Shreekara SS (001545668)

Program Structures & Algorithms
Fall 2021
Assignment No. 5

Task

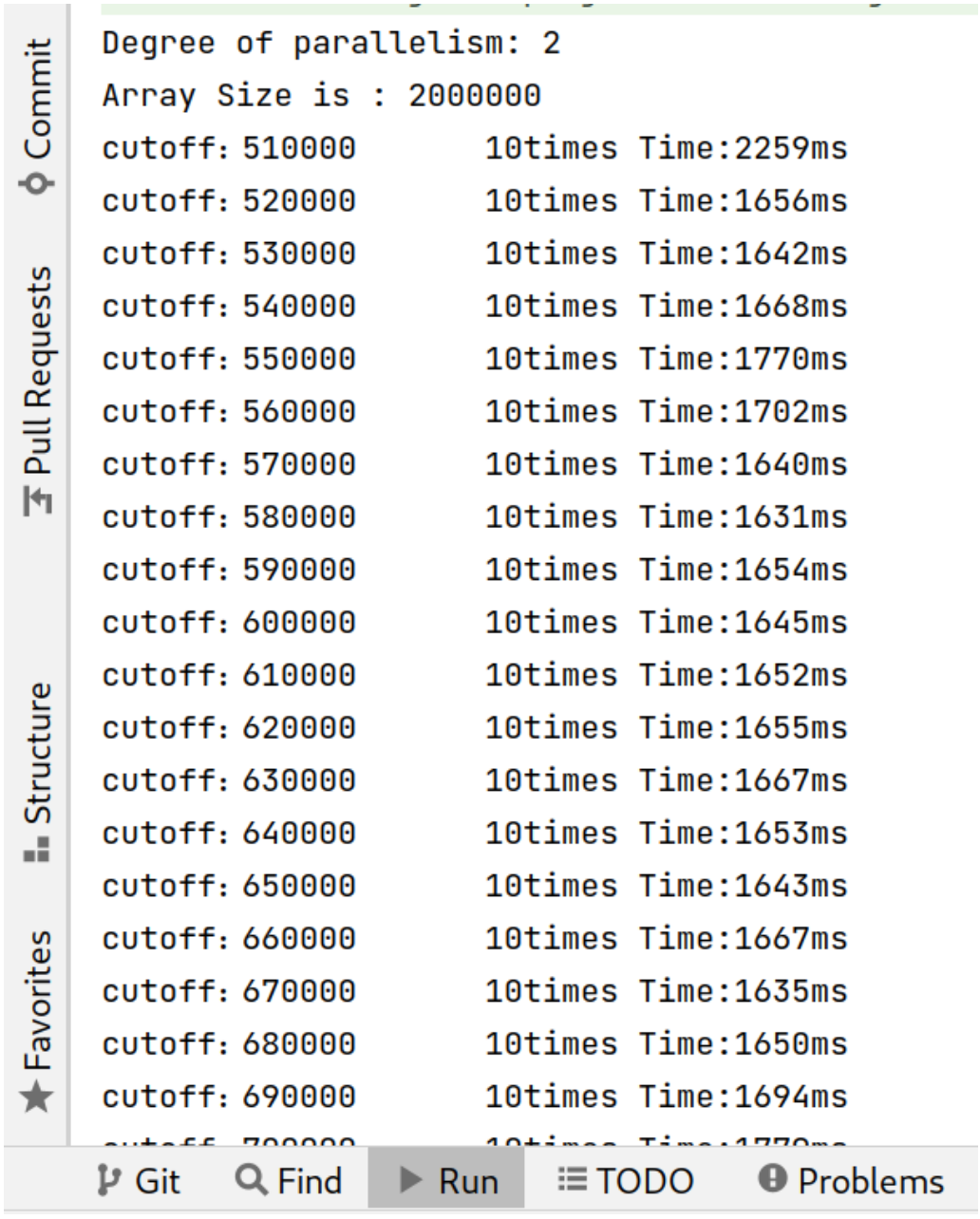
Task is to implement a parallel sorting algorithm such that each partition of the array is sorted in parallel. You will consider two different schemes for deciding whether to sort in parallel.

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.

