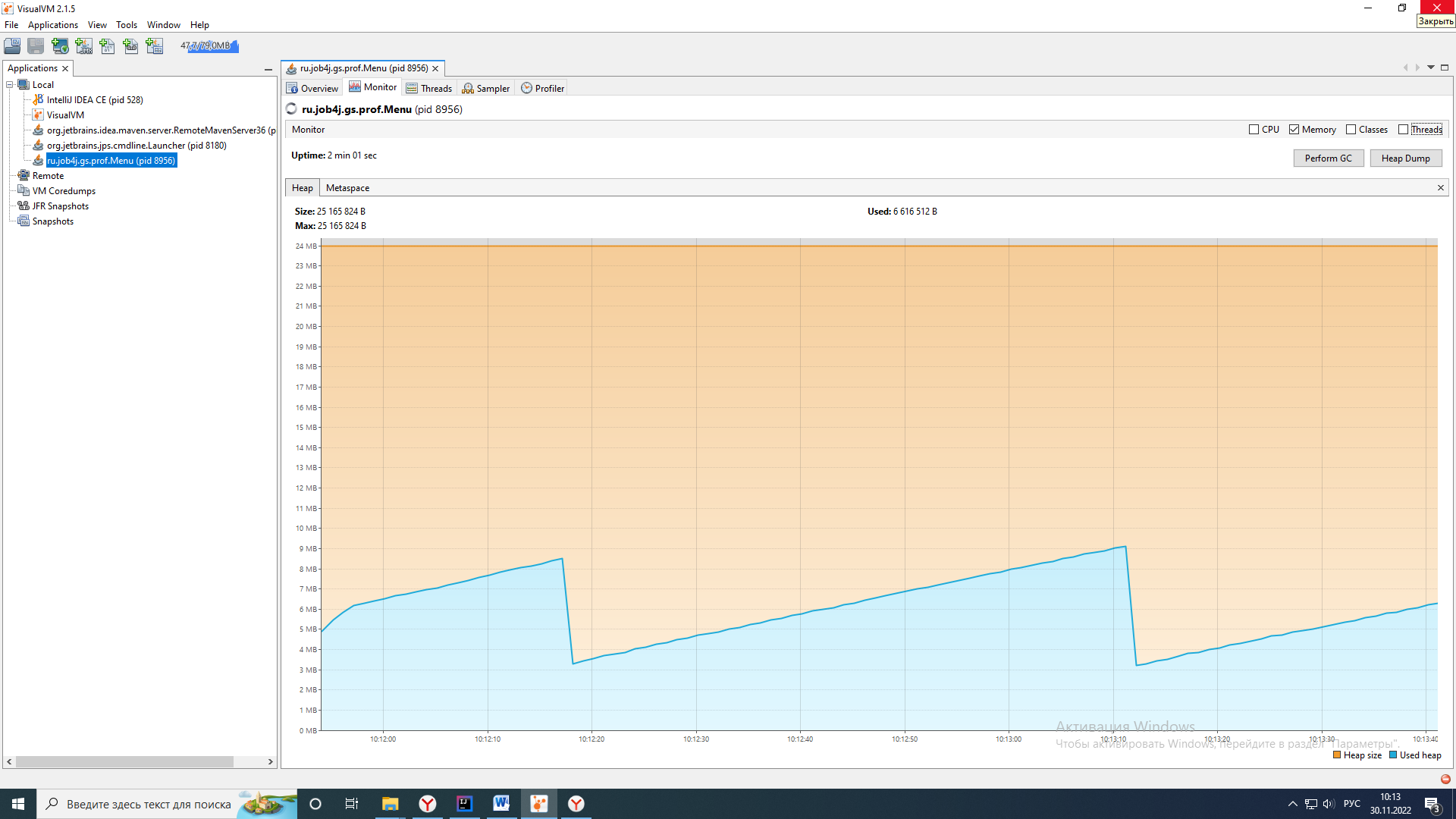
Запускаем программу. При запуске в хип поступило почти 9 мегабайт объектов. Затем произошла малая сборка мусора. Дальше можем наблюдать поднятие до 9 и снова сборку:

