|  |  |
| --- | --- |
| **CSE 331** | **Fall 2017** |

Project 4

100 points

Due September 28th 8:00pm

This is not a team work, do not copy somebody else’s work.

**Assignment Overview**

A common problem in computer science is quickly computing statistical information.

The median of N numbers is defined as the middle value after sorting them, if N is odd, else, it would be the average of the middle two numbers. You will receive an instruction on each line “a X” for add integer X to collection or “r X” for remove integer X from collection. After each instruction, you should return the median.

A simple, naïve and slow solution is to create a single array and maintain sorted order by removing and adding to that sorted value that will implicitly shuffle the element before or after around and grab the median from that. **Do not do that.** A less intuitive but faster solution is to utilize the loose hierarchical relationship of heaps.

**Assignment Deliverables**

RunningMedian.py

Be sure to use the specified file name(s) and to submit your files for grading **via D2L Dropbox** before the project deadline.

**Assignment Specifications**

Your task will be to complete the methods listed below.

* find\_median(self, command)

**Assignment Notes**

Points will be deducted if your solution has any warnings of type:

* Any modification of the method signatures of **\_\_init\_\_(self)** and **find\_median\_\_(self, command)**
* Here you should not you will **NOT** be given any invalid commands i.e. anything that’s not ‘a’ or ‘r’ and every remove ‘X’, you can guarantee that the element is already in the collection.
* The find\_median method already has some boiler plate code that handles the parsing for you, feel free to modify but you will not need to, to get the valid input.
* I have provided a helper function, **heapq\_remove**. It takes in a heap to modify, the index of the element in which you want to remove and correctly removes and validates the heap after removal. It’s completely optional to use.
* You are provided with the skeleton code for the stack class definition. Your job is to complete the methods given. You may add (and should) more methods, than what is provided, **but may not alter the original method signatures in any way.**
* **Mandatory:** Make sure to run your code one more time before submission as a ‘mercy check’, in case of any accidental syntax errors.
* Please refer to https://d2l.msu.edu/d2l/le/content/482170/viewContent/5369895/View for a detailed tutorial on how to run and debug a python project within PyCharm.

Description the Problem

The naïve solution would result in an overall

* O(N) for add instruction and return median, why?
* O(log(N)) to find the place to insert the element
* O(N) to shuffle the other elements to fit the new element
* O(1) to compute the median
* O(log(N)) + O(N) + O(1) = O(N) algorithm
* You should infer the remove an element would be the same.

The optimal solution should target

* O(log(N)) for an add instruction and return median, why?
* O(log(N)) to add an element to a given heap
* O(log(N)) for the case where adding an element to a heap could cause imbalanced heaps
  + O(1) pop top from an heap
  + O(log(N)) to add that popped element into anther heap
* O(1) to compute the median
* O(N) to remove an element and return the median, why?
* O(N) to find the element to replace from a heap
* O(log(N)) to remove that given element from heap
* O(log(N)) for the case where removing an element to a heap could cause imbalanced heaps
  + O(1) pop top from an heap
  + O(log(N)) to add that popped element into anther heap
* O(1) to compute the median
* O(N) + O(log(N)) + O(log(N) + O(1) = O(N)

**Reference:** Author of this project is Cyndy Ishida