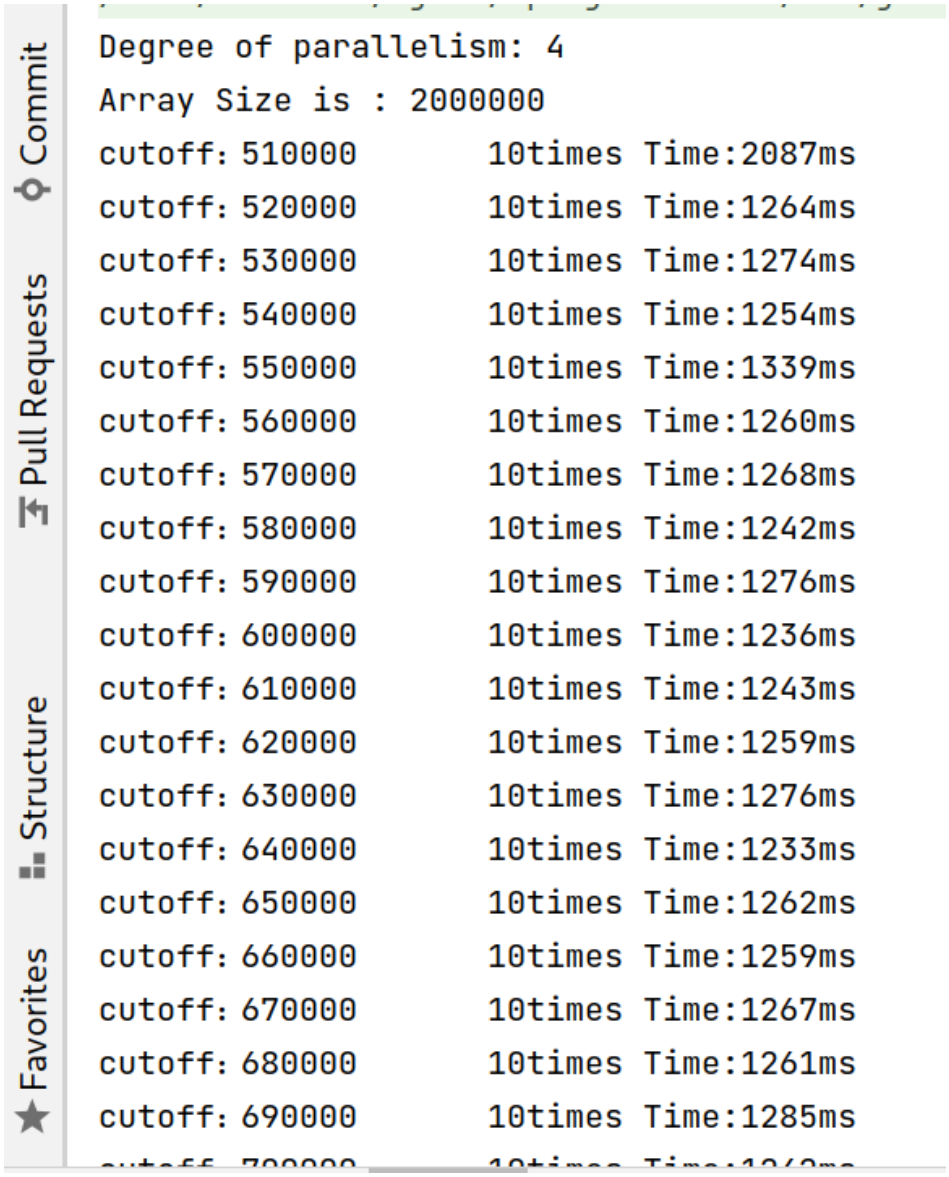
Output

First I wrote code in main.java for thread count (power of 2) and fixed array size.

