**Program Structures & Algorithms**

**Spring 2022**

**Assignment No. 4**

Name: Yucong Liu

(NUID): 002192381

* **Task**

Please see the presentation on *Assignment on Parallel Sorting* under the *Exams. etc.*module.

Your task is to implement a parallel sorting algorithm such that each partition of the array is sorted in parallel. You will consider two different schemes for deciding whether to sort in parallel.

1. A cutoff (defaults to, say, 1000) which you will update according to the first argument in the command line when running. It's your job to experiment and come up with a good value for this cutoff. If there are fewer elements to sort than the cutoff, then you should use the system sort instead.
2. Recursion depth or the number of available threads. Using this determination, you might decide on an ideal number (*t*) of separate threads (stick to powers of 2) and arrange for that number of partitions to be parallelized (by preventing recursion after the depth of *lg t* is reached).
3. An appropriate combination of these.

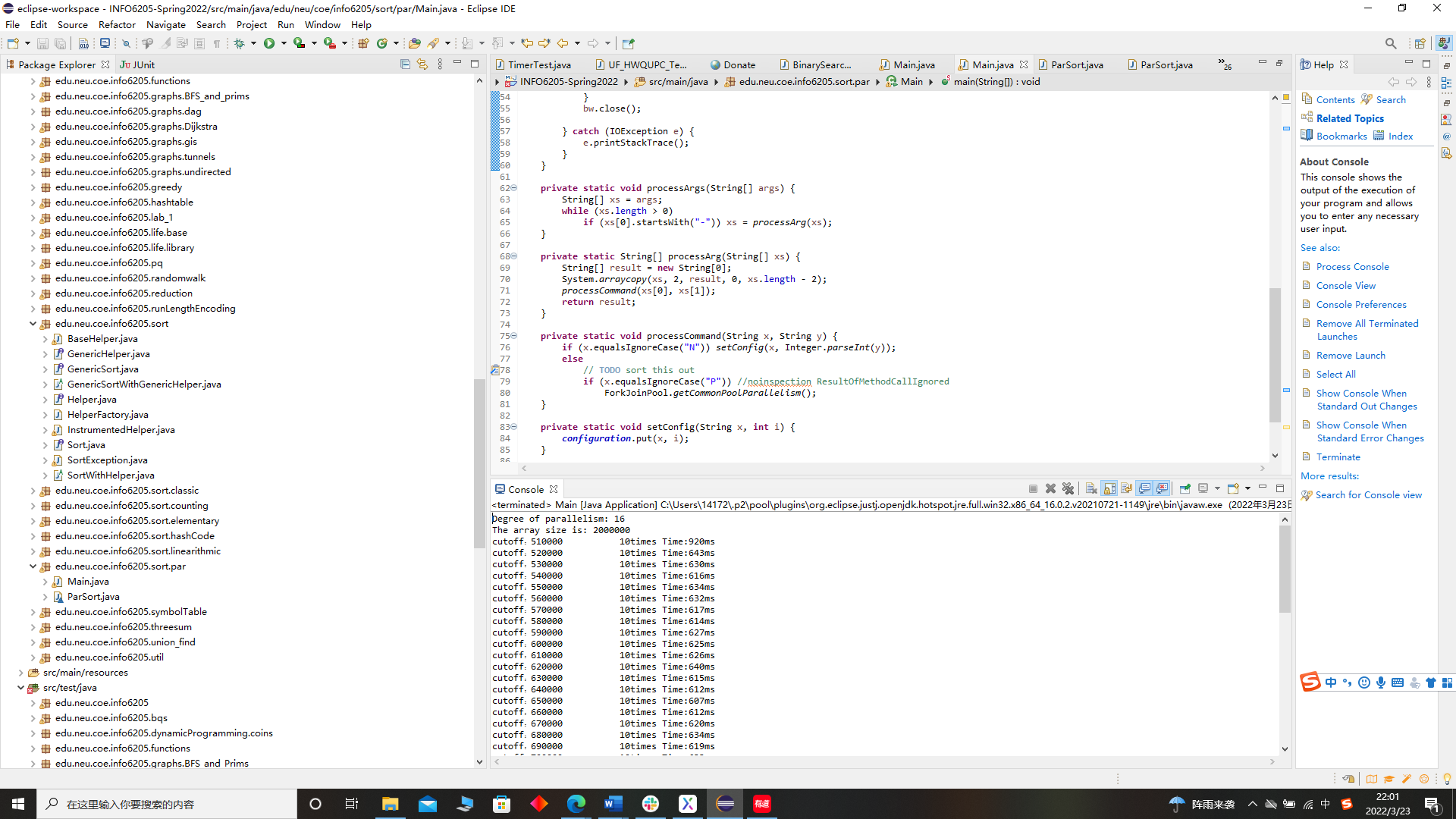
There is a *Main* class and the *ParSort* class in the *sort.par* package of the INFO6205 repository. The *Main* class can be used as is but the *ParSort* class needs to be implemented where you see "TODO..." [it turns out that these TODOs are already implemented].

Unless you have a good reason not to, you should just go along with the Java8-style future implementations provided for you in the class repository.

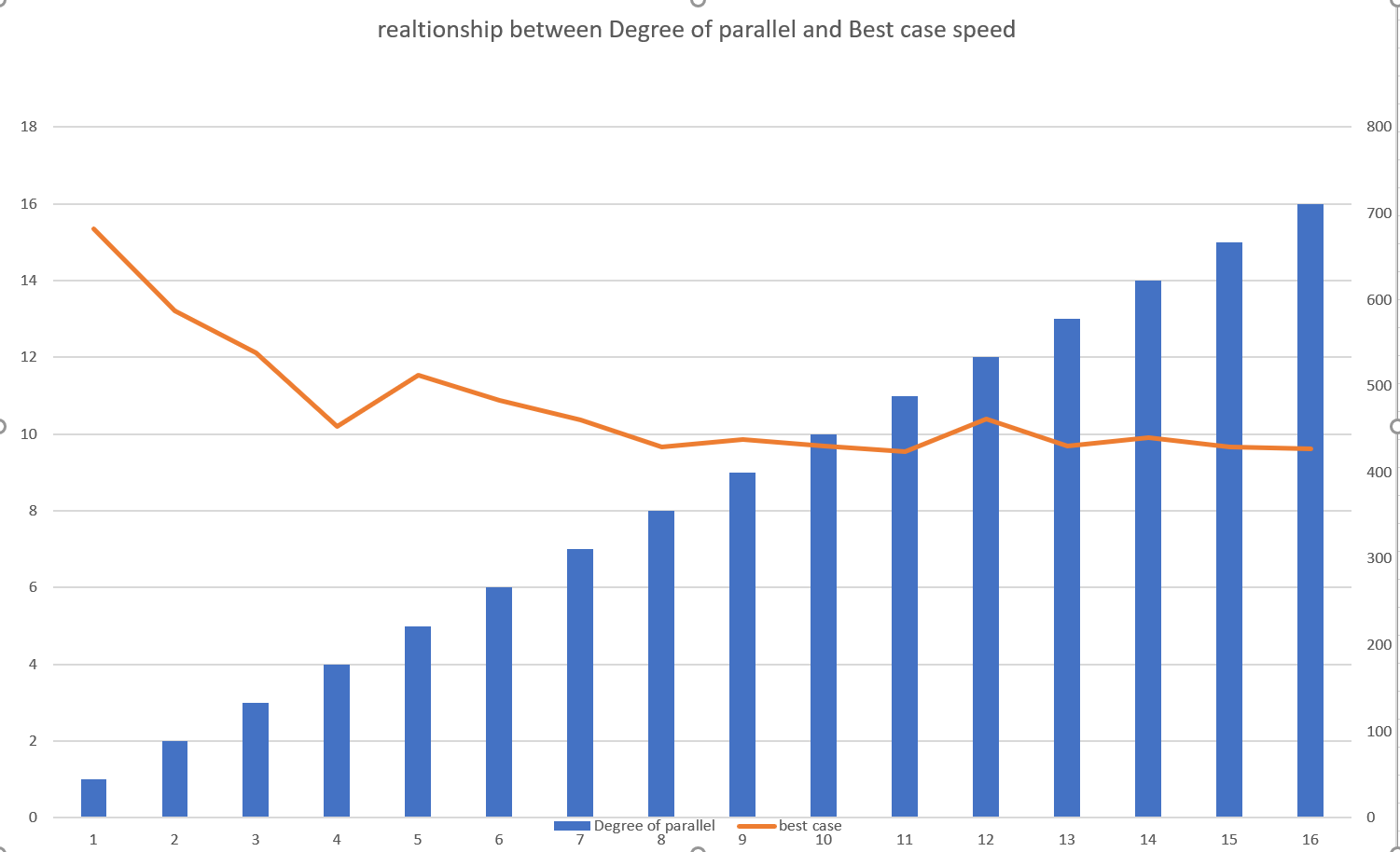
You must prepare a report that shows the results of your experiments and draws a conclusion (or more) about the efficacy of this method of parallelizing sort. Your experiments should involve sorting arrays of sufficient size for the parallel sort to make a difference. You should run with many different array sizes (they must be sufficiently large to make parallel sorting worthwhile, obviously) and different cutoff schemes.