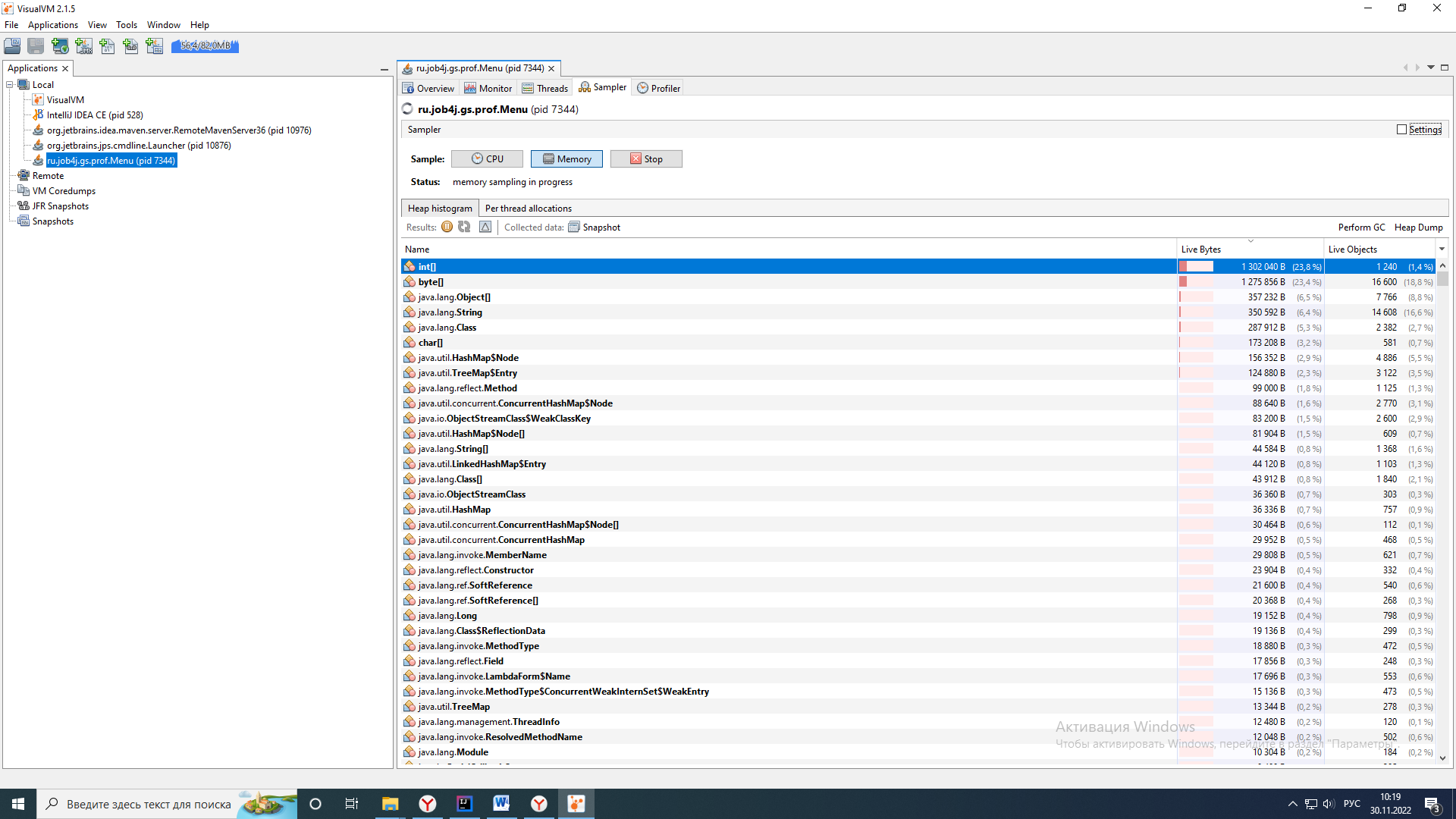


Открываем вкладку **Sampler**, отслеживаем количество объектов каждого типа:



Далее создаем массив на 250000 элементов. Мы можем его заметить - int[] вырос на  размер массива. После чего наблюдаем малую и полную сборку мусора.

