Due: Wednesday Nov. 15th, 2017 at 10 am

Assignment Guidelines

- This assignment covers material in Modules 07 and 08.
- Submission details:
  - Solutions to these questions must be placed in files `a07q1.py`, `a07q2.py`, `a07q3.py`, and `a07q4.py`.
  - You must be using Python 3 or higher.
  - Download the interface file from the course Web page to ensure that all function names are spelled correctly and each function has the correct number and order of parameters.
  - All solutions must be submitted to MarkUs. No solutions will be accepted through email, even if you are having issues with MarkUs.
  - Verify using MarkUs and your basic test results that your files were properly submitted and are readable on MarkUs.
  - For full style marks, your program must follow the Python section of the CS116 Style Guide.
  - Be sure to review the Academic Integrity policy on the Assignments page
- Download the testing module from the course web page. Include `import check` in each solution file.
- Restrictions:
  - Do not import any modules other than `math` and `check`.
  - Do not use any other Python functions not discussed in class or explicitly allowed elsewhere. See the allowable functions post on Piazza.
  - While you may use global *constants* in your solutions, do **not** use global *variables* for anything other than testing.
  - Read each question carefully for additional restrictions or tips.

The solutions you submit must be entirely your own work. Do not look up either full or partial solutions on the Internet or in printed sources.

1. **Run Time**

   The interface file contains several functions. You need to determine the worse-case running time in big-O notation for each function, and indicate your answers in your `a07q1.py` file, as explained in the interface. Note that the basic test result for the question will only indicate that a file was received – it will not indicate any correctness results.

2.

Searching

Ternary search is an algorithm used to search for an item within a sorted list. It is similar to binary search, except that the search region is divided into three smaller regions (having lengths as equal as possible) at each iteration by picking two indexes ind1 and ind2 (ind1 < ind2):

• Region 1 contains all items having index value less than ind1
• Region 2 contains all items having index value greater than ind1 but less than ind2
• Region 3 contains all items having index value greater than ind2

If possible, the sizes of these regions should be equal. If this is not possible, then the size of Region 1 must be greater than or equal to the size of Region 2, and the size of Region 2 must be greater than or equal to the size of Region 3. The sizes of any two regions may differ by at most one.

For example, consider the list L of length 19 in Figure 1. We pick ind1 to be 6 and ind2 to be 13, which gives Region 1 a size of 6 (indexes 0 through 5), Region 2 a size of 6 (indexes 7 through 12), and Region 3 a size of 5 (indexes 14 through 18).

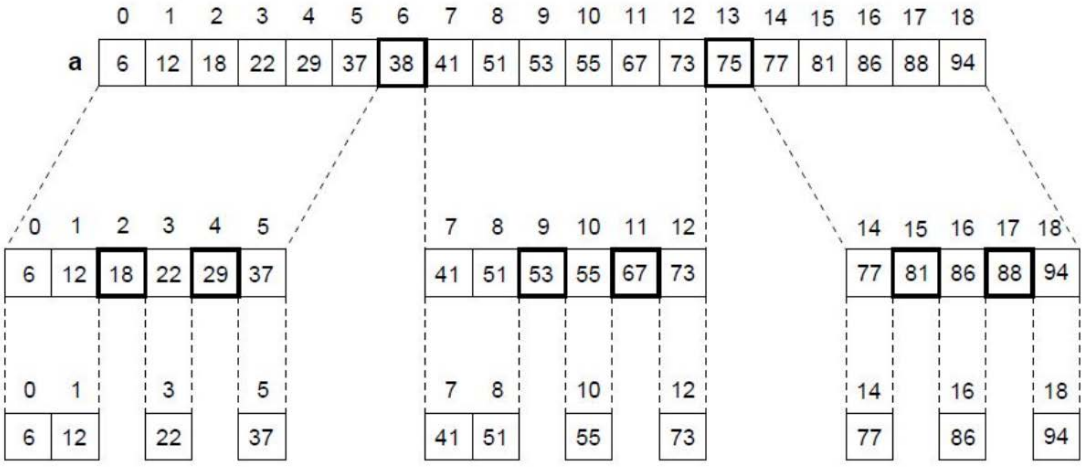

Figure 1: Selecting index values in the trinary search algorithm.

