Find Median from Data Stream

Group Members:
6135102 Suchawan Chaiworn
6118516 Siyu Han

Hello!

We are Luis and Siyu

We're here to present our algorithm design project

Table of Contents

The Problem

295. Find Median from Data Stream (From LeetCode)

Median is the middle value in an ordered integer list. If the size of the list is even, there is no middle value. So the median is the mean of the two middle value.

For example,

`[2,3,4]`, the median is `3`

`[2,3]`, the median is `(2 + 3) / 2 = 2.5`

Design a data structure that supports the following two operations:

- void addNum(int num) - Add an integer number from the data stream to the data structure.
- double findMedian() - Return the median of all elements so far.

The Solution

Fields & Functions

Fields:

- **self.A:** a list data structure

Functions:

- **addNum():** appends an integer to the list
- **findMedian():** order and find the median of the list

addNum()

Parameters

- **self:** current instance of the class
- **num:** the input

What does it do?

- Checks if num is an integer
- If yes, append num to list
- If no, prints out a message to remind the user about wrong input

findMedian()

Parameters

- **self:** current instance of the class

What does it do?

- Sort the list in ascending order using sorted()
    - The built-in sorting algorithm: Timsort
- Check for the cases: is the list length even or odd?
- Find the index of the middle element(s) and return the median using the index

Timsort

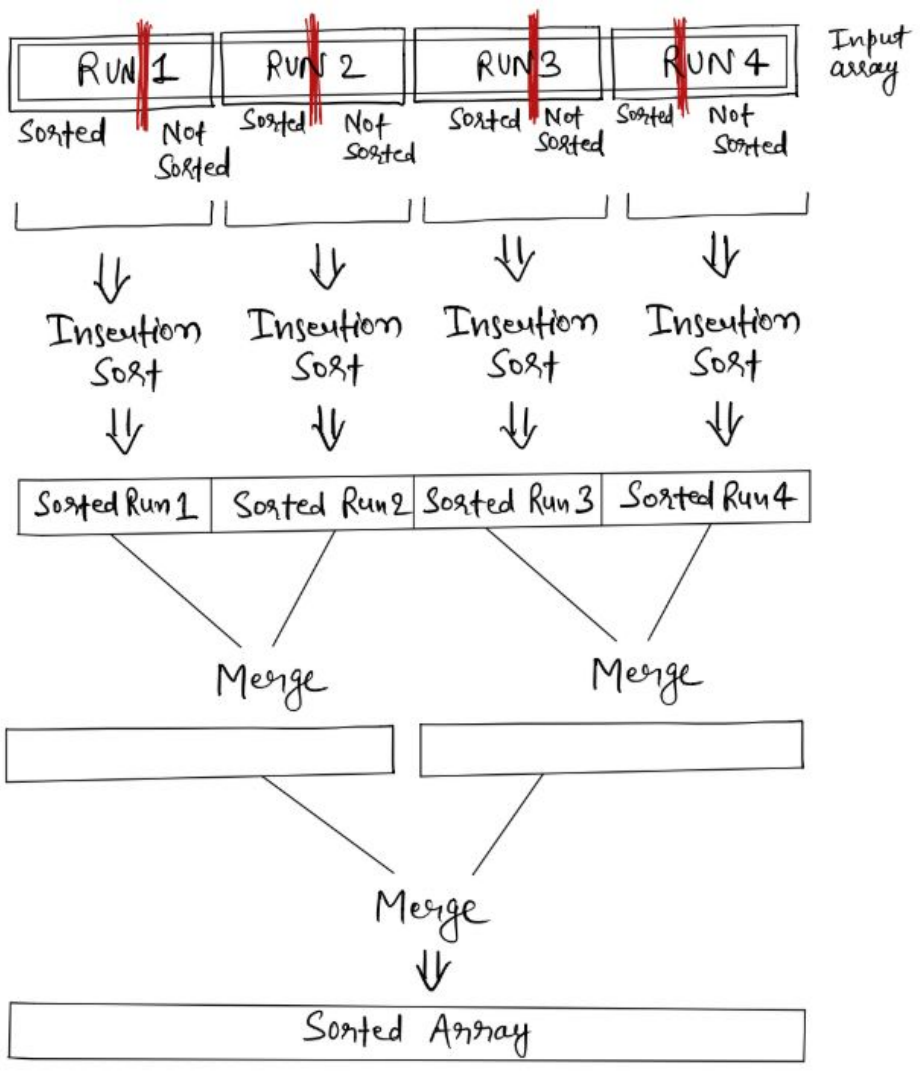
- Timsort is a hybrid stable sorting algorithm, derived from insertion sort and merge sort
  - Insertion sort: small size of data; input array is partially sorted
  - Merge sort: the size of subarrays are close to power of 2
- In Timsort, if the input array has more than 64 elements, it will divide the array into blocks known as **Run** and look for blocks that are already sorted.