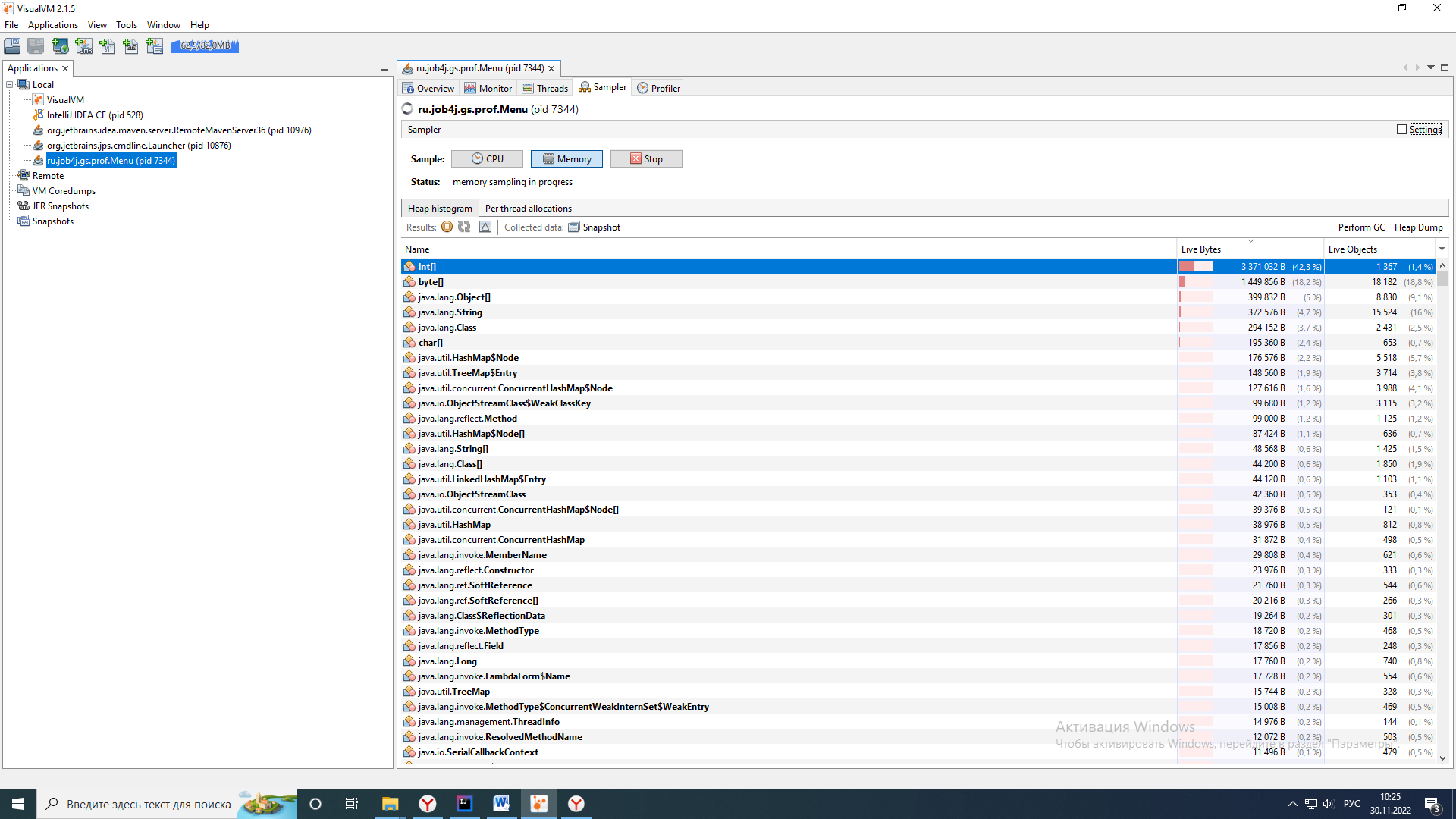


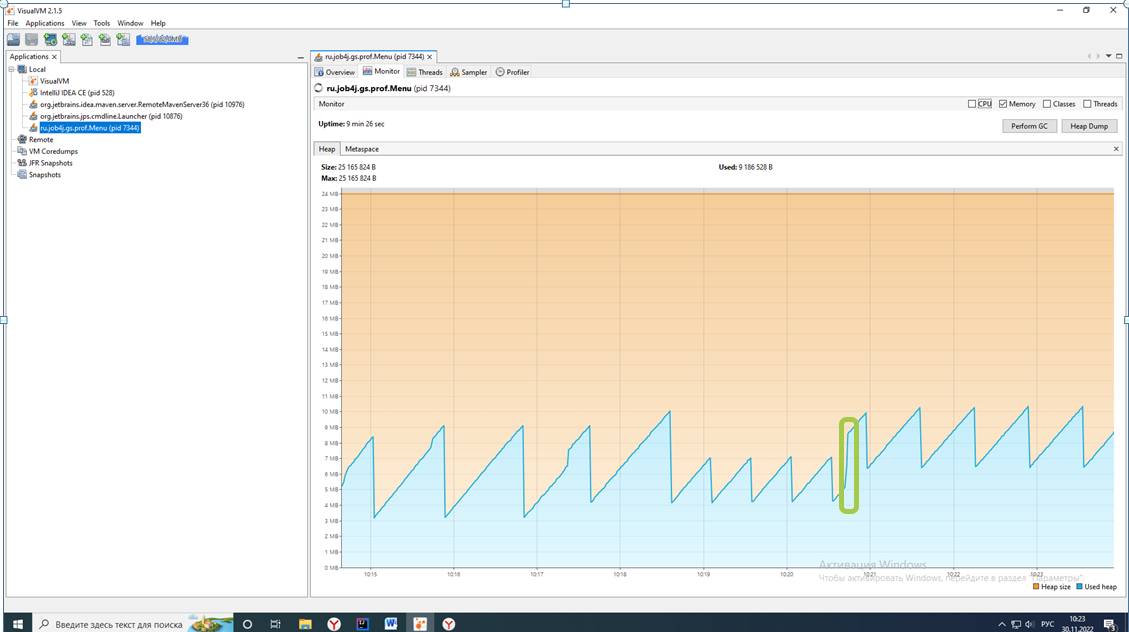


**1) Parallel GC**

**Сортировка слиянием.**

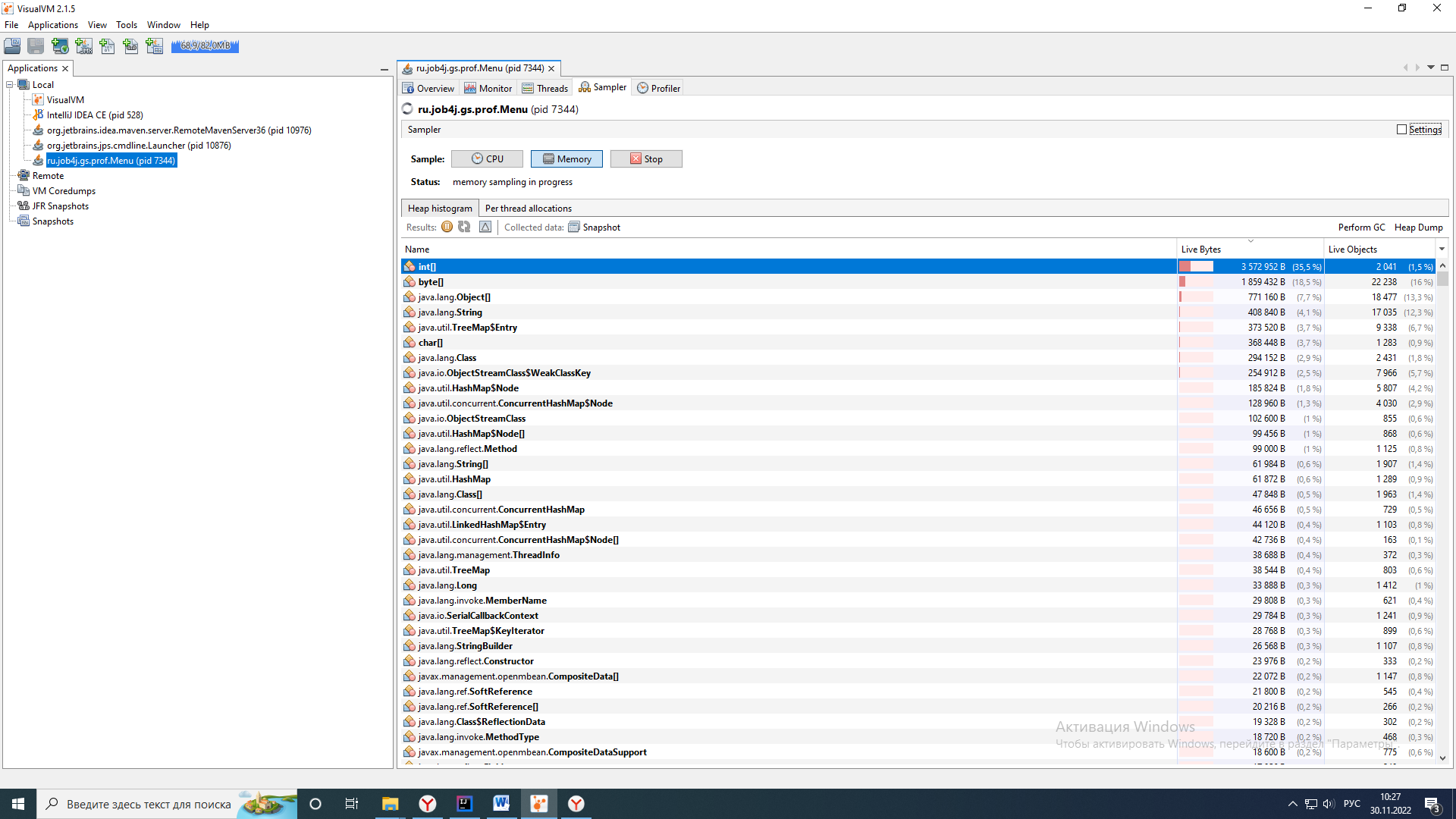
Делаем сортировку слиянием и видим, что объем int[] вырос примерно на 2 068 992‬+/- байт. Учитываем, что мы также создали клон массива(1000000+ байт). И 1 068 992‬+/- байт потребовалось при разбиении на массивы (что вдвое меньше по сравнению с Serial GC). Вся сортировка заняла 6 миллисекунд.



