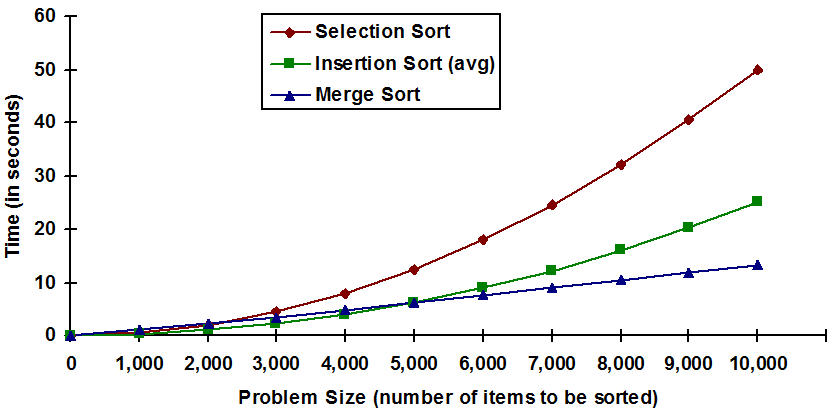
# **Project 3: Hybrid Sorting**

**Due: Thursday, September 27th 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 lists, Insertion Sort may be preferred at small list sizes. You will be sorting a list using Merge Sort until the partitioned 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:

* HybridSort.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.

### Assignment Specifications

You are given one file, **HybridSort.py**. You must complete and implement the following functions in **HybridSort.py**. Take note of the specified return values and input parameters. **Do not change the function signatures.**

* **merge\_sort(unsorted, threshold, reverse)**
  + Use merge sort (Recursively!) to sort and return the list unsorted. merge\_sort must be written recursively.
  + When splitting lists in half, once the list reaches a size less than or equal to the threshold, use insertion\_sort.
  + Sort the list in increasing order if reverse is False, otherwise in decreasing order.
* **merge(first, second, reverse)**  
  + Given two lists, first and second, merge them into a single, sorted list and return it.
  + Sort the list in increasing order if reverse is False, otherwise in decreasing order.
* **insertion\_sort(unsorted, reverse)**
  + Use Insertion Sort to sort and return the list unsorted
  + Sort the list in increasing order if reverse is False, otherwise in decreasing order.

Each test case will provide:

1. List: The list to sort
2. Int: A threshold to be used when switching sorting algorithms
3. Bool: The order to sort the given list. (Decreasing or Increasing)
   * Note:**All sorting should be done without the help of slicing or reversing a list after it has already been sorted.**
   * #The following is NOT allowed:
   * unsorted = unsorted[::-1]
   * unsorted = [i for i in reversed(unsorted)]

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 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)**
* **Insertion Sort**
  + Time Complexity
    - Best case:**O(n),**Average case:**O(n2),**Worst case: **O(n2)**
  + Space Complexity  
    - **O(1)**

### 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.

### Rubric

MIMIR TEST CASES:

All Test Cases   \_\_ / 70

RUNTIME / SPACE / RECURSIVE (MANUAL GRADING)

merge\_sort      \_\_ / 10 (Space (5), Time (5))

merge             \_\_ / 10 (Space (5), Time (5))

insertion\_sort  \_\_ / 10 (Space (5), Time (5))

Total:              \_\_ / 100