Good luck and enjoy.

* **Output screenshot**

****

* **Relationship Conclusion**
* The computation speed can be optimized by multi-thread parallel sorting. But my experiments show that splitting threads into more than 8 threads leads to very little improvement. There is a margin of error because the workload on each thread is not evenly distributed. But the overall result was what I expected, and sorting with eight threads in parallel on my computer was the best solution.
* **Evidence / Graph**

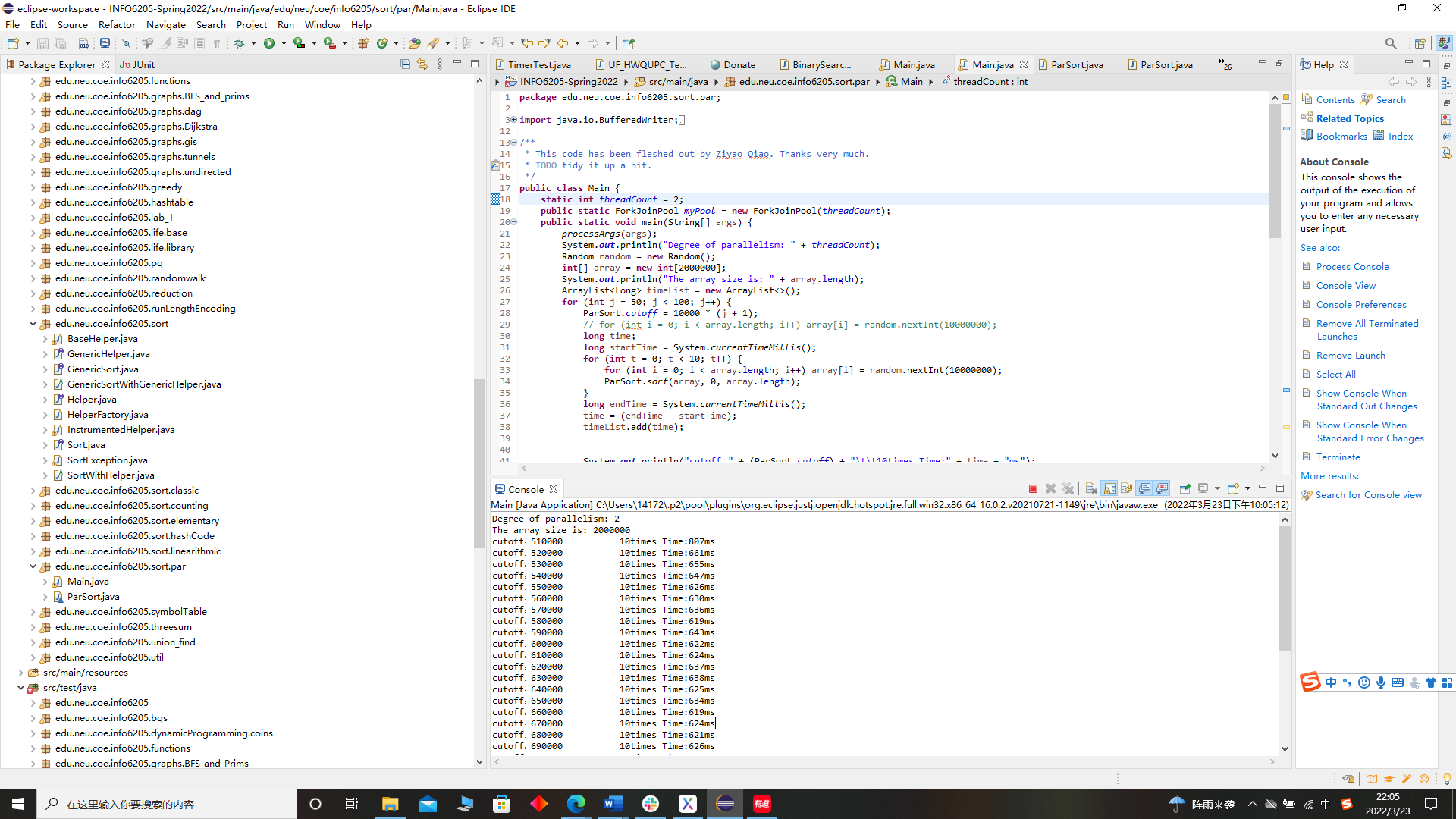


