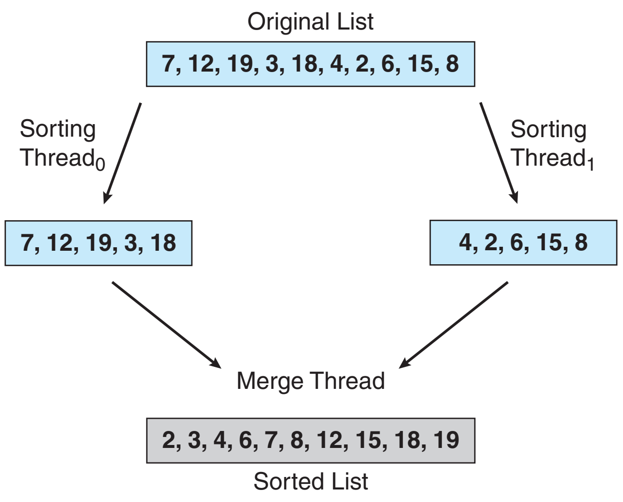
# Programming Assignment #5

- **Due:** Noon on Sunday 18 March 2018. Late submissions (if second time) will be penalized 5%.
- **Team Based:** You may work in groups of up to 3 members.
- **Required Artifacts:**
  - The source code (.c) file. Only one student needs to upload the file to Titanium for the entire group. Write the following information as comments at the beginning of the file.
    - All the student names with the associated CWIDs
      - Ex: Gina Ackerman 135798
    - Assignment #
  - Large screenshot(s) of your program output.
- **Note:** No late assignment will be accepted after 24 hours of the due date, i.e., your group will get 0 points.
- **Grading Rubric:**
  - Build cleanly: 5 points
  - Correct Output: 5 points
  - Functions:
    - main(…) // 30 points
    - sorter(…) // 10 points
    - merger(…) // 20 points

**Multithreaded Sorting Application** (extracted from chapter 4 project 2)
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.

Because global data are shared cross all threads, perhaps the easiest way to set up the data is to create a global array. Each sorting thread will work on one half of this array. A second global array of the same size as the unsorted integer array will also be established. The merging thread will then merge the two sublists into this second array. Graphically, this program is structured as follows:



This programming project will require passing parameters to each of the sorting threads and the merge thread. In particular, it will be necessary to identify the starting index from which each thread is to begin sorting.

```
/* structure for passing data to threads */
typedef struct {
    int row;
    int column;
} parameters;
```

The book shows that the parameters' variable data is dynamically created. It's not necessary. However, you need 3 different parameters, one for each thread.

The parent thread will output the sorted array once all sorting and merge threads have exited.

Output:

```
2  3  4  6  7  8  12  15  18  19
```

Note: For the school Linux server, you must include "-lrt **-lpthread**" as follows:

```
gcc multhreaded_sorting.c -o test -lrt -lpthread
```