Use a more efficient algorithm than the copy-and-sort version in the slides.

Basic concurrency

1) Using std::async, open multiple files in parallel and count the total number of characters across all the files

2) Fill a vector with random integers in the range 1 to 1000. Count the number of odd integers using the optimum number of threads

3) Implement a simple spin lock using an atomic

4) Initialise a global shared object pointed to by an atomic pointer and make sure the object is correctly deleted when the program exits. Do this without using any locks

5) Insert items at the head of a singly-linked list from multiple threads using a non-blocking algorithm using compare and exchange (CAS). Check that the correct number of items have been inserted into the list and that the list structure is correct

6) Create a single-producer multi-consumer queue based on a circular buffer. Use atomics to hold the positions for the writer and the readers. Use non-blocking queue operations throughout (such as spin locks or polling loops)

More concurrency

1) Update the RUList to be thread-safe by locking the appropriate methods. Is this a sensible design approach?

2) Create two bank accounts, each with its own associated lock. Transfer random amounts based on a normal distribution between the two accounts in both directions using several parallel threads. Ensure that the accounts are locked and unlocked correctly to avoid deadlock

3) Initialise the random number engine in the previous exercise using a hardware random number source from only one of the threads

4) Implement a thread-safe fixed-length queue using blocking operations (instead of spin locks or polling)

5) Add push and pop methods that will time out if they block for longer than a given time. Refactor out any duplication

6) Add appropriate memory ordering flags to the circular buffer queue from the previous section