* **Unit tests result**
* **Degree of parallelism from 1 to 16**

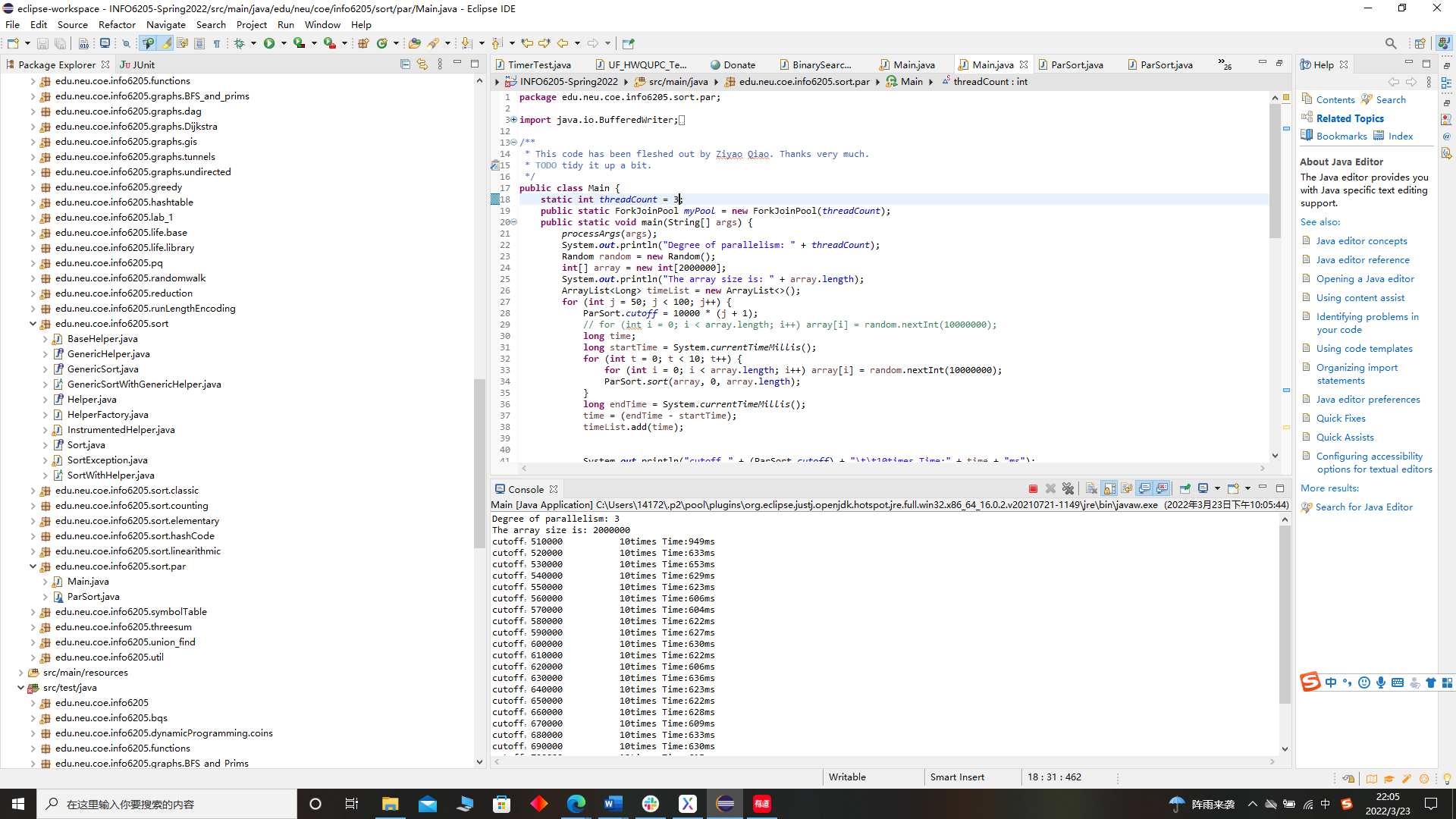
1



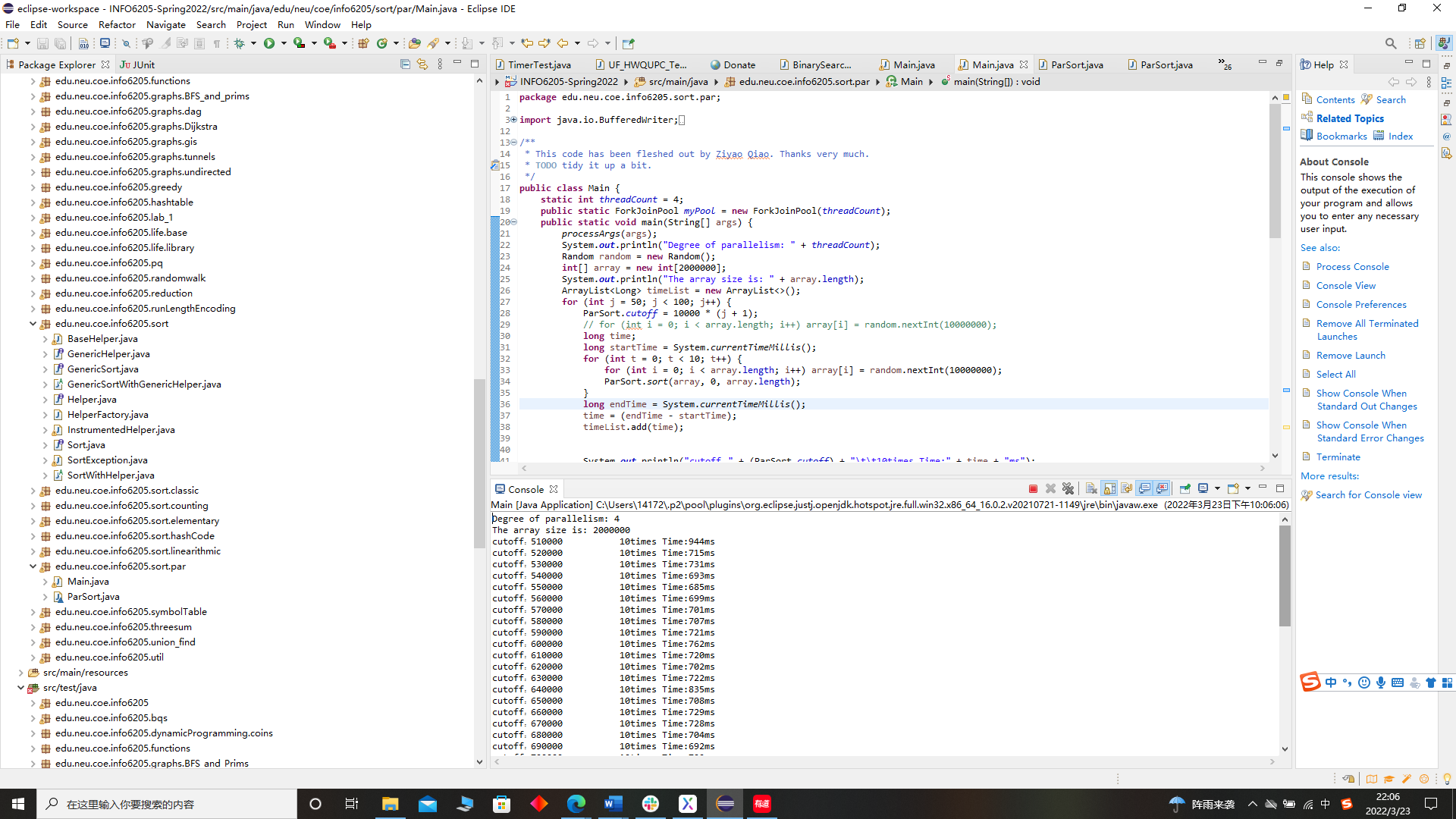
2



