HW-1(CS 6240)

SHUBHAM DEB

B) Weather Data Results(over 10 runs)

	MAX(in ms)	AVG(in ms) /speedup	MIN(in ms)
Sequential	798	608	542
NoLock	552	353(1.72)	290
CoarseLock	567	378(1.60)	319
FineLock	633	429(1.41)	358
NoSharing	931	646(0.94)	592

C) Weather Data Results (in Fibonacci over 10 runs)

	MAX(in ms)	AVG(in ms) /speedup	MIN(in ms)
Sequential	859	609	548
NoLock	593	406/(1.5)	337
CoarseLock	591	479/(1.27)	330
FineLock	681	467/(1.3)	367
NoSharing	935	650/(0.93)	600

Number of worker Threads = 2

Speedup(expected) = 1/(1/2) = 2

1) I expect no-lock to finish the fastest because in no-lock we are executing multiple threads and each thread has a part of data associated with it. Since multiple threads are executing at the same time, it will run faster than sequential. And since there are no locks associated with it, there would be no waiting time associated with execution of multiple threads.
   The weather data results confirm this observation as it has the max speedup among the other 3 multithreading options.
2) I normally expect sequential version to run slowly because multithreading will make execution time faster as we are distributing work and hence the time taken to execute the task should reduce considerably. In sequential, no parallel computation is being done so each record will be read one by one making it time-consuming to read large files.
   But in the experiments, NoSharing is taking more time than sequential because it depends on the task we assign to each thread. If the processing task in each thread is heavy(such as a very large code in function), then it would take more time for the threads to execute than sequential.
3) The program versions of sequential, coarseLock, FineLock and NoSharing return consistent values both with and without Fibonacci function.
   But NoLock is returning ConcurrentException and random average temperature values because of the race conditions occurring between multiple threads as there is no locking specified between the threads.

4) The running time of sequential is more than the coarse lock because of multithreading in coarse lock, so multiple threads are executing different parts of the data at the same time which reduces the time. Hence the processing time of coarse lock is less than that of sequential.
   But in C) I experienced that the average time of sequential is almost the same for both B) and C) ,but in C) the average time of coarse-lock increases considerably as the waiting time for the threads increases due to the added waiting time for executing the Fibonacci function.
5) The higher computation of C) affects fine lock as it will run faster than coarse lock in C) rather than in B). This is due to the fact that in coarse lock multiple threads now have to wait longer for the hashmap which is the lock to be released.
   But in fine lock, this will be overcome by the fact that multiple threads can work on the hashmap at the same time as they can access the different keys of the hashmap at the same time.
   So it increases concurrency and hence less threads have to wait and so the process execution will be faster in the case of fine lock.