CS1210 Computer Science I: Fundamentals

Lab 5: Best k of N Elements

Introduction

Recall the selection sort algorithm discussed in class:

```
def selsort(L):
    for i in range(len(L)-1):
        m = L[i:].index(min(L[i:]))
        L[i],L[i+m] = L[i+m],L[i]
    return(L)
```

In this assignment, you will be writing a new function *bestKofN*(L, k, *fn*) that, much like selection sort, takes a (mutable) list of elements, L, and returns the "best" k elements of that list according to a specified criteria. In fact, we can use *selsort*(L) to define a very simplistic sort of "best k of n" function as follows:

```
def bestKofN(L):
    return(selSort(L)[:k])
```

There are two problems with this version *bestKofN*():

it's not particularly efficient; and
"best" is always defined as "smallest."

To address the first point, realize that you needn't sort the entire list to find the k smallest elements; just running through the first k passes of selection sort will do! Given that we know, from our discussion in class, that selection sort is $O(N^2)$, then alternative algorithm is only $O(N * k)$, and if $k \ll N$, this tends towards $O(N)$.

As for the second point, what if L were instead a list of words, and "best" were defined by the length of the words (shorter is better) and, all else being equal, by alphabetical order? So, for example:

```
>>> L
['this', 'is', 'a', 'test', 'of', 'sorts']
>>> bestKofN(L, 3, shorter)
['a', 'is', 'of']
>>> L
['a', 'is', 'of', 'test', 'this', 'sorts']
```

Remember, your solution may not earn all possible points if you do not comment it, or properly complete the *header* of the file (the section of comments at the beginning of the file which should contain your name -- not mine -- and your section identifier).

1. lt(a, b)

Write a function *lt*(a, b) that returns True if $a < b$.

2. gt(a, b)

Write a function *gt*(a, b) that returns True if $a > b$.

3. shorter(a, b)

Write a function *shorter*(a, b) that returns True if *len*(a) < *len*(b) or if a and b are equal length and $a < b$.

4. longer(a, b) Write a function *longer*(a, b) that returns True if *len*(a) > *len*(b) or if a and b are equal length and $a < b$.

5. bestKofN(L, k, fn)

bestKofN$(L, k,$ *fn*$)$ returns a list of the "best" k elements selected from the input list L based on the definition of "best" implied by *fn*. So:

```
>>> L="four score and seven years ago our fathers brought forth on   \\
    this continent a new nation".split()
>>> bestKofN(L,4,lt)
['a', 'ago', 'and', 'brought']
>>> L
['a', 'ago', 'and', 'brought', 'years', 'score', 'our', 'fathers', 'seven',   \\
 'forth', 'on', 'this', 'continent', 'four', 'new', 'nation']
>>> bestKofN(L,4,gt)
['years', 'this', 'seven', 'score']
>>> bestKofN(L,5,shorter)
['a', 'on', 'ago', 'and', 'new']
>>> bestKofN(L,7,longer)
['continent', 'brought', 'fathers', 'nation', 'forth', 'score', 'seven']
>>>
```

Notice how *bestKofN*$(L, 4, lt)$ returns the first four elements of L in alphabetical order but does not disrupt the ordering of the remainder of L (save for moving the short words to the front). Also note how *bestKofN*$(L, 5,$ *shorter*$)$ conserves alphabetical order on equal-length words ('ago', 'and', and 'new'), and how *bestKofN*$(L, 7,$ *longer*$)$ returns the longest words but conserves (regular) alphabetical order within length.

Here's a hint on how to get started. The template file contains a definition of selection sort that relies on the Python *min*() function. Begin by rewriting this function so as to use a "helper" function of your own design instead. Next, modify the "helper" function to accept a comparison function, like *lt*(), *gt*(), *shorter*() or *longer*() instead of relying on a built-in comparison operator like <.

Methods

For paired assignments, you should always login as one of the two partners and work in that partner's directory (in other words, don't try to switch from one machine to the other). In every lab, you will turn in just **one** solution, with both partner's names in it. **Important: you must upload your solution prior to the end of discussion section; no late labs are accepted.**

(1) Download the Python file template provided and save it to your local machine (on the Desktop is fine).

(2) Open the file using IDLE3.

(3) The file contains a special function *hawkid*() along with function *signatures* for each of the functions in the assignment.

(4) Complete the *hawkid*() function according to the instructions given in the template file. **Important: a correct version of this function is required by the autograder to properly**

credit you for your work.

(5) Complete each function by replacing the placeholder body of that function with the appropriate expression (you should alternate which partner is "driving" and which partner is "navigating" on each function or at least for every two functions).

(6) You should test each of your functions in the Python REPL.

(7) You should make sure your Python file loads and runs properly; files with syntax or indentation errors will receive no credit.

(8) Upload your Python file to ICON. Note that only the last file updated will be graded. **Important: you need upload only one file to one partner's ICON account; just be sure you include all the partner's names in the** *hawkid*() **function.**

Because we give partial credit, you should consider uploading your Python file to ICON several times (each time making sure the file loads and runs without error) over the course of the lab period to make sure all completed work gets graded.

Finally, don't forget to fill out the lab 5 survey, which is open and available to you until midnight next Monday (October 15). Failure to fill out the survey may result in less-than-full-credit on the assignment.