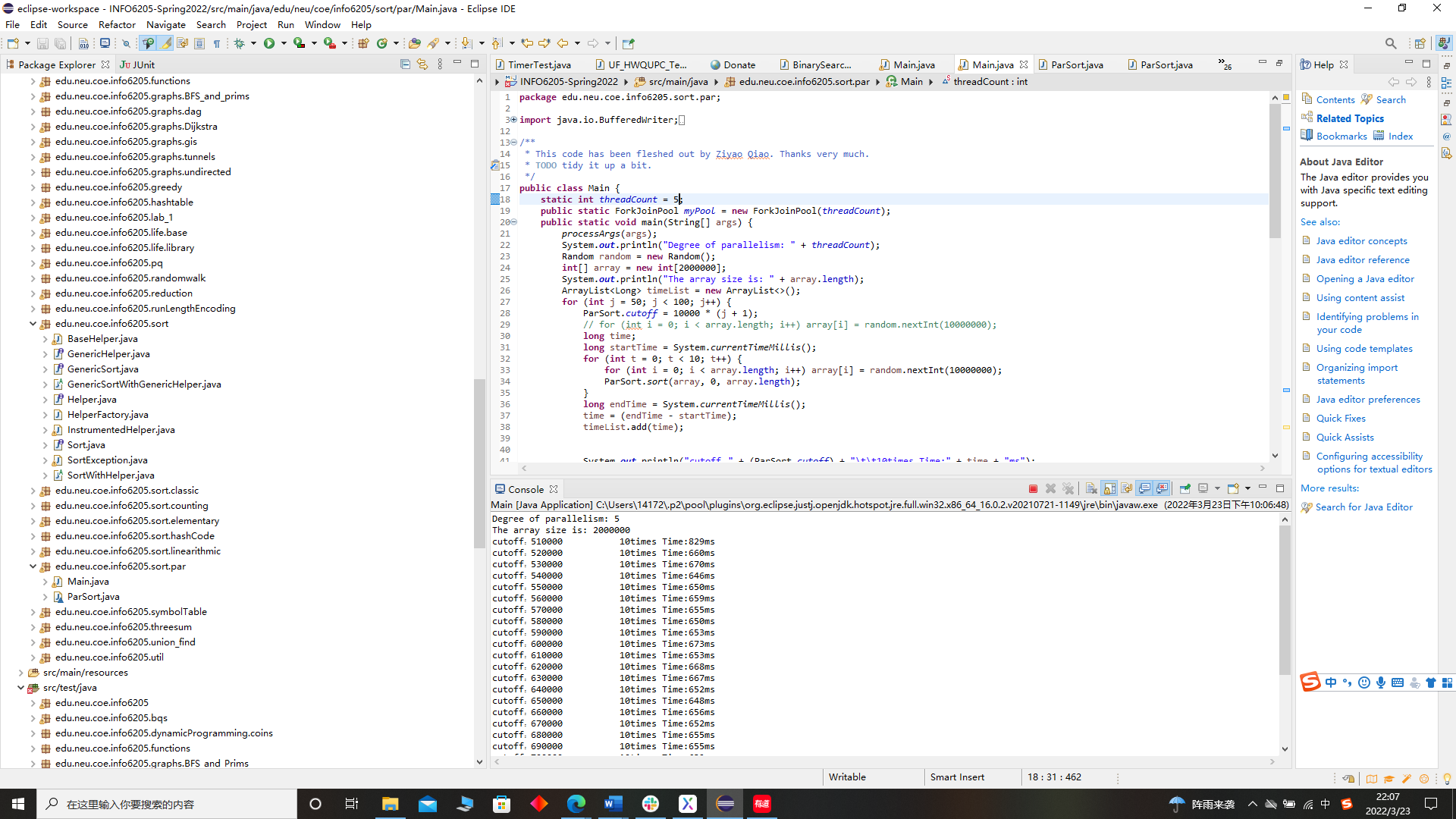
3



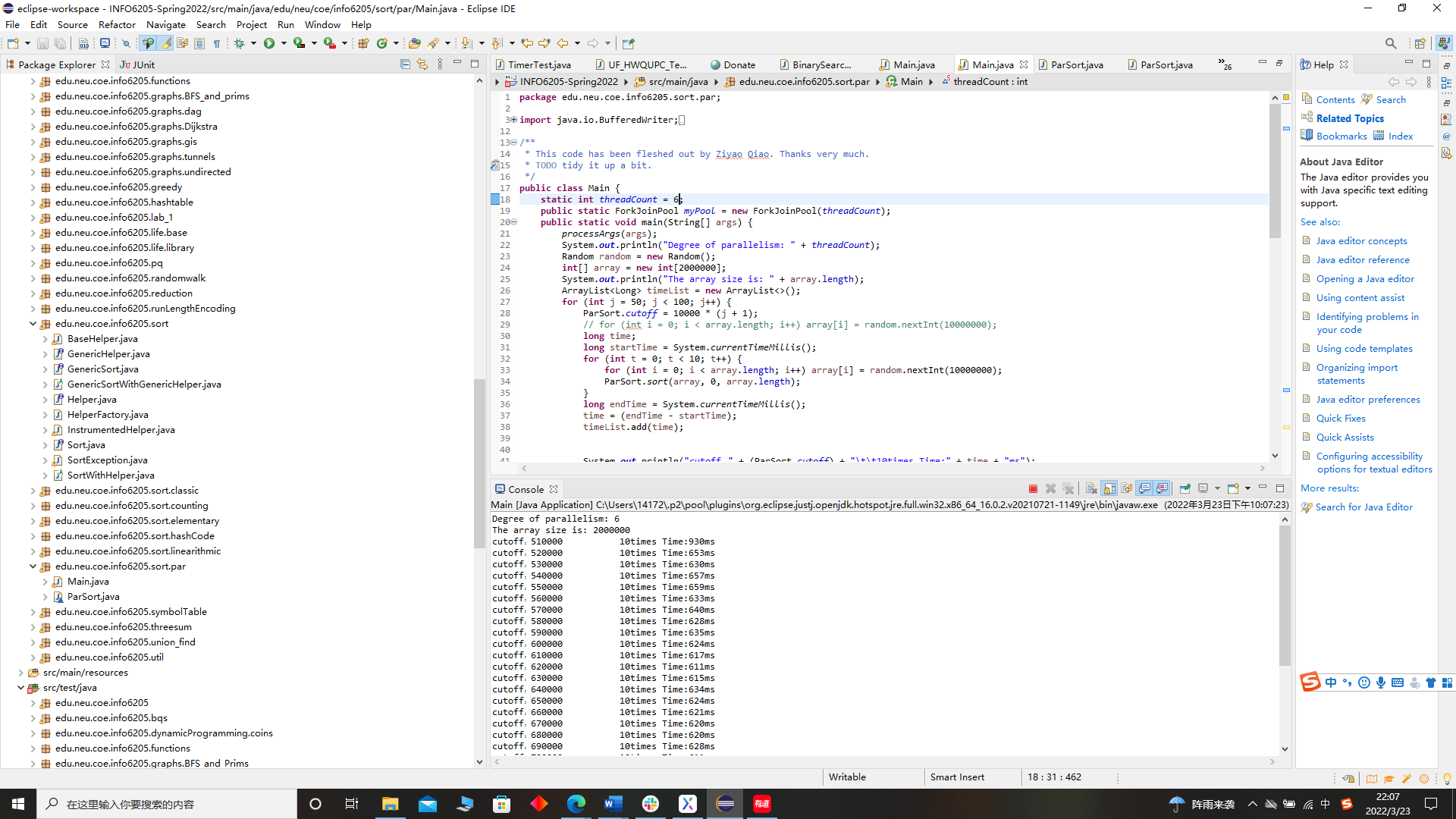
4



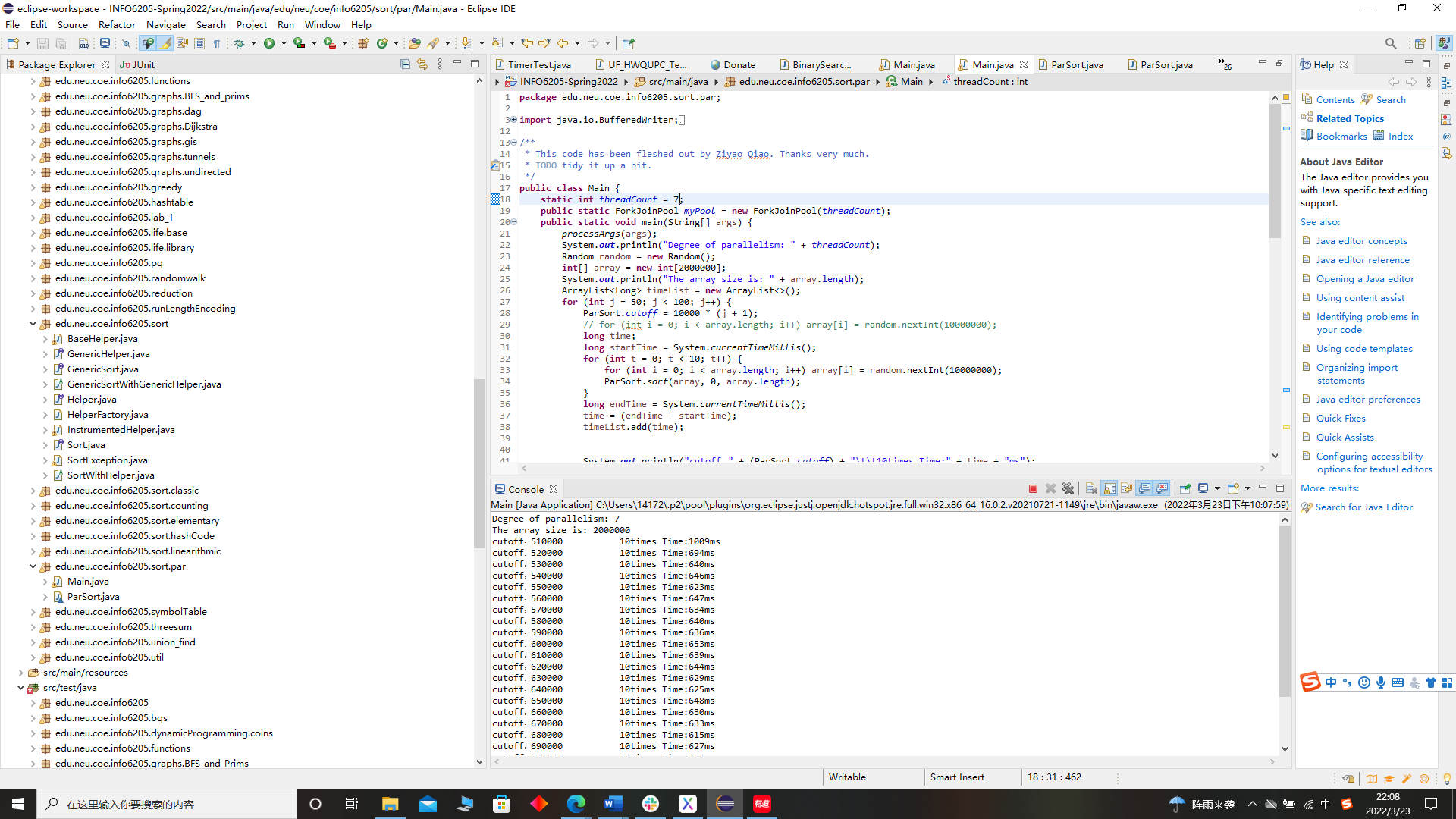
5



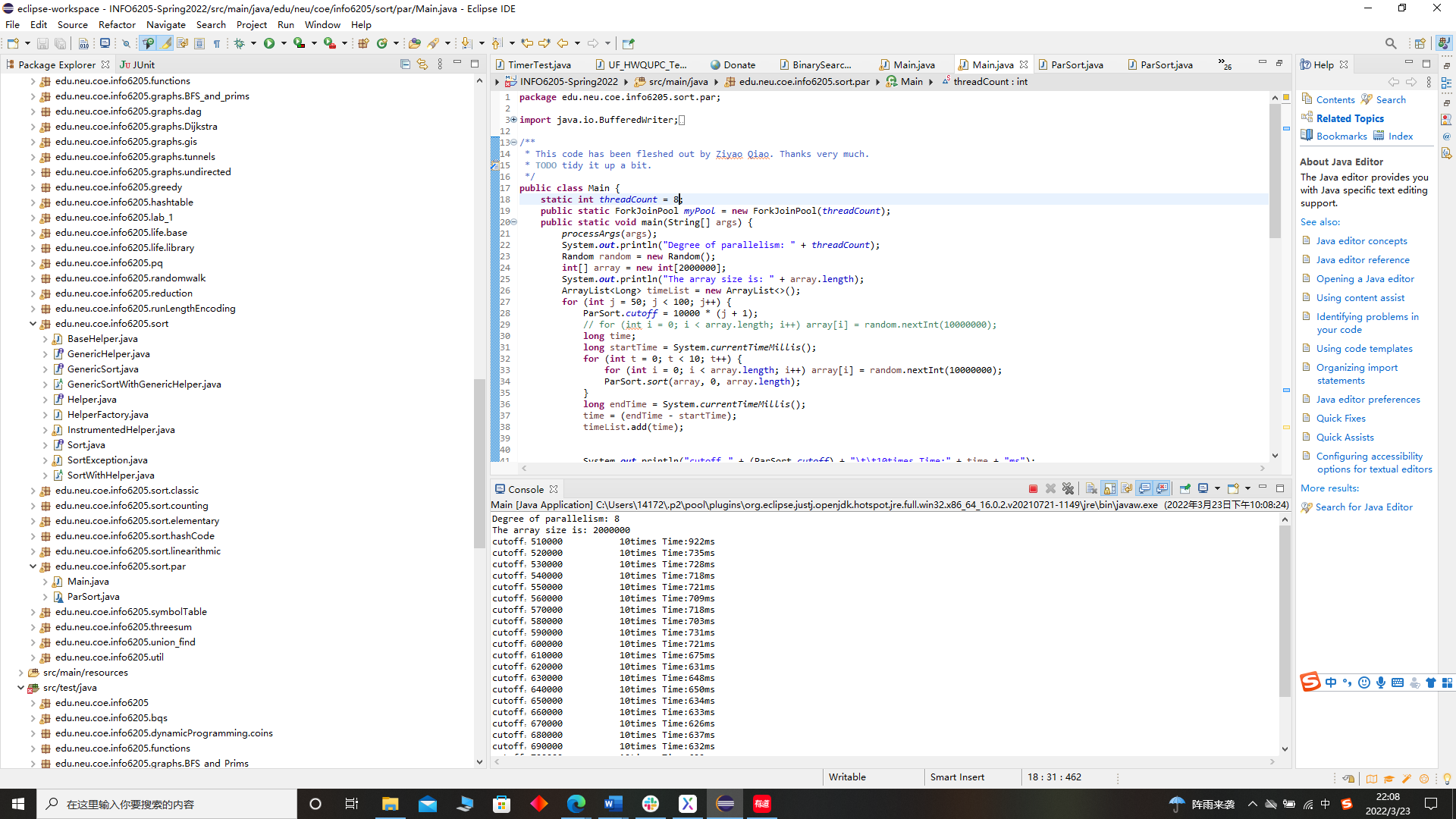