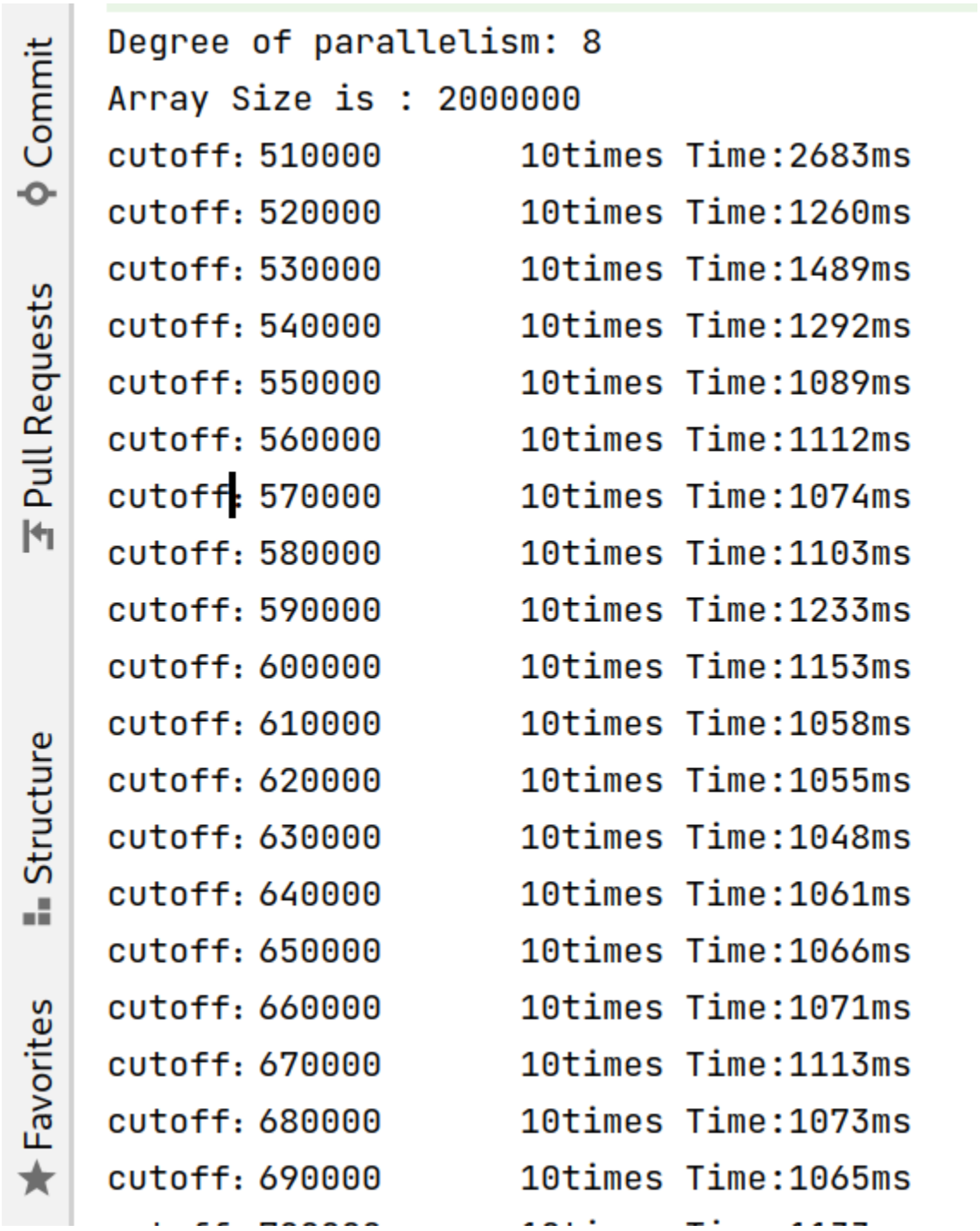
1. Available thread count : 2 ; Array size: 2000000 ; Sorted 10 times(average)



