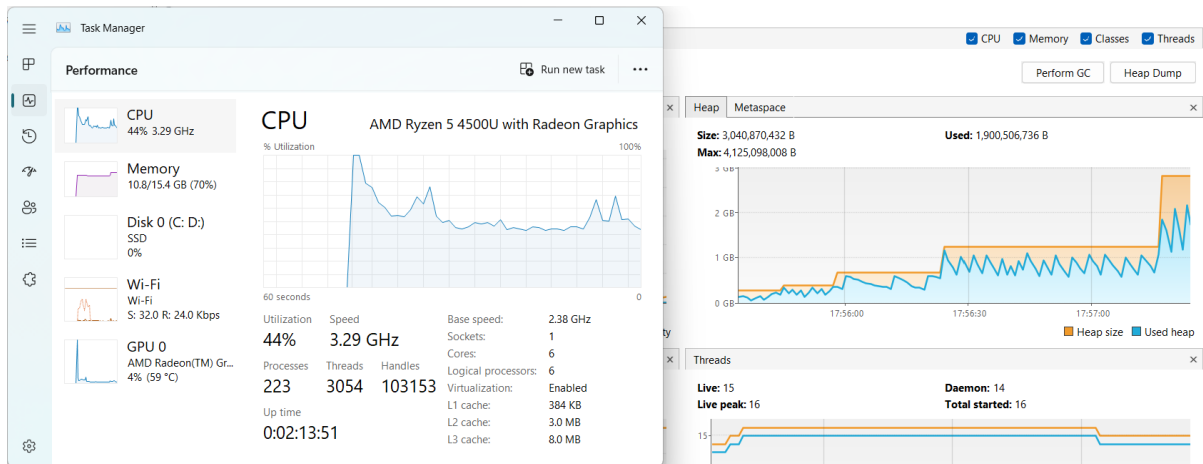


Lab13



ในส่วนของ RAM นั้นการที่มันเพิ่มขึ้นมาอย่างที่เราเห็นในรูปถือว่าเป็นไปตามปกติของสิ่งที่ควรจะเกิดขึ้น เพราะว่ามันเป็นการทำงานแบบ whileloop แบบไร้ที่สิ้นสุด ทำให้มันกิน RAM อย่างต่อเนื่องอย่างที่เราเห็น

CPU samples			Thread CPU time		
Results: [Icons]			Collected data: Snapshot		
Name			Total Time	Total Time (CPU)	Thread Dump
main					
Main.main ()			126,992 ms (100%)	126,992 ms (100%)	
java.io.BufferedReader.readLine ()			126,992 ms (100%)	126,992 ms (100%)	
java.nio.file.Files.newBufferedReader ()			38,461 ms (30.3%)	38,461 ms (30.3%)	
java.io.BufferedReader.close ()			37,121 ms (29.2%)	37,121 ms (29.2%)	
Parser_is_can.parse ()			26,479 ms (20.9%)	26,479 ms (20.9%)	
Parser_is_can.parseE ()			13,173 ms (10.4%)	13,173 ms (10.4%)	
Parser_is_can.parseT ()			13,173 ms (10.4%)	13,173 ms (10.4%)	
Parser_is_can.parseF ()			6,479 ms (5.1%)	6,479 ms (5.1%)	
Tokenizer_is_can.consume ()			4,976 ms (3.9%)	4,976 ms (3.9%)	
Self time			4,976 ms (3.9%)	4,976 ms (3.9%)	
Tokenizer_is_can.computeNext ()			3,976 ms (3.1%)	3,976 ms (3.1%)	
Self time			999 ms (0.8%)	999 ms (0.8%)	
Tokenizer_is_can.consume ()			0.0 ms (0%)	0.0 ms (0%)	
Self time			901 ms (0.7%)	901 ms (0.7%)	
Tokenizer_is_can.computeNext ()			800 ms (0.6%)	800 ms (0.6%)	
Self time			100 ms (0.1%)	100 ms (0.1%)	
Tokenizer_is_can.consume ()			601 ms (0.5%)	601 ms (0.5%)	
Self time			4,689 ms (3.7%)	4,689 ms (3.7%)	
Tokenizer_is_can.computeNext ()			3,089 ms (2.4%)	3,089 ms (2.4%)	
Self time			1,600 ms (1.3%)	1,600 ms (1.3%)	
Tokenizer_is_can.computeNext ()			2,004 ms (1.6%)	2,004 ms (1.6%)	
Self time			0.0 ms (0%)	0.0 ms (0%)	