6



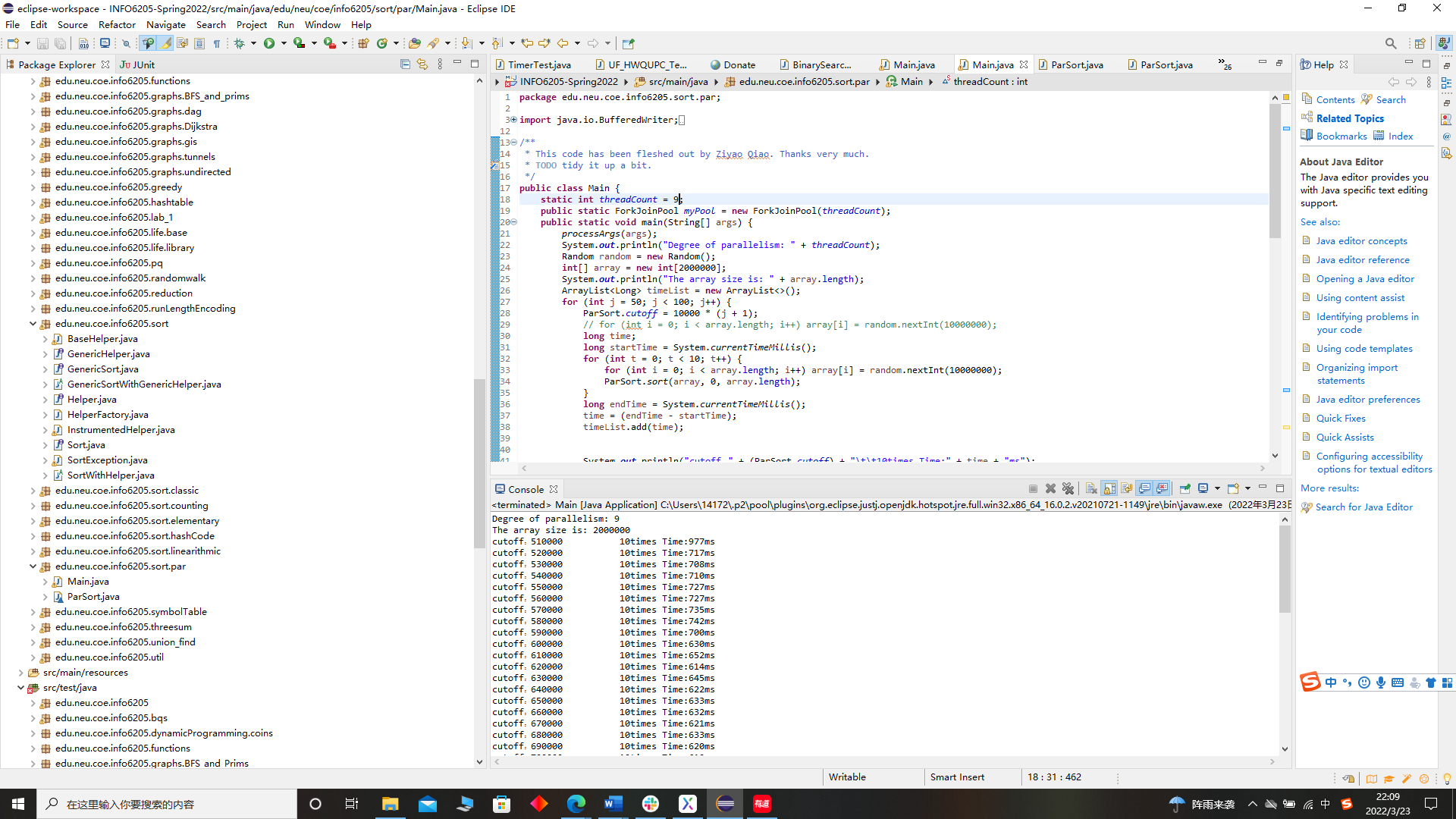
7



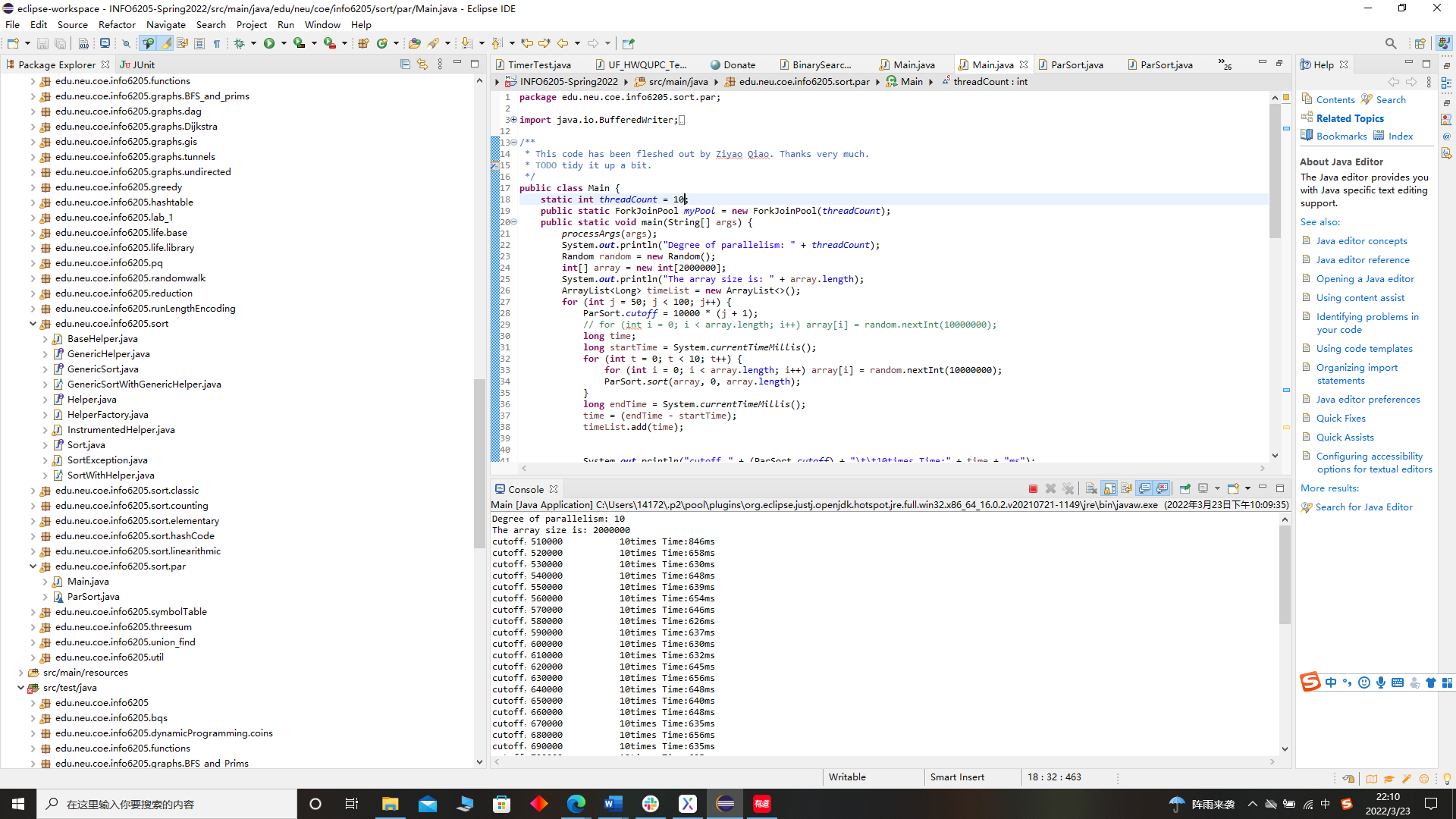
8



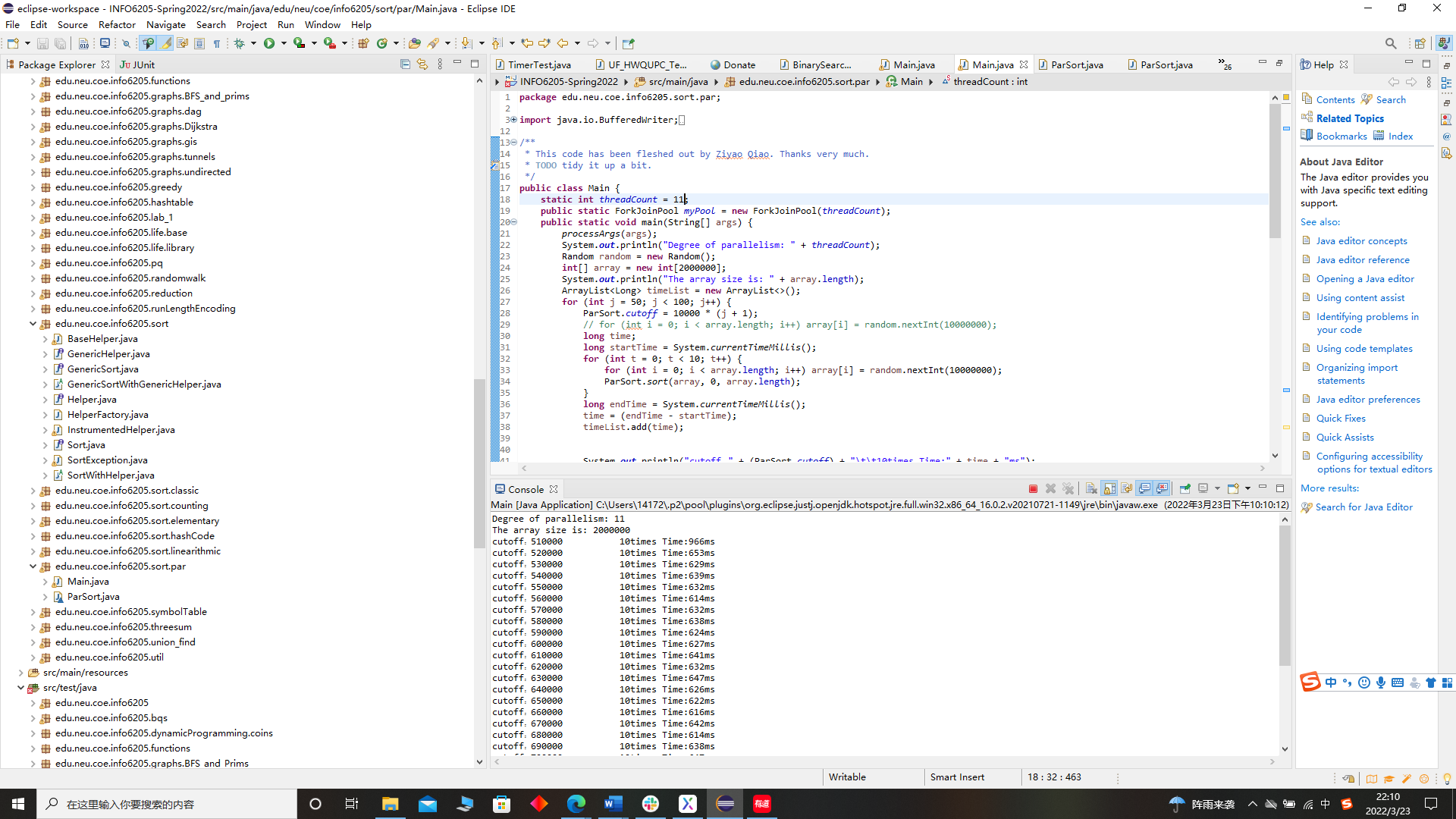
9



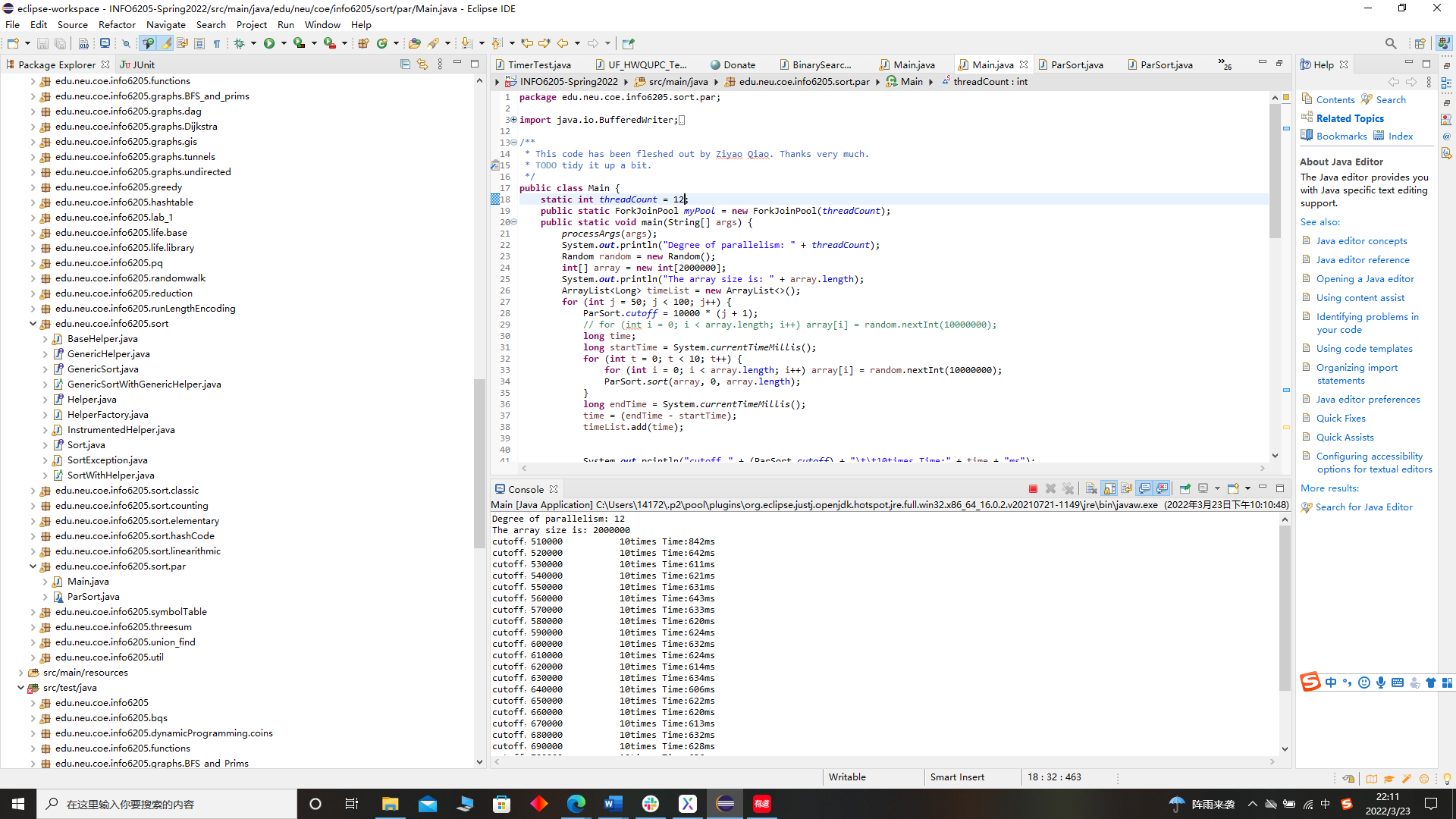
