Lab06: Binary Search
CSE2050 Fall 2017
Instructor: Jeffrey Meunier & Wei Wei
TA: Jenny Blessing, Param Bidja, Yamuna Rajan & Zigeng Wang

1. Introduction

In this exercise, we will complete one Python function in order to find a potential **indexMatch i** for a sorted list with distinct integers in which **list[i] = i** by using binary search.

2. Objectives

The purpose of this assignment is to help you:

- Practice binary search.
- Refresh knowledge on python lists and functions.

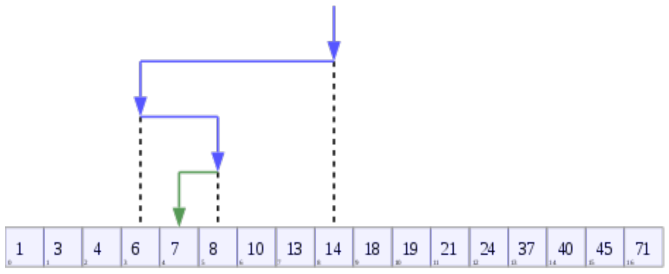
Note*: Before you start, if you are not familiar with binary search, list or function in Python, you are recommended to review your lecture notes and the additional materials from Dr. Don Sheehy.*

3. Background

3.1. Binary Search

Binary search is a classic search algorithm which finds the position of a target value within a sorted list.[1] In binary search, when we are looking for a particular item in a sorted list, we compare the targeted value with the **median** of the current list and break the list in half iteratively by keeping the half which may contain the targeted item. So, given a list of length n, the asymptotic running time of the binary search is $O(log n)$.

The following figure shows the implementation of a binary search for target value 7 on a list with length 17. With binary search, it takes only 4 steps to find target value, which is much faster than linear search asymptotically.



Fig[2]. 1 Visualization of Binary Search with Target Value Equal to 7

[1] Cited from Wikipedia, Binary search algorithm. https://en.wikipedia.org/wiki/Binary_search_algorithm
[2] Cited from Wikipedia. https://commons.wikimedia.org/wiki/File:Binary_Search_Depiction.svg#

4. Assignment

In this assignment, we only need to complete **indexMatch** function in binary.py. The input of this function is a sorted list **list1** with distinct integers. We need to utilize binary search to find out and return a potential index **i** for which **list1[i] = i**.

4.1. Index Match using Binary Search

In order to have a nice understanding of the binary search implementation in index match, we give an example as the following.

For a certain input,

list1 = [-3, -1, 0, 2, 3, 4, 5, 6, 8, 12, 20]

we can visualize the item values and their corresponding indices as,

value	-3	-1	0	2	3	4	5	6	**8**	12	20
index	0	1	2	3	4	5	6	7	**8**	9	10

where we can see **8** is the matched index that the **indexMatch** function should return.

Here, you may wonder that it is pretty intuitive to use a linear search that traverse **list1** to locate the **match** by comparing each value in **list1** with its corresponding index, and we can definitely get the correct output in $O(n)$ time.

However, using a linear search for index match has not fully utilized the charming feature of **list1** that **list1** is a ***sorted*** list. We can, by elegantly implementing binary search, reduce the computational complexity of **indexMatch** to $O(logn)$ time. Any ideas?

OK. Let's go back to the example of **list1** = [-3, -1, 0, 2, 3, 4, 5, 6, 8, 12, 20] and we add another row in the table which calculates the difference between value and its index as the following.

value	-3	-1	0	2	3	**4**	5	6	**8**	**12**	20
index	0	1	2	3	4	**5**	6	7	**8**	**9**	10
value - index	-3	-2	-2	-1	-1	**-1**	-1	-1	**0**	**3**	10

After looking at the difference between value and its index, we can easily discover that the list in the 3rd row is also an un-descending list where the position whose corresponding value is **0** exists a **match**.

Note: *Adding the 3rd row in the table above never means asking you to create a new list in your code that contains the subtraction results, since only doing the linear subtractions takes linear time, where we want to design some algorithm runs in* $O(logn)$. *Please see the following paragraph to understand what we really need to do in this assignment.*

And we can further discover that if a certain index is larger than its corresponding value (e.g. index 5 > value 4), all the indices which are to the left of the current index cannot contain the targeted **match**. Similarly, if a certain index is smaller than its corresponding value (e.g. index 9 > value 12), all the indices that are to the right of the current index cannot contain the targeted **match** either. So, with this discovery, we can utilize the binary search algorithm to shrink the search window into half in each iteration, and identify the potential **match** in $O(logn)$ time.

4.2. Index Match Implementation

Now, we can open the skeleton code. In the code, the following is the **indexMatch** function is we need to complete. Input **list1** is a sorted list with distinct integers, and the function will return index **i** for which **list1[i] = i**. If there is no index that satisfies **list1[i] = i**, the function should return "No Match Found!".

```
def indexMatch(list1):
  global listMedian
  ### add your code here
  return ("No Match Found!")   # the function should return "No Match Found!" if there is no list1[i] = i
```

In order to check the functionality of your code, we added a global variable **listMedian** into the function to save all the medians that having been used in the iterations of the binary search. So, in each iteration, we should append the **listMedian** with the new median value (the index).

4.3 Basic Structure

The following section is an addendum to the original document to provide additional clarity.

Note that in this implementation, the binary search algorithm (and the process of adding elements to **listMedian**) will only run while the right and left indices are greater than one element apart. In other words, the current range of the list from the right index to the left index is greater than one element.