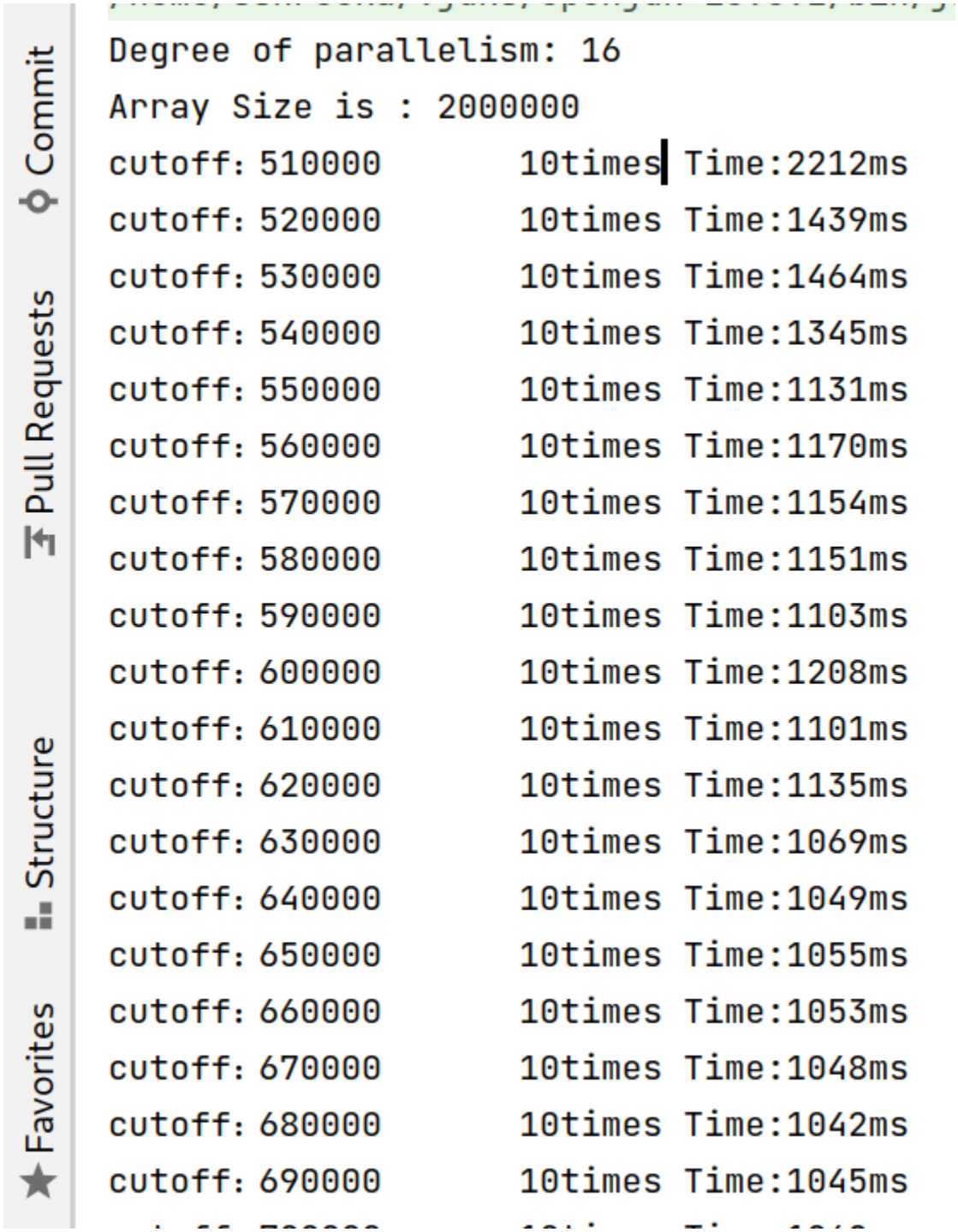
2. Available thread count : 4 ; Array size: 2000000 ; Sorted 10 times(average)