10



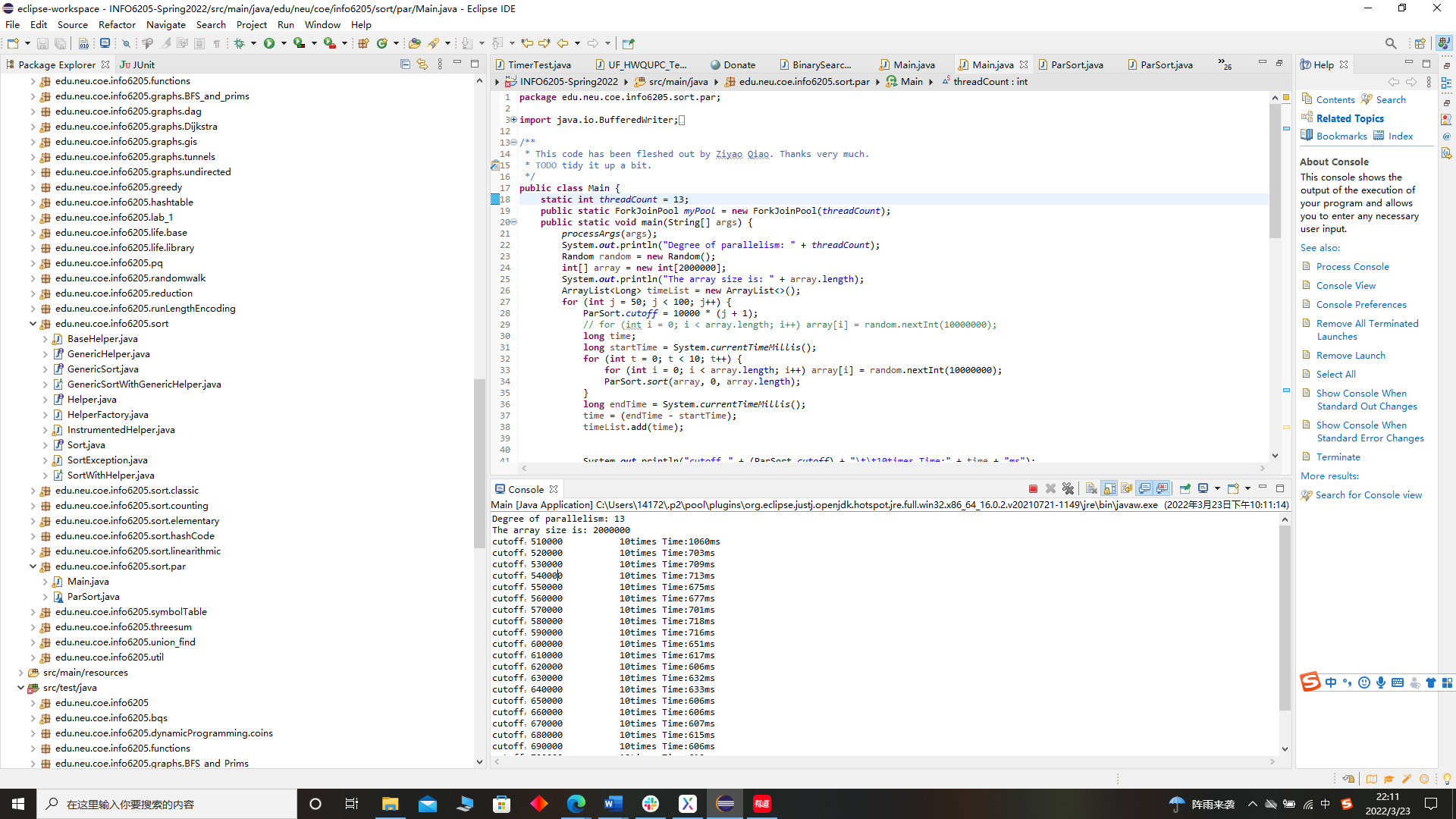
11



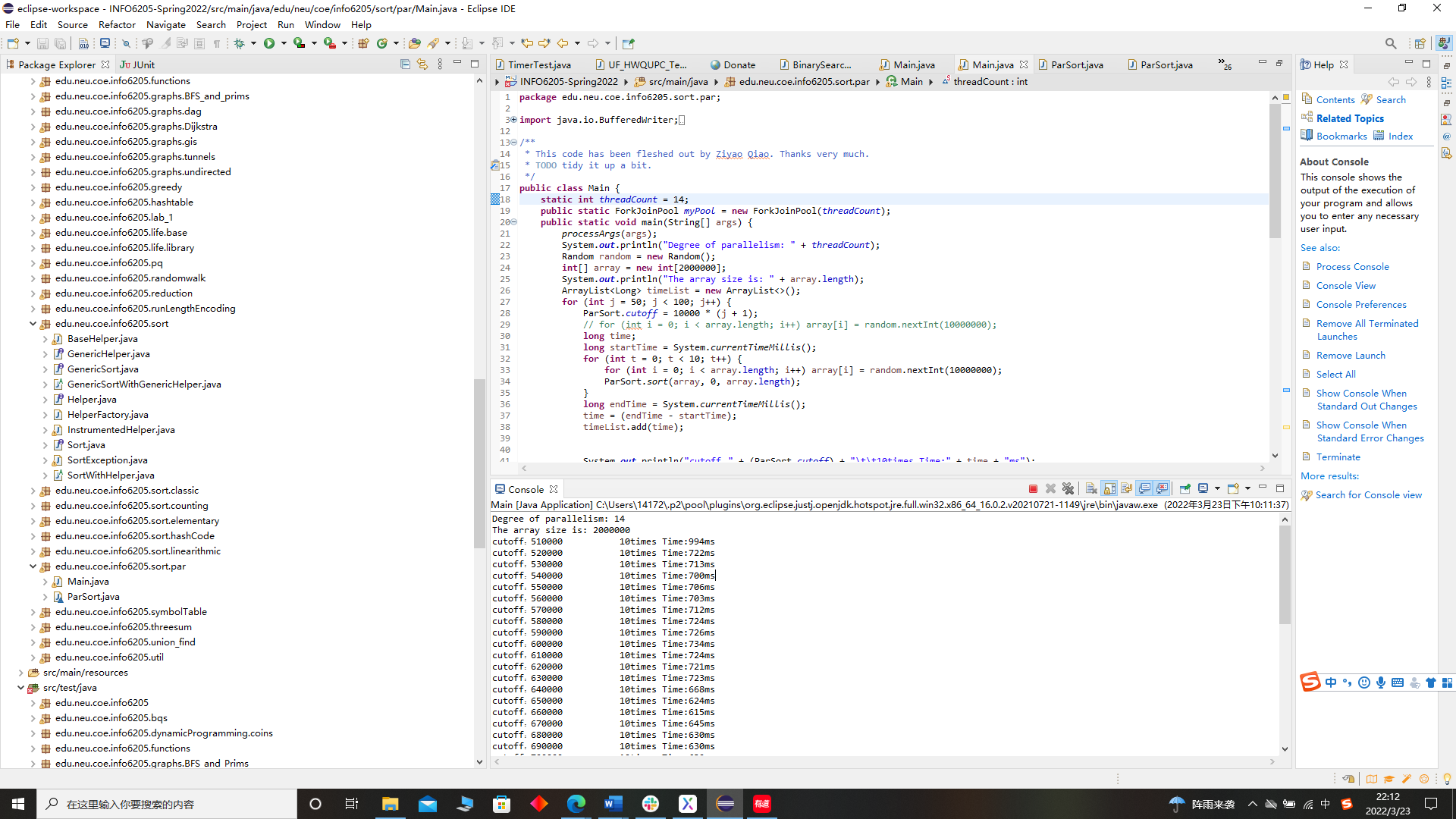
12



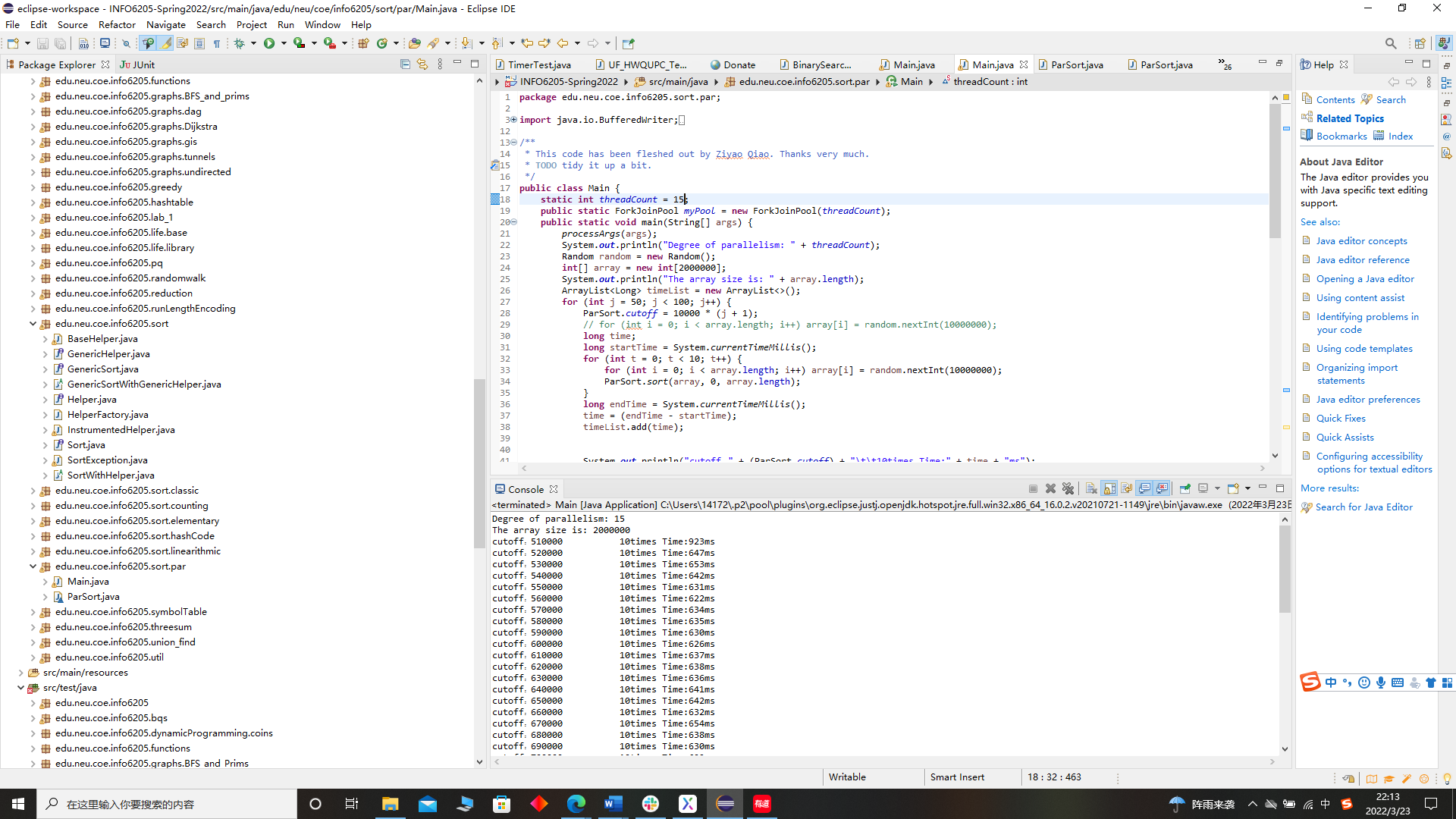
13



14



15



16