If the program has modified the left and right indices such that there is only a single element remaining in the range, then this element, even if it is the match you are looking for, should **not** be included in **listMedian**. In Mimir, the test *testMatch2* will test for this case.

To do this, the structure of the code could look something like this:

```
def indexMatch(list1):
  global listMedian
  # initialize the index variables 'left' and 'right'
  while (right – left) > 1:
    # implement binary search
    # With each iteration, add the current index of the median into listMedian
  # conditional check to see if the single remaining element is a match
  return ("No Match Found!")
```

4.4. Multiple Matches

In this assignment, you may have another concern, which is, for a list, there can be more than one **matches** in it, like the following with **5 matches**.

value	-3	-1	0	2	**4**	**5**	**6**	**7**	**8**	12	20
index	0	1	2	3	**4**	**5**	**6**	**7**	**8**	9	10

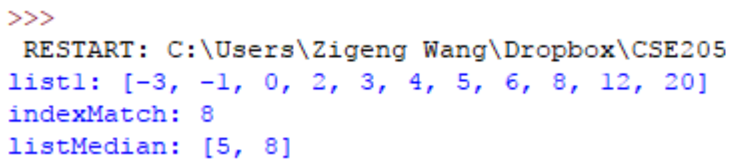
Here, we do not need to worry about the multiple **matches** cases, returning the first match we discovered should be fine.

Note: "The first match we discovered" does not mean the match with the lowest index, but the first match we discovered by using the binary search implementation.

4.5. Some Sample Outputs

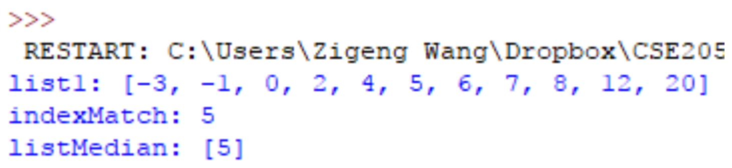
The following are 3 sample outputs for your reference.

Single Match:



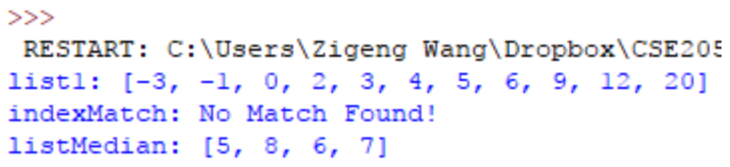
```
>>>
 RESTART: C:\Users\Zigeng Wang\Dropbox\CSE205
list1: [-3, -1, 0, 2, 3, 4, 5, 6, 8, 12, 20]
indexMatch: 8
listMedian: [5, 8]
```

Multiple Matches:



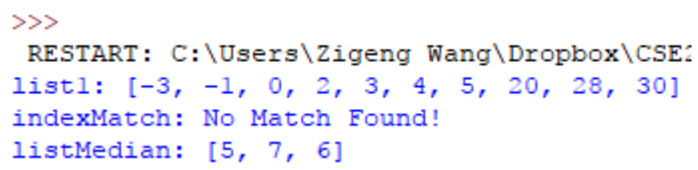
```
>>>
 RESTART: C:\Users\Zigeng Wang\Dropbox\CSE205
list1: [-3, -1, 0, 2, 4, 5, 6, 7, 8, 12, 20]
indexMatch: 5
listMedian: [5]
```

No Matches_1:



```
>>>
 RESTART: C:\Users\Zigeng Wang\Dropbox\CSE205
list1: [-3, -1, 0, 2, 3, 4, 5, 6, 9, 12, 20]
indexMatch: No Match Found!
listMedian: [5, 8, 6, 7]
```

No Matches_2:



```
>>>
 RESTART: C:\Users\Zigeng Wang\Dropbox\CSE
list1: [-3, -1, 0, 2, 3, 4, 5, 20, 28, 30]
indexMatch: No Match Found!
listMedian: [5, 7, 6]
```

5. Submit your work to Mimir

Submit your code to Mimir after you complete your code. Please delete or comment out all the testing codes before your submission (We should only keep the **indexMatch** function and the global assignment **listMedian = []**). The Mimir will automatically your submission based on different unit test cases. You can submit your code to Mimir up to **30 times** to refresh your existing score before the submission deadline.

6. Due date

This lab assignment is worth **2 points** in the final grade. It will be due by **11:59pm on Wednesday, Oct 25th 2017**. A penalty of **10% per day** will be deducted from your grade, starting at 12:00am.

7. Getting help

Start your project early, because you will probably not be able to get timely help in the last few hours before the assignment is due.

- Go to the office hours of instructors and TAs.
    - Prof. Wei Wei: Mon. 2:30 - 3:15pm, Wed. 2:30 - 3:15pm, Thurs. 2:30 - 3:15pm, Fri. 2:30 - 3:15pm @ITE258
    - Jenny Blessing: Fri. 12pm - 2pm @ITE140
    - Param Bidja: Tues. 2pm - 3pm @ITE140
    - Yamuna Rajan: Tues. 11am - 12pm, Wed. 9:30am - 10:30am @ITE140
    - Zigeng Wang: Mon. 3pm - 4pm @ITE140
- Post your questions on Piazza. Instructors, TAs and many of your classmates may answer your questions.
- Search the answer of your unknown questions on the Internet. Many questions asked by you might have been asked and answered many times online already.