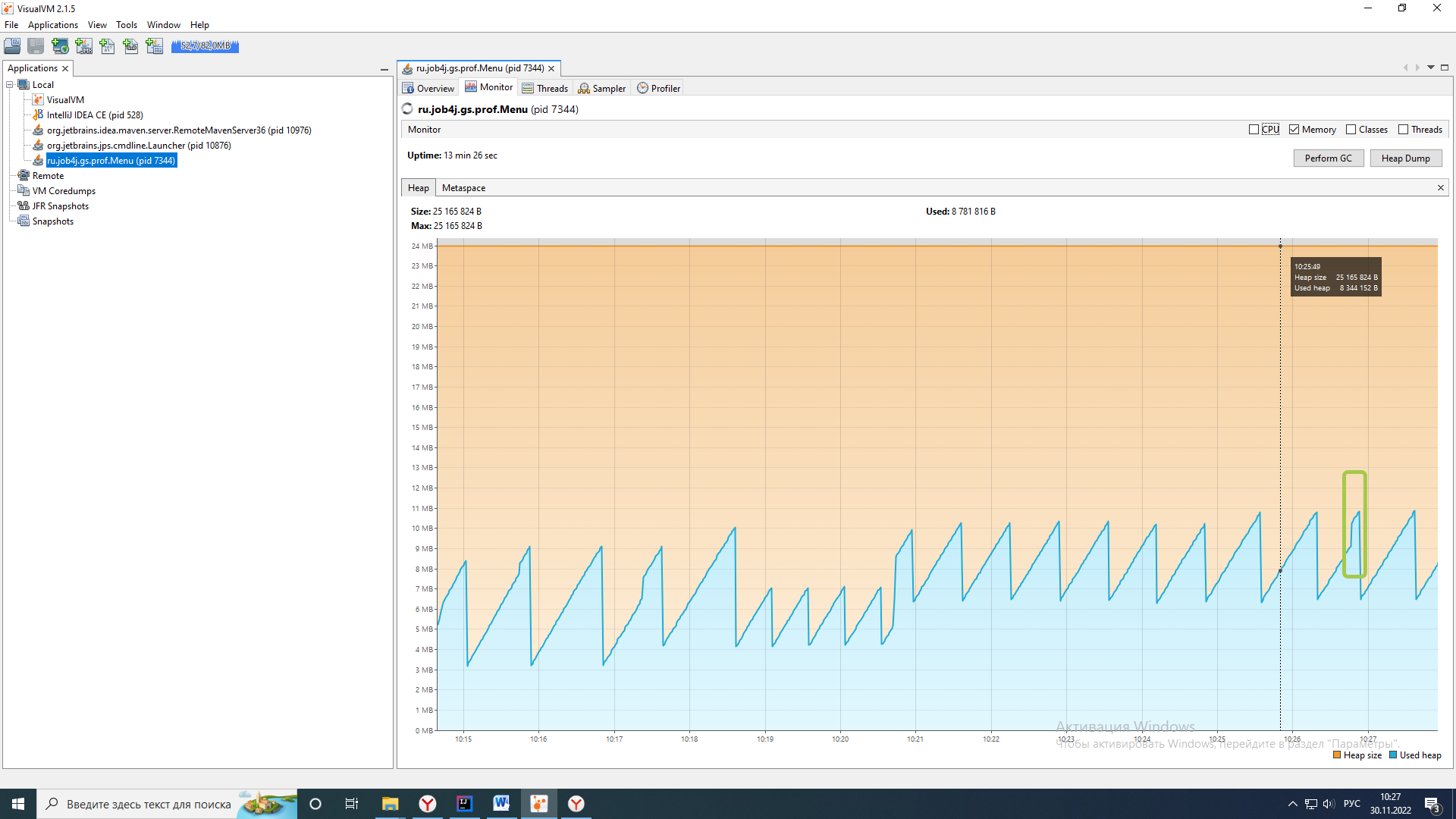


**Сортировка методом вставки.**

Теперь делаем сортировку методом вставками. Можем наблюдать, что хип вырос не существенно:

