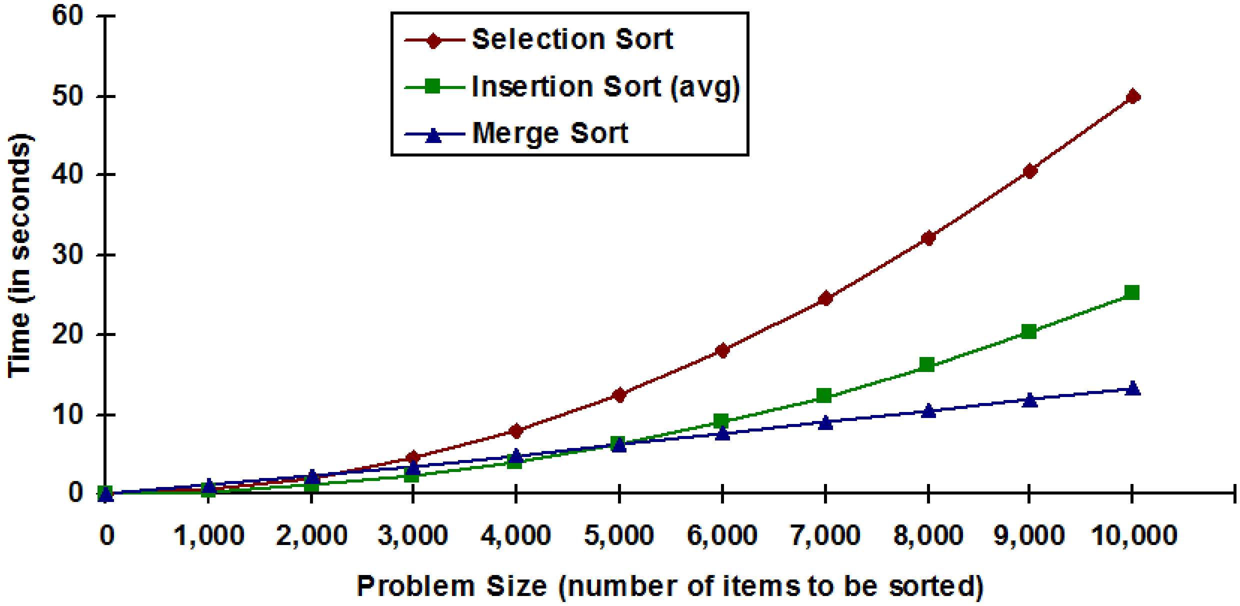
Project 3: Hybrid Sorting

Due: Thursday, February 14th 8:00 pm
This is not a team project, do not copy someone else's work.



Description

For this project, you will be implementing a hybrid sort using Merge Sort and Insertion Sort. Due to the overhead of recursively splitting containers, Insertion Sort may be preferred at small list sizes. You will be sorting a singly linked list using Merge Sort until the partitioned linked lists are **less than or equal to** a given threshold, at which point you will switch to Insertion Sort.

Turning It In

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

- mergesort.py, a python3 file.
- LinkedList.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 two files, **mergesort.py & LinkedList.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.**

mergesort.py:

- merge_lists(lists, threshold)
  - *lists*: a list of *n* different unsorted LinkedLists
  - *threshold*: Use insertion sort when the LinkedList is smaller than or equal to the threshold
  - This function will sort and combine every LinkedList in lists and return the final LinkedList.
- merge_sort(linked_list, threshold)
  - linked_list: an unsorted singly LinkedList
  - *threshold*: Use insertion sort when the LinkedList is smaller than or equal to the threshold
  - this function will use merge sort to sort the given linked list.
  - return the sorted linked list
  - must be recursive
- split_linked_list(linked_list)
  - This function will take a linked list and split it in half.
  - If the size is **odd**, split it into sizes (n/2, n/2 +1)
  - return a tuple of 2 linked lists.
- merge(list1, list2)
  - This function takes in 2 **sorted** LinkedLists and merges them together.
  - return one sorted linked list

LinkedList.py:

- LinkedList.insertion_sort(self)
  - Use insertion sort to sort the current instance of the linked list.
  - **This is the only function where you can access member variables directly (head, tail, size)**

Each test case will provide:

1. List: A list of linkedlists to combine / sort
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.

- **Merge Lists:**

  - Time Complexity
    - **O(mnlgn)**
    - n: Linked List size
    - m: Amount of Linked Lists
  - Space Complexity
    - **O(nm)**
- **Merge Sort**
  - Time Complexity
    - **θ(nlgn)**
  - Space Complexity
    - **O(n)**
- **Merge**
  - Time Complexity
    - **θ(n+m)**
    - n: size of first list
    - m: size of second list
  - Space Complexity
    - **O(n+m)**
- **Split Linked List**
  - Time Complexity
    - **O(n)**
  - Space Complexity
    - **O(n)**
- **Insertion Sort**
  - Time Complexity
    - Best case:**O(n),** Average case: **O(n^2),** Worst case: **O(n^2)**
  - Space Complexity
    - **O(n)**

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.
- You may not use Python Lists or any other containers in this project.
- You may **not access LinkedList member variables in mergesort.py**
- You may access LinkedList member variables in LinkedList.
- 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:
All Test Cases __ / 75

RUNTIME / SPACE / RECURSIVE (MANUAL GRADING)
merge_sort __ / 8 *(Space (4), Time (4))*
merge __ / 5 *(Space (2), Time (3))*
split_linked_list __ / 5 *(Space (2), Time (3))*
insertion_sort __ / 7 *(Space (3), Time (4))*

Total: __ / 100

Project written by Nathan Rizik