Project 7: LIS

Due: Friday April 26, 11:59 pm

1 Assignment Overview

For this project you will implement a dynamic program that solves the longest increasing subsequence problem. A sequence S is a subsequence of another sequence T if you can produce S by removing some items from T (without reordering them). For instance, [0, 2, 5] is a subsequence of [0, 1, 2, 3, 4, 5,], but [5, 2, 0] is not. Your objective is to find the longest possible subsequence whose items are *strictly increasing* (that is if $i < j$ then $s_i < s_j$).

2 Assignment Deliverables

You must turn in completed versions of the following files:

- Lis.py

Be sure to use the specified file name and to submit your files for grading via **Mimir** before the project deadline.

3 Assignment Specifications

Your task will be to complete the methods listed below:

- `verify_subseq`: Determines whether one sequence is a subsequence of another.
- `verify_increasing`: Determines whether a sequence is in increasing order.
- `find_lis`: Finds the longest increasing subsequence of the given sequence.

If multiple subsequences each have the maximum possible size you may return any of them.

You may use any of the standard collections discussed in class as well as the built-in functions that act upon them. You may add any additional helper functions that you need.

You may assume that each sequence contains mutually comparable items (they have a $<$ operator), but you cannot make other assumptions about their types. You are not allowed to modify the input sequence in any way. The sequence is indexable (but it might not be a list).

Both of the verification methods should run in $O(n)$ time and require only a constant amount of extra space. While a $O(n^2)$ solution to the `find_lis` method is readily apparent, your solution should take $O(n \log n)$ time and require at most $O(n)$ extra space.

You should include comments where appropriate. In particular, you should describe the method used for computing the LIS. You must also add documentation for each of the above methods (as well as for any helper functions created).

4 Assignment Notes

- Points will be deducted if your solution has warnings of any type.
- You have been supplied with stubs for the required methods. You must complete these methods to complete the project. You may add more functions than what was provided, but **you may not modify the signatures of the provided methods**.
- You have been provided with some unit tests that can be used to assess your solution. There are additional test cases that will be kept hidden.
- It is your responsibility to check for potential edge cases. The supplied unit tests do not account for all possible valid inputs
- You may use a wide variety of collections classes. You may want to use `list`, `set`, `dict`, `heapq`, `deque`, or `defaultdict`.
- The items in the subsequence must be *strictly* increasing. Equal items are not allowed.
- You may not modify the input sequence.
- Some of the tests give you a string (character sequence) instead of a list. You may still return a list rather than a string.