**CMPE 180 C Operating System Project**

**SJSU**

**Instructor: Jahan Ghofraniha**

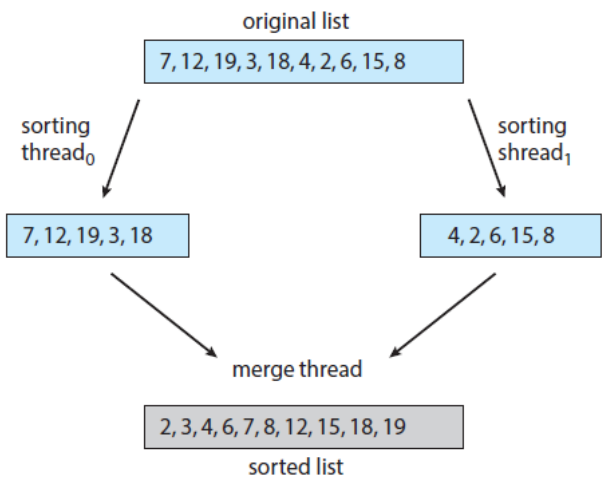
Due date: 5/15/22

This project can be done in teams of three members per team. The project should be submitted as a formal report.

**The project should be written as a formal report and should include the following sections:**

i. Introduction
ii. Problem statement
iii. Software Architecture
iv. Implementation
v. Test & results (this should include unit test and integrated test results)
vi. Name of each member who works on each section.
vii. Conclusions
viii. References (the material should be fully referenced with links to the reference source code or content)
ix. Appendix (should include any extra information related to the project including the source code for your project. If you provide a link to your github, the link should be accessible. If the code is not accessible you will not get any credit for the project)

Write a multithreaded sorting program that works as follows: A list of integers is divided into two smaller lists of equal size. Two separate threads (which we will term ***sorting threads***) sort each sublist using a sorting algorithm of your choice. The two sublists are then merged by a third thread—a ***merging thread***—which merges the two sublists into a single sorted list. Graphically, this program is structured as the following:

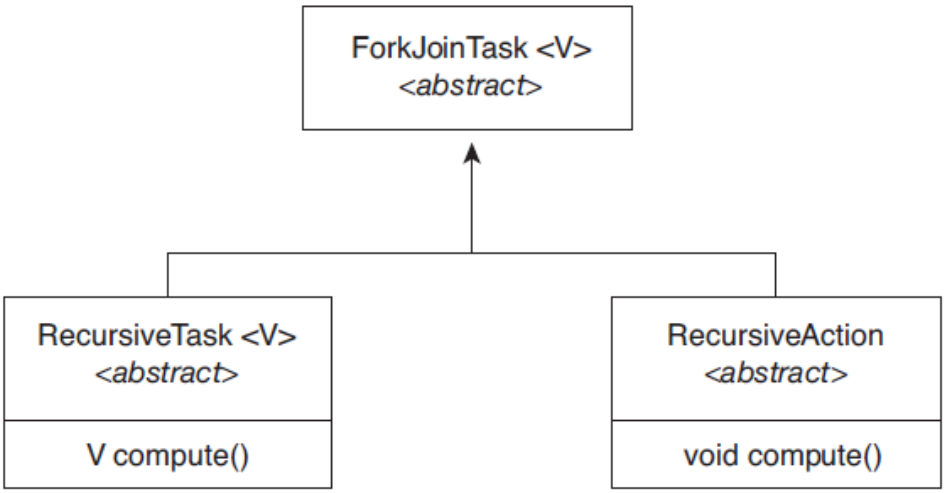


Implement the preceding project (Multithreaded Sorting Application) using Java's fork-join parallelism API. This project will be developed in two different versions. Each version will implement a different divide-and-conquer sorting
algorithm:

- Quicksort
- Mergesort

The Quicksort implementation will use the Quicksort algorithm for dividing the list of elements to be sorted into a *left half* and a *right half* based on the position of the pivot value. The Mergesort algorithm will divide the list into two evenly sized halves. For both the Quicksort and Mergesort algorithms, when the list to be sorted falls within some threshold value (for example, the list is size 100 or fewer), directly apply a simple algorithm such as the Selection or Insertion sort. Refer to data structures texts for description of these two well-known, divide-and-conquer sorting algorithms.

The class SumTask shown in Section 4.5.2.1 of the textbook extends RecursiveTask, which is a result-bearing ForkJoinTask. As this project will involve sorting the array that is passed to the task, but not returning any values, you will instead create a class that extends RecursiveAction, a non-result-bearing ForkJoinTask. The following figure shows the class hierarchy for the above abstract classes.




The objects passed to each sorting algorithm are required to implement Java's Comparable interface, and this will need to be reflected in the class definition for each sorting algorithm.

Implementation options:

You can implement this project in any other languages (Python, C, C++, …) but you may have to implement the comparable interface in that language, i.e. define __lt__ (less than), __eq__ (equal), …, etc.