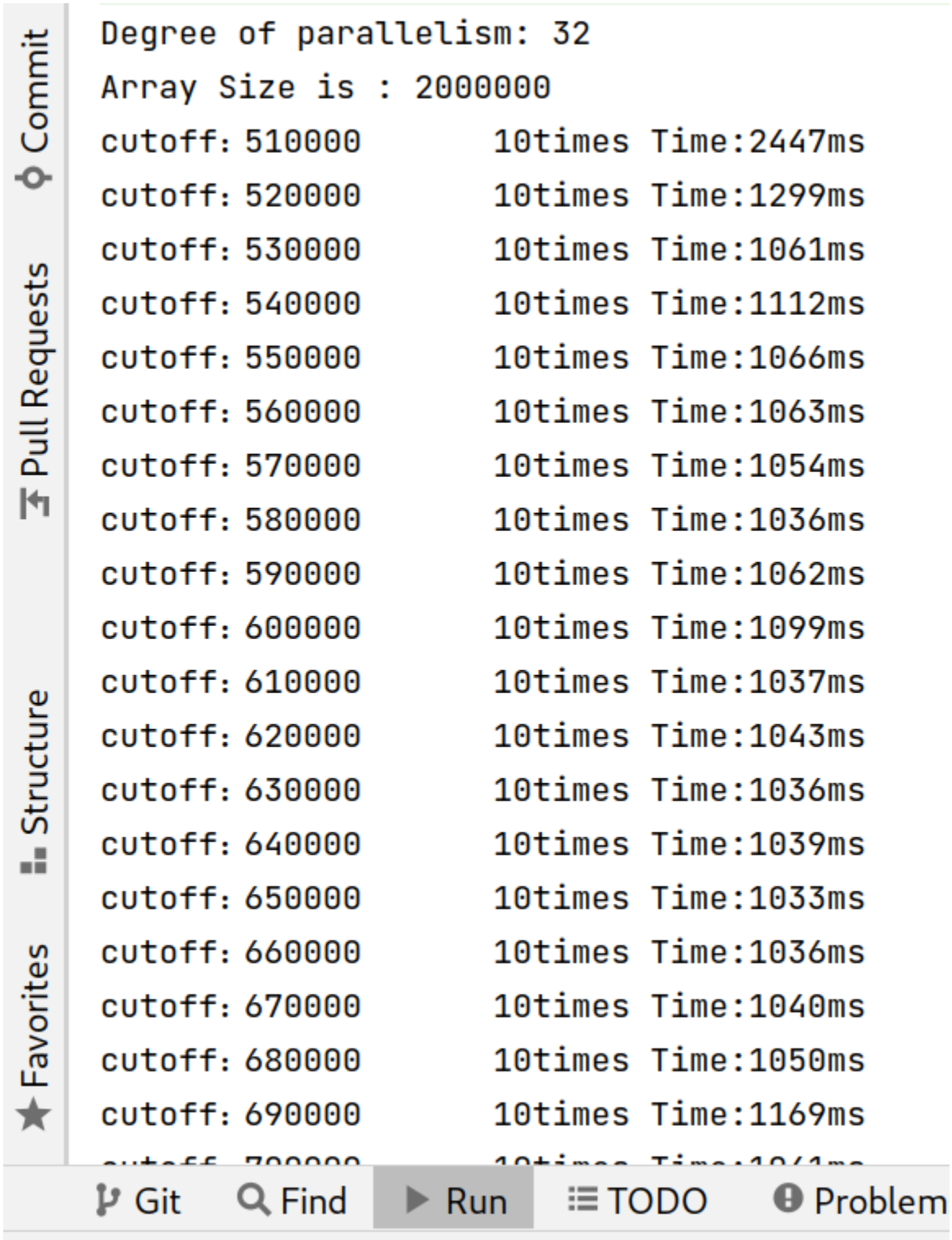
3. Available thread count : 8 ; Array size: 2000000 ; Sorted 10



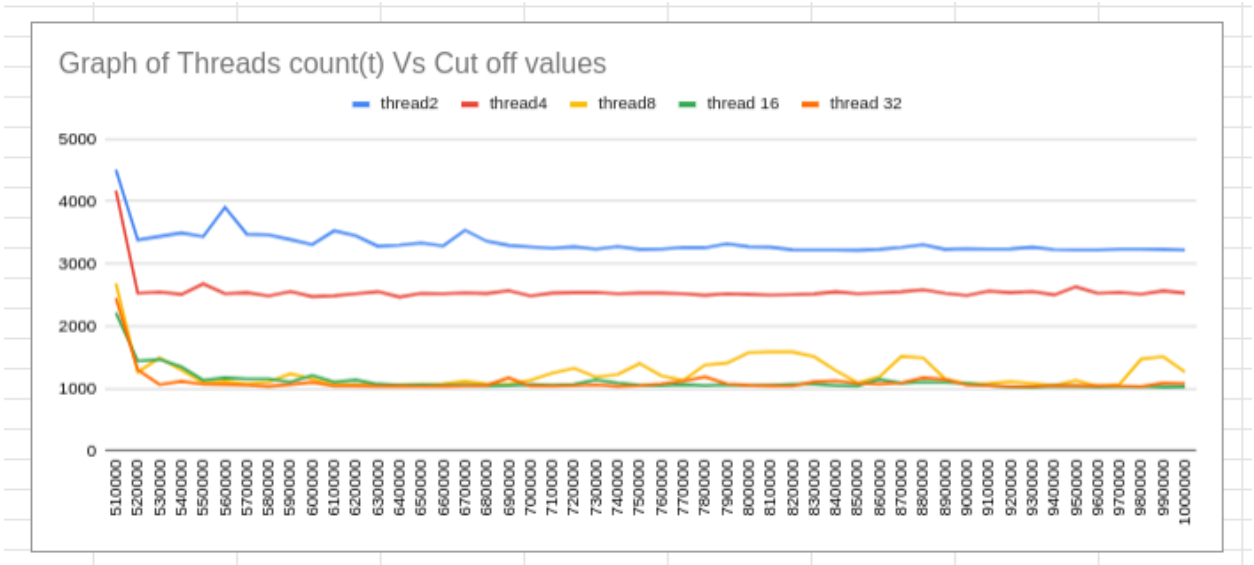
4. Available thread count : 16 ; Array size: 2000000 ; Sorted 10 times(average)