A single iteration of the ternary search algorithm is described below using pseudocode. L is a sorted list of integers and v is the value for which we are searching.

if the size of the search region is <= 4
perform a linear search for v
else
select indexes ind1 and ind2
if L[ind1] is equal to v
stop, we have found v
else if v < L[ind1]
repeat with Region 1 being the new search region
else if L[ind2] is equal to v
stop, we have found v
else if v < L[ind2]
repeat with Region 2 being the new search region
else
repeat with Region 3 being the new search region

As an example, consider calling the ternary search algorithm with the list L in Figure 1 and 51 as the value v.
The initial search region is between indexes 0 and 18 (inclusive), which has length greater than 4.
The values ind1 = 6 and ind2 = 13 are selected.
L[ind1] == v is false.
v < L[ind1] is false.
L[ind2] == v is false.
v < L[ind2] is true, so the new search region is between indexes 7 and 12 (inclusive), which has length greater than 4.
The values ind1 = 9 and ind2 = 11 are selected.
L[ind1] == v is false.
v < L[ind1] is true, so the new search region is between indexes 7 and 8 (inclusive), which has length less than or equal to 4.
L[7] == v is false.
L[8] == v is true, so we stop; v has been found in the list L.

You will write a Python function `ternary_search` that consumes `nlst`, a non-empty list of integers, and `val`, an integer. This function does not return a value, but prints out information about the steps performed at runtime. In the following first two examples, the list `L` refers to the list in Figure 1.

Example 1: Calling `ternary_search(L, 51)` should print:

```
Checking if 51 is equal to 38
Checking if 51 is less than 38
Checking if 51 is equal to 75
Checking if 51 is less than 75
Checking if 51 is equal to 53
Checking if 51 is less than 53
Checking if 51 is equal to 41
Checking if 51 is equal to 51
Search successful
51 is located at index 8
A total of 8 comparisons were made
```

Example 2: Calling `ternary_search(L, 75)` should print:

```
Checking if 75 is equal to 38
Checking if 75 is less than 38
Checking if 75 is equal to 75
Search successful
75 is located at index 13
A total of 3 comparisons were made
```

Example 3: For `L=[6,12,18,22]. ternary_search(L, 18)` should print:

```
Checking if 18 is equal to 6
Checking if 18 is equal to 12
Checking if 18 is equal to 18
Search successful
18 is located at index 2
A total of 3 comparisons were made
```

Example 4.

`ternary_search([6,12,18,22,29,37,38,41,51,53,55,67,73,75,77,81,86,88,94], 27)` should print:

```
Checking if 27 is equal to 38
Checking if 27 is less than 38
Checking if 27 is equal to 18
Checking if 27 is less than 18
Checking if 27 is equal to 29
Checking if 27 is less than 29
Checking if 27 is equal to 22
Search not successful
A total of 7 comparisons were made
```

3. Merging

Note: You are NOT allowed to define helper function for this question, nor to sort.

Hint: Use merge for 2 sorted lists provided in course notes and change it to work for 3 sorted lists.

Write a Python function `merge3` that consumes three sorted lists in ascending order, `L1`, `L2`, and `L3`, and merges them into one new sorted list and returns it (the new sorted list).

For example:

```
merge3([-7,1,3,4,100,200],[-3,10],[110,120,400,500,600]) =>
            [-7,-3,1,3,4,10,100,110,120,200,400,500,600])
```

4. Go Efficiency Go…

Write a Python function `max_times` that consumes `L,` a non-empty list of non-empty lists of integers, and returns a list of length two, where the first element is the maximal value in `L`, and the second element is the number of times the maximal value appears in `L`.

Requirements: Your solution for this question must meet ALL of the following requirements:

- You are not allowed to use recursion
- You are not allowed to use helper functions
- You are not allowed to use built-in function max nor any list methods
- Your solution must go through the nested list only once. (You are not allowed to make a new list other than the list to be returned.

Examples:

```
max_times([[1,2,2,2,3],[2,2,1],[3,2,3,3]]) => [3,4]
max_times([[-1]]) => [-1,1]
```