จากการวิเคราะห์ด้วยข้อมูลที่มี ตรงส่วนข้อมูลของ CPU ทำให้เราได้ทราบว่าการทำงานของ method จำพวก newBufferedReader(), readLine() และ close() ทำให้ CPU ทำงานถึง 29.2%, 30.3% และ 20.9% ตามลำดับ ซึ่งเป็นการใช้งาน CPU อย่างสิ้นเปลืองทรัพยากรมากพอสมควร ทำให้คิดว่าเรานั้นอาจเปลี่ยนจากการอ่านเนื้อหาจาก input.txt มาเป็น อ่านค่าทดลองโดยตรงเลยจะเป็นสิ่งที่ดีกว่าอย่างเห็นได้ชัด

Heap histogram		Per thread allocations			
Results:		Collected data: Snapshot		Perform GC Heap Dump	
Name		Live Bytes		Live Objects	
byte[]		1,621,189,672 B	(69.6 %)	6,284,692	(3.8 %)
char[]		462,986,984 B	(19.9 %)	28,298	(0.2 %)
java.lang.StringBuilder		75,197,616 B	(3.2 %)	3,133,234	(18.9 %)
java.lang.String		74,930,088 B	(3.2 %)	3,122,087	(18.9 %)
BinaryArithExpri		31,840,056 B	(1.4 %)	1,326,669	(8 %)
IntLit		24,839,760 B	(1.1 %)	1,552,485	(9.4 %)
Tokenizer_is_can		5,419,584 B	(0.2 %)	225,816	(1.4 %)
java.nio.HeapCharBuffer		4,742,192 B	(0.2 %)	84,682	(0.5 %)
Parser_is_can		3,613,056 B	(0.2 %)	225,816	(1.4 %)
int[]		3,567,016 B	(0.2 %)	1,674	(0 %)
java.nio.HeapByteBuffer		3,161,704 B	(0.1 %)	56,459	(0.3 %)
sun.nio.ch. FileChannelImpl		2,032,344 B	(0.1 %)	28,227	(0.2 %)
java.io. FileCleanable		1,581,328 B	(0.1 %)	28,238	(0.2 %)
jdk.internal.ref. CleanerImpl\$PhantomCleanableRef		1,356,336 B	(0.1 %)	28,257	(0.2 %)
java.io. BufferedReader		1,354,944 B	(0.1 %)	28,228	(0.2 %)
sun.nio.cs. StreamDecoder		1,354,944 B	(0.1 %)	28,228	(0.2 %)
java.io. FileDescriptor		1,129,840 B	(0 %)	28,246	(0.2 %)
sun.nio.cs. UTF_8\$Decoder		1,129,080 B	(0 %)	28,227	(0.2 %)
java.lang.Object		908,272 B	(0 %)	56,767	(0.3 %)
long[]		906,880 B	(0 %)	28,286	(0.2 %)
sun.nio.ch. ChannelInputStream		903,264 B	(0 %)	28,227	(0.2 %)
sun.nio.fs. WindowsChannelFactory\$Flag ;		903,232 B	(0 %)	28,226	(0.2 %)
java.io. InputStreamReader		677,472 B	(0 %)	28,228	(0.2 %)

ส่วนของ Heap นั้น การที่ byte[] กินความจำของ Heap เยอะที่สุดถือว่าปกติ เพราะว่า byte[] นั้นเกี่ยวข้องกับ String ที่เราใช้งานมากที่สุดถือว่าในส่วนของ Heap นั้นไม่มีอะไรที่ผิดปกติ