5. Available thread count : 32 ; Array size: 2000000 ; Sorted 10 times(average)



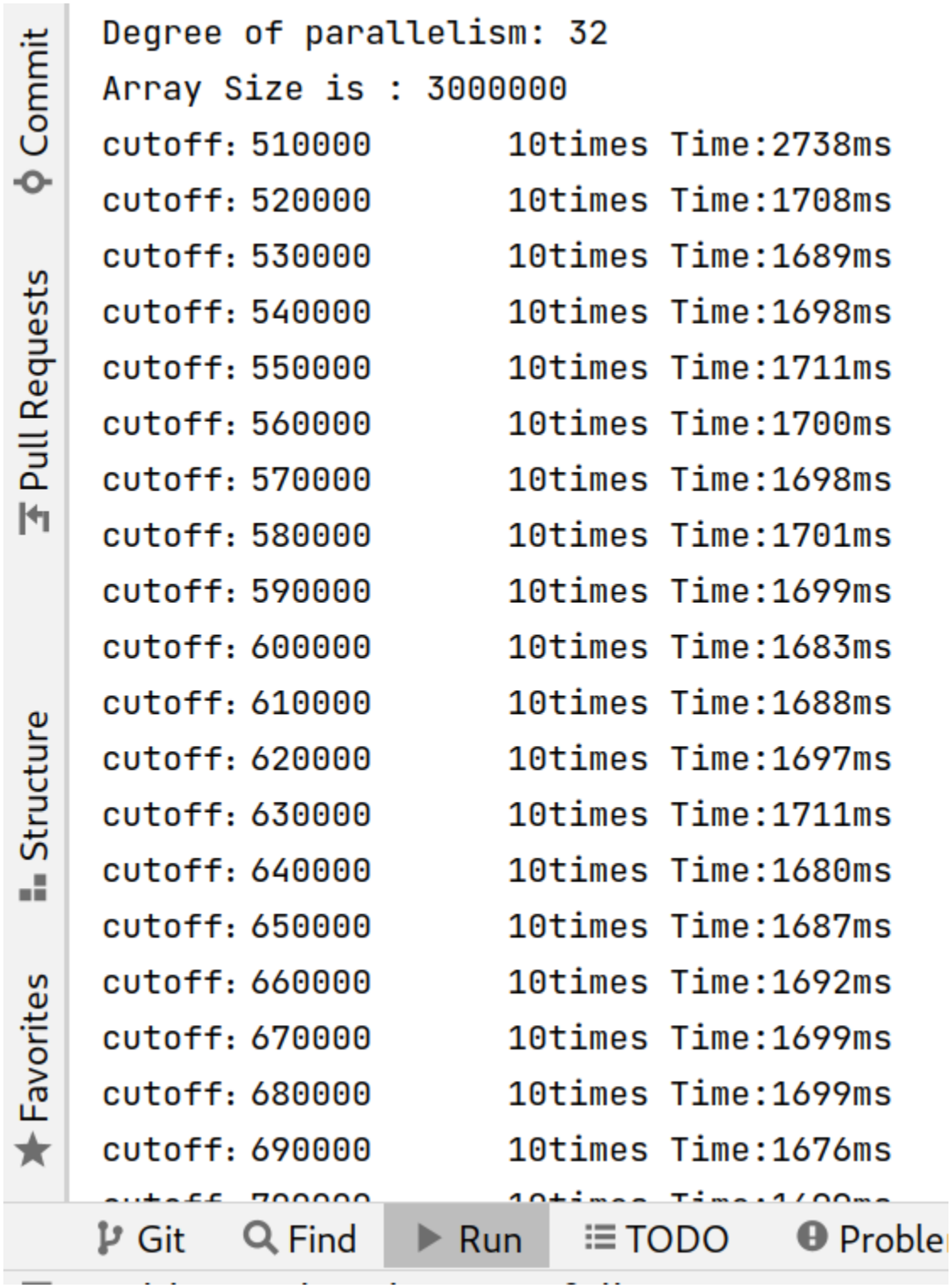
Graphical representation of evidence



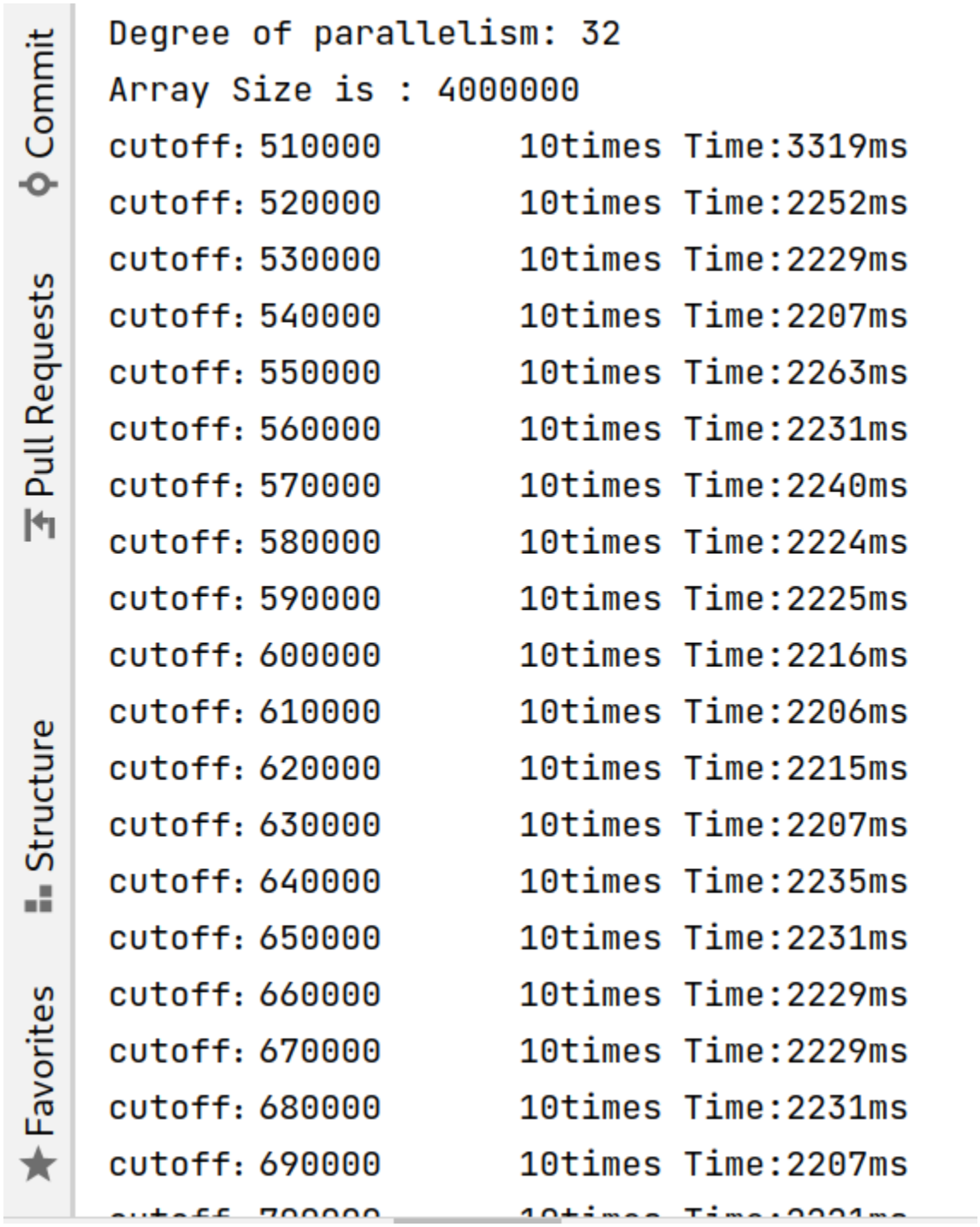
The above experiment was conducted using a cutoff range of 510000 – 920000, an array size of 2000000, and thread counts ranging from 2 to 32. According to the results of the above trials, the appropriate cutoff value is between 640000 and 820000. As a result, 640000 is the most efficient thread count 32.

