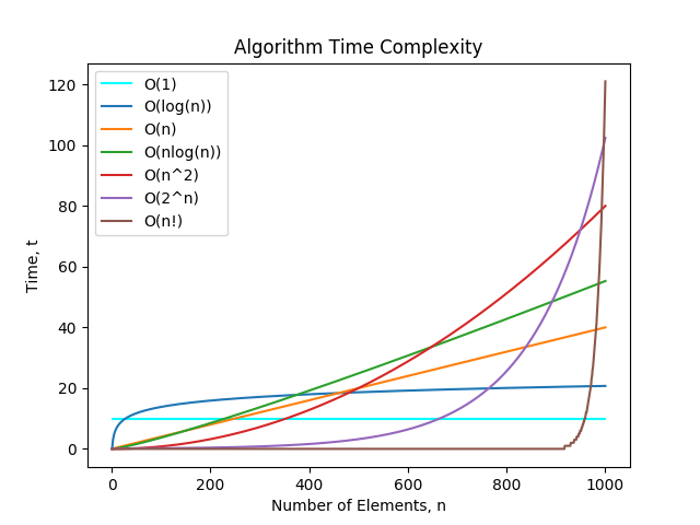
Project 3: Hybrid Sorting

**Due: Thursday, October 10th 8:00 pm**

*This is not a team project, do not copy someone else’s work.*

Description

For this project, you will be creating a hybrid sort algorithm using Quick Sort and Insertion Sort. Due to the overhead of recursively splitting containers, Insertion Sort may be preferred at small list sizes. Quick Sort has expected nlogn run time complexity while insertion has n2. In the figure above, there is a certain "threshold" that yields a better complexity while using insertion.

You will be sorting a doubly linked list using Quick Sort until the (partitioned) linked list is **less than or equal to** a given threshold, at which point you will switch to Insertion Sort.

(If you want to visualize this, input n2and nlogn in a graphing calculator. Here is one online: <https://www.desmos.com/calculator>)

(In addition, click on this link <https://visualgo.net/bn/sorting>and click on quick sort if you would like to see a visual of this algorithm)

Turning It In

Your completed project must be submitted as a folder named "**Project3**" and must include:

* QuickSort.py, a python3 file.
* InsertionSort.py, a python3 file.
* DoublyLinkedList.py, a python3 file.
* README.txt, a text file that includes:
  + Your name and feedback on the project
  + How long it took to complete
  + A list of any external resources that you used, especially websites (make sure to include the URLs) and which function(s) you used this information for.
* \_\_init\_\_.py, a python3 file
  + This should be blank and left in the submission folder.

Assignment Specifications

You are given three files, **QuickSort.py, InsertionSort.py, & DoublyLinkedList.py**. You must complete and implement the following functions. Take note of the specified return values and input parameters. **Do not change the function signatures.**

**QuickSort.py:**

* quick\_sort(dll, start, end, size, threshold)
  + This function will sort a doubly linked list from the given start and end nodes with the quick sort algorithm while calling insertion sort when the threshold has been underpassed
  + dll: the doubly linked list that is being sorted
  + start: node to start sorting at (inclusive)
  + end: node to end sorting at (inclusive)
  + size: size from start node to end node
  + threshold: Use insertion sort when the DoublyLinkedList's size is **less than or equal**to the threshold
  + return: None - DLL class should be altered
  + **must be recursive**
* partition(low, high)
  + This function will take a set of nodes between and including low to high. It then partitions the nodes according to their values by moving all nodes **greater than or equal**to the "high" node which is the pivot to the right of it and all nodes **less**than to the left.
  + low: beginning node to start partitioning at
  + high: last node to partition at used as pivot
  + return: tuple of pivot node and new size from the start to pivot

**InsertionSort.py:**

* insertion\_sort(dll, low, high)
  + This function will sort a doubly linked list from the given low and high points using insertion sort
  + dll: the doubly linked list that is being sorted - used for insertion\_sort wrapper
  + low: node to start sorting at (inclusive)
  + high: node to end sorting at (inclusive)
  + return: None - DLL class should be altered

**Application Problem**

**DoublyLinkedList.py:**

DLL Class

* count\_unique()
  + You are working at Google on a project involving a large data set. To send this information to a supplier, it needs to be at its shortest length to preserve memory for delivery. You are then put to the task to devise an algorithm that takes a doubly linked list of random integers and represents all distinct numbers in order with their count proceeding them. When an element occurs only once, you do not supply the value of its count. (You are allowed to use a string in order to help display the count across nodes.)
  + Examples:
    - input: 1->1->1->2->2->3->3->4
    - output: 1->**3**->2->**2**->3->**2**->4
    - input: 1->1->1->1->1->1->1->1->1->1->1->1->2->2->2
    - output: 1->**1->2**->2->**3**
  + Time Complexity O(nlogn)
  + Space Complexity O(logn)

Each test case will provide:

1. List: list of values passed through the constructor of the DLL
2. Int: A threshold to be used when choosing a sorting algorithm

In addition to the Mimir testing, you will also be graded on the **run time**performance of each sorting algorithm. See below what is expected for each function.

* **Quick Sort**
  + Time Complexity
    - Best case:**O(nlogn),**Average case:**O(nlogn),**Worst case: **O(n2)**
  + Space Complexity
    - **O(logn)**- quick sort uses only constant additional space before making any recursive call. Quicksort must store a constant amount of information for each nested recursive call. Since the best case makes at most *O*(log *n*)nested recursive calls, it uses *O*(log *n*)space.
* **Partition**
  + Time Complexity
    - **θ(n)**
  + Space Complexity
    - **O(1)**
* **Insertion Sort**
  + Time Complexity
    - Best case:**O(n),**Average case:**O(n2),**Worst case: **O(n2)**
  + Space Complexity
    - **O(n)**
* **count\_unique [Application]**
  + Time Complexity
    - **θ(nlogn)**
  + Space Complexity
    - **O(logn)**

Assignment Notes

* You are required to add and complete the docstring for each function. Use Project1 as a guideline to help you document your code when refering to the doubly linked lists.
* You may not use Python Lists or any other containers in this project.
* You may**not access DoublyLinkedList member variables in QuickSort.py & InsertionSort.py**
* You may access DoublyLinkedList member variables in DoublyLinkedList.
* You will be tested on the amount of calls you make to insertion sort
  + sizes of 0 & 1 will be ignored

Rubric:

**MIMIR TEST CASES:**75 points

**MANUAL GRADING:**

|  |  |  |  |
| --- | --- | --- | --- |
|  | Total | Space | Time |
| docstrings | 5 | 0 | 0 |
| quick\_sort | 6 | 3 | 3 |
| partition | 4 | 2 | 2 |
| insertion\_sort | 4 | 2 | 2 |
| application | 6 | 3 | 3 |
| *Total* | ***25*** |  |  |

**Test Cases (75) + Manual Grading (25): 100 points**

*Project written by Anna De Biasi*