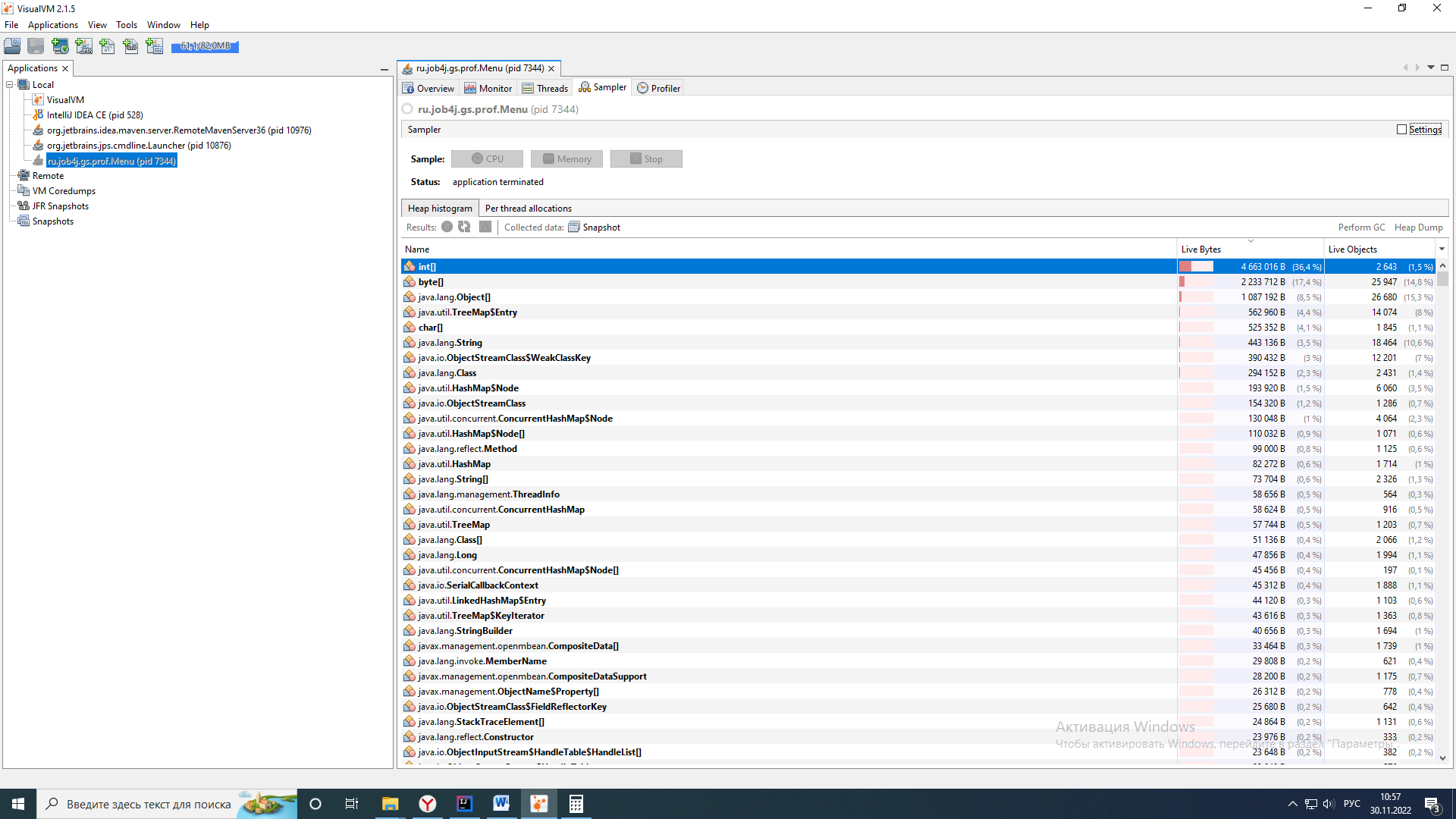


По времени процедура заняла 6 миллисекунд. Parallel  GC обходится малыми сборками:



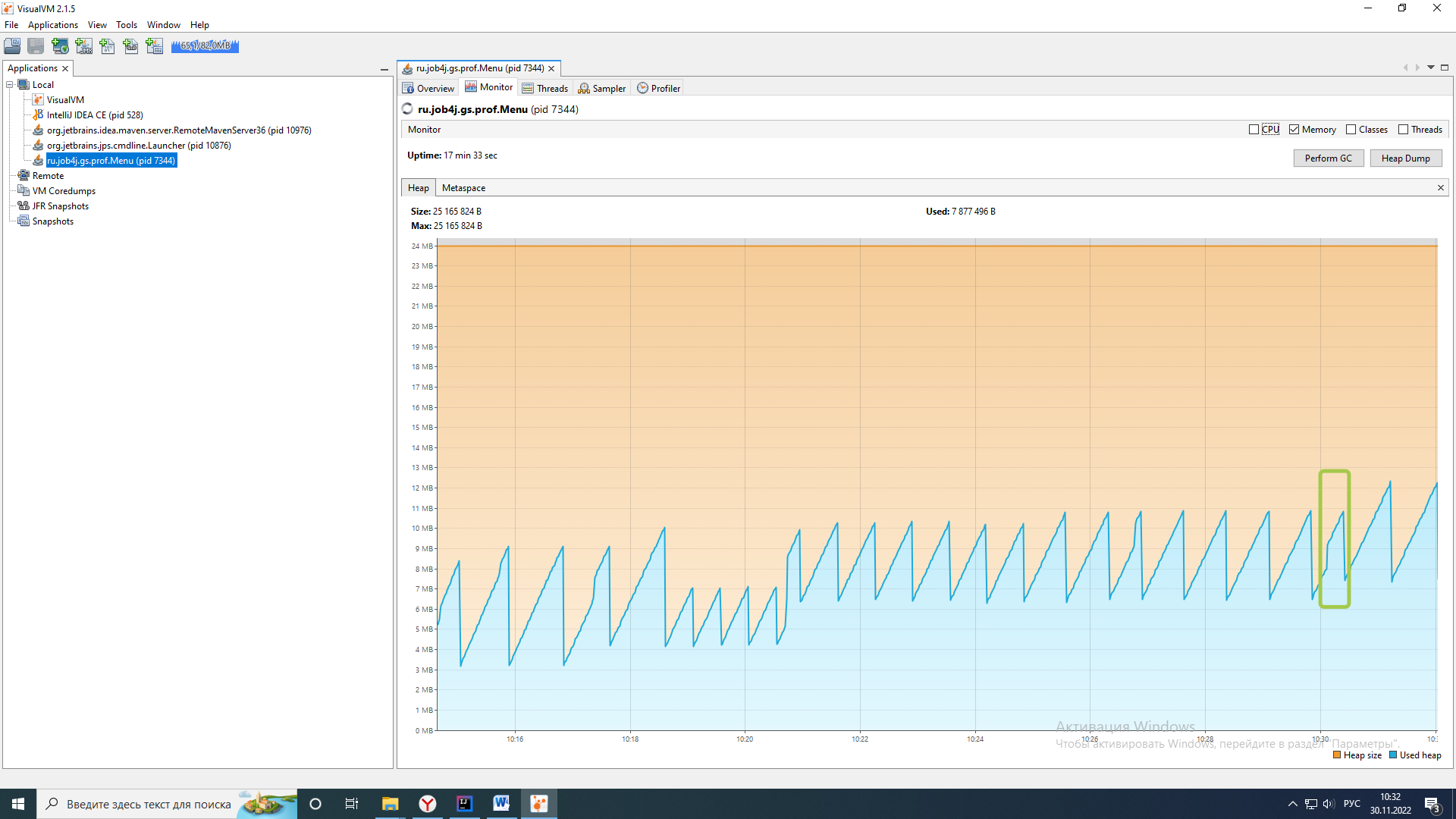
**Сортировка пузырьком.**

В хип добавился клон массива:

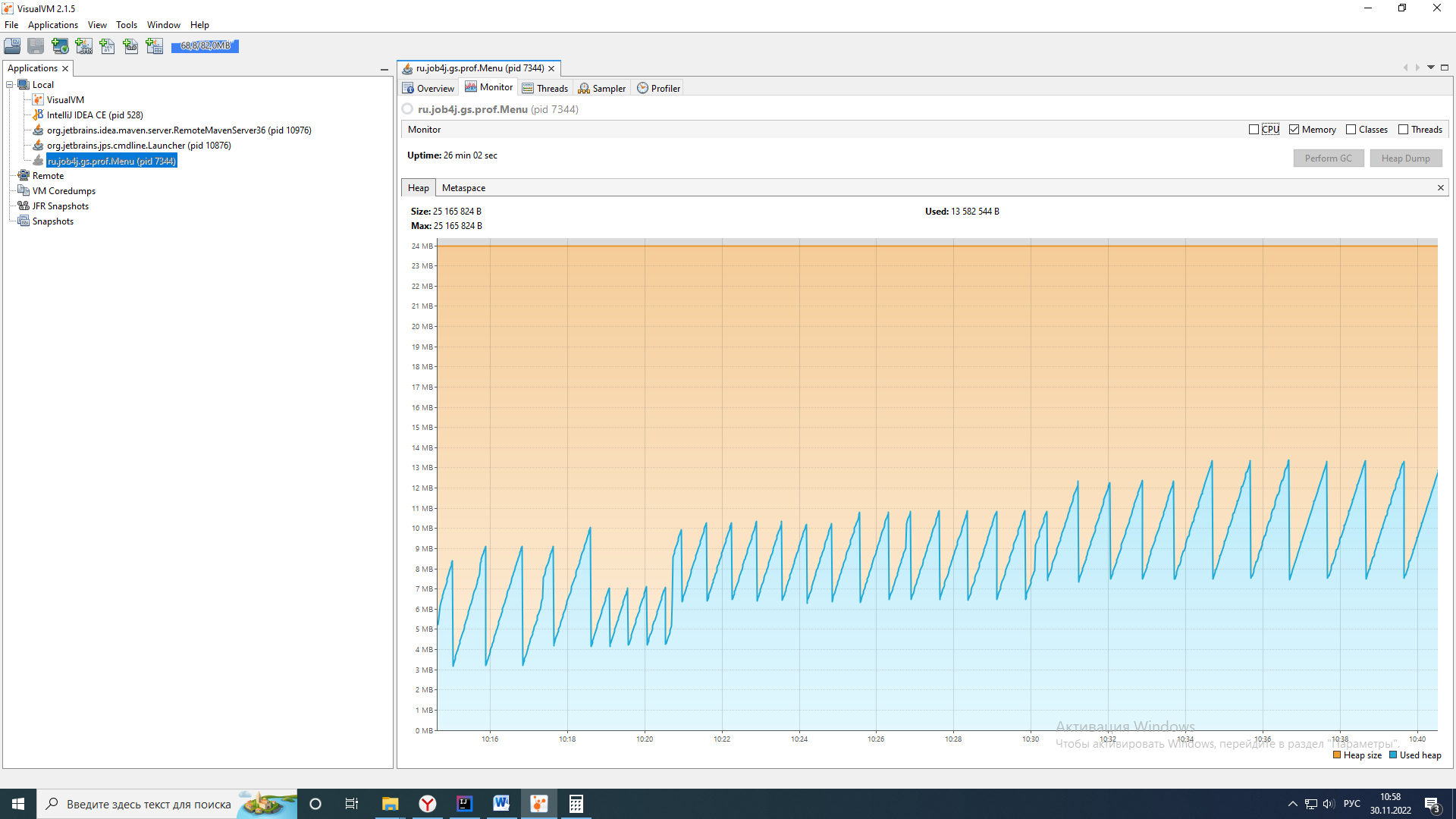


Parallel GC обходится малыми сборками.