Let's validate this by running a test with a thread count of 32 and a range of 640000 to 820000. for various array sizes by altering main.java according to array size

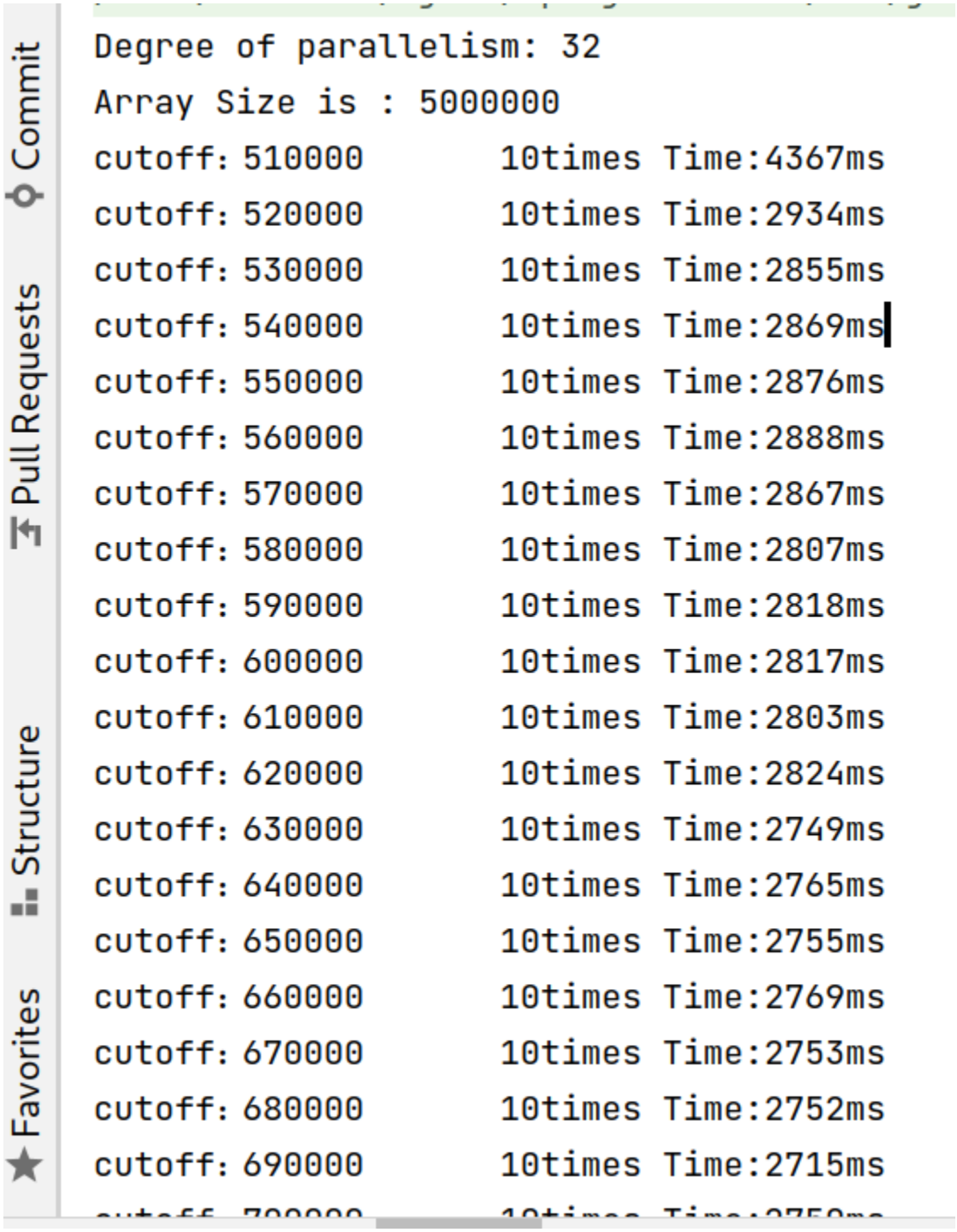
Available thread count : 32 ; Array size: 3000000 ; Sorted 10 times(average)



Available thread count : 32 ; Array size: 4000000 ; Sorted 10 times(average)



Available thread count : 32 ; Array size: 5000000 ; Sorted 10 times(average)



Conclusion

According to the results of the above studies, even though we increased the Array size from 2000000 to 5000000 by a factor of 1000000, it took longer, but when the cutoff value is 640000 and the thread count is 32, the algorithm works most efficiently.
Cut off Time : 640000 Thread count : 32