CS 2302 - Data Structures
Fall 2018
Project 5 - Option B

Overview

Complete the implementation of the Min-Heap data structure (you'll find a partial implementation in the Appendix that you need to complete). After doing this, use your implementation of the data structure to implement heapsort. Test your implementation by doing at least one of the following options:

- Unit Tests
- Reading a file with a list of numbers separated by commas and printing the sorted sequence
- Creating a separate file where you call your heapsort implementation using hard-coded lists of numbers

Appendix

```
class Heap:

  def __init__(self):
     self.heap_array = []

  def insert(self, k):

     self.heap_array.append(k)

     # TODO: Complete implementation

  def extract_min(self):

     if self.is_empty():
        return None

     min_elem = self.heap_array.append[0]

     # TODO: Complete implementation

     return min_elem

  def is_empty(self):
     return len(self.heap_array) == 0
```

What you need to do

Extra Credit (10 points total):

Solve the following LeetCode Problems

1. https://leetcode.com/problems/top-k-frequent-elements/description/
2. https://leetcode.com/problems/design-twitter/description/
3. https://leetcode.com/problems/find-k-pairs-with-smallest-sums/description/

Part 1 - Due Tuesday, November 27, 2018

Implement the program described above and upload your code to GitHub.

Part 2 - Due Thursday, November 29, 2018

Add your team members as collaborators to your GitHub repo. They will add you to their projects as a collaborator as well. Read their code and give them feedback. Use *pull requests* and/or the *Issues* section to do so .

Rubric

Criteria	**Proficient**	**Neutral**	**Unsatisfactory**
Correctness	The code compiles, runs, and solves the problem.	The code compiles, runs, but does not solve the problem (partial implementation).	The code does not compile/run, or little progress was made.
Space and Time complexity	Appropriate for the problem.	Can be greatly improved.	Space and time complexity not analyzed
Problem Decomposition	Operations are broken down into loosely coupled, highly cohesive methods	Operations are broken down into methods, but they are not loosely coupled/highly cohesive	Most of the logic is inside a couple of big methods
Style	Variables and methods have meaningful/appropriate names	Only a subset of the variables and methods have meaningful/appropriate names	Few or none of the variables and methods have meaningful/appropriate names

Robustness	Program handles erroneous or unexpected input gracefully	Program handles some erroneous or unexpected input gracefully	Program does not handle erroneous or unexpected input gracefully
Documentation	Non-obvious code segments are well documented	Some non-obvious code segments are documented	Few or none non-obvious segments are documented
Code Review	Useful feedback was provided to team members. Feedback received from team members was used to improve the code.	Feedback was provided to team members, but it was not very useful. Feedback received from team mates was partially used to improve the code	Little to no feedback was provided to team mates. Received feedback was not used to improve the code.
Report	Covers all required material in a concise and clear way with proper grammar and spelling.	Covers a subset of the required material in a concise and clear way with proper grammar and spelling.	Does not cover enough material and/or the material is not presented in a concise and clear way with proper grammar and spelling.