people?
Note you are encouraged to discuss with your classmates,
but each students should submit his/her own code.
4. How deeply do you feel you understand the material it covers (0%-100%)?
5. Which part(s) of the course you like the most so far?
6. Which part(s) of the course you dislike the most so far?

This section is intended to help us calibrate the homework assignments.
Your answers to this section will *not* affect your grade; however, skipping it will certainly do.