[3,2,1,7,9,8] -> [1,2,3,7,9,8]

Not all part of input array is sorted in real world

Minrun: the minimum size of a Run (32 to 64)

- Small enough for Insertion sort to run
- As close as possible to power of 2 for Merge sort to run faster



Time Complexity

Time Complexity

	Best Case	Average Case	Worst Case
Insertion	O(1)	O(1)	O(1)
Timsort	O(n)	O(nlogn)	O(nlogn)
Get	O(1)	O(1)	O(1)

Implementation

```python
class MedianFinder:

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

    def addNum(self, num):
        if type(num) == int:
            self.A.append(num)
        else:
            print("The input should be an integer!")

    def findMedian(self):
        self.A = sorted(self.A)
        if len(self.A) % 2 == 1:
            median = self.A[int(len(self.A)/2)]
            return median
        else:
            median = (self.A[int(len(self.A)/2)] + self.A[int(len(self.A)/2) - 1]) / 2
            return median
```

Test Cases

Input

```
from median_finder import MedianFinder

mf = MedianFinder()
mf.addNum(1)
mf.addNum(3)
print(mf.findMedian())
mf.addNum(4)
mf.addNum(2)
print(mf.findMedian())
mf.addNum(5)
print(mf.findMedian())
mf.addNum(6)
mf.addNum(7)
print(mf.findMedian())
```

Output

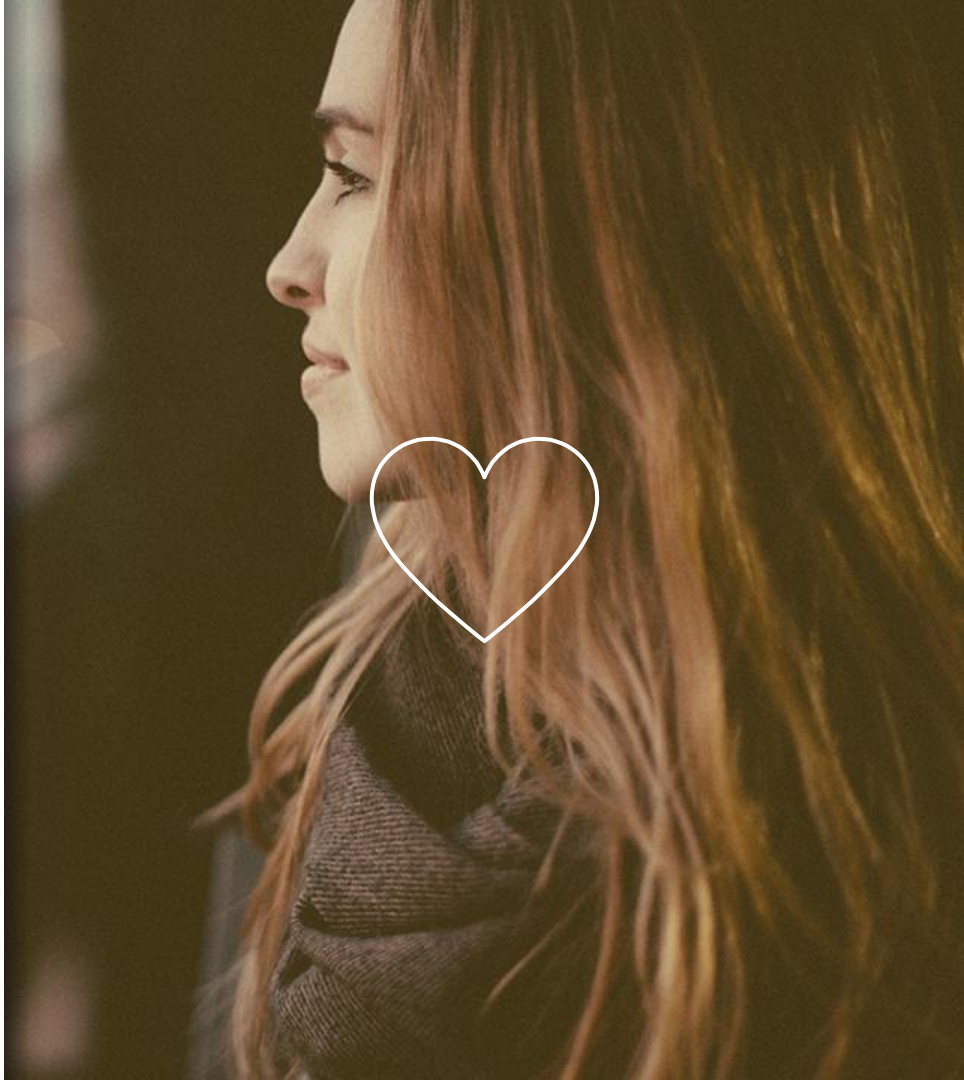
```
2.0
2.5
3
4
```

Submission

Time Submitted	Status	Runtime	Memory	Language
02/27/2021 13:58	Accepted	2264 ms	25.5 MB	python3

Thanks!

Any questions?



References

https://www.slidescarnival.com/

https://en.wikipedia.org/wiki/Timsort

https://leetcode.com/problems/find-median-from-data-stream

https://njha-collab.github.io/blogs/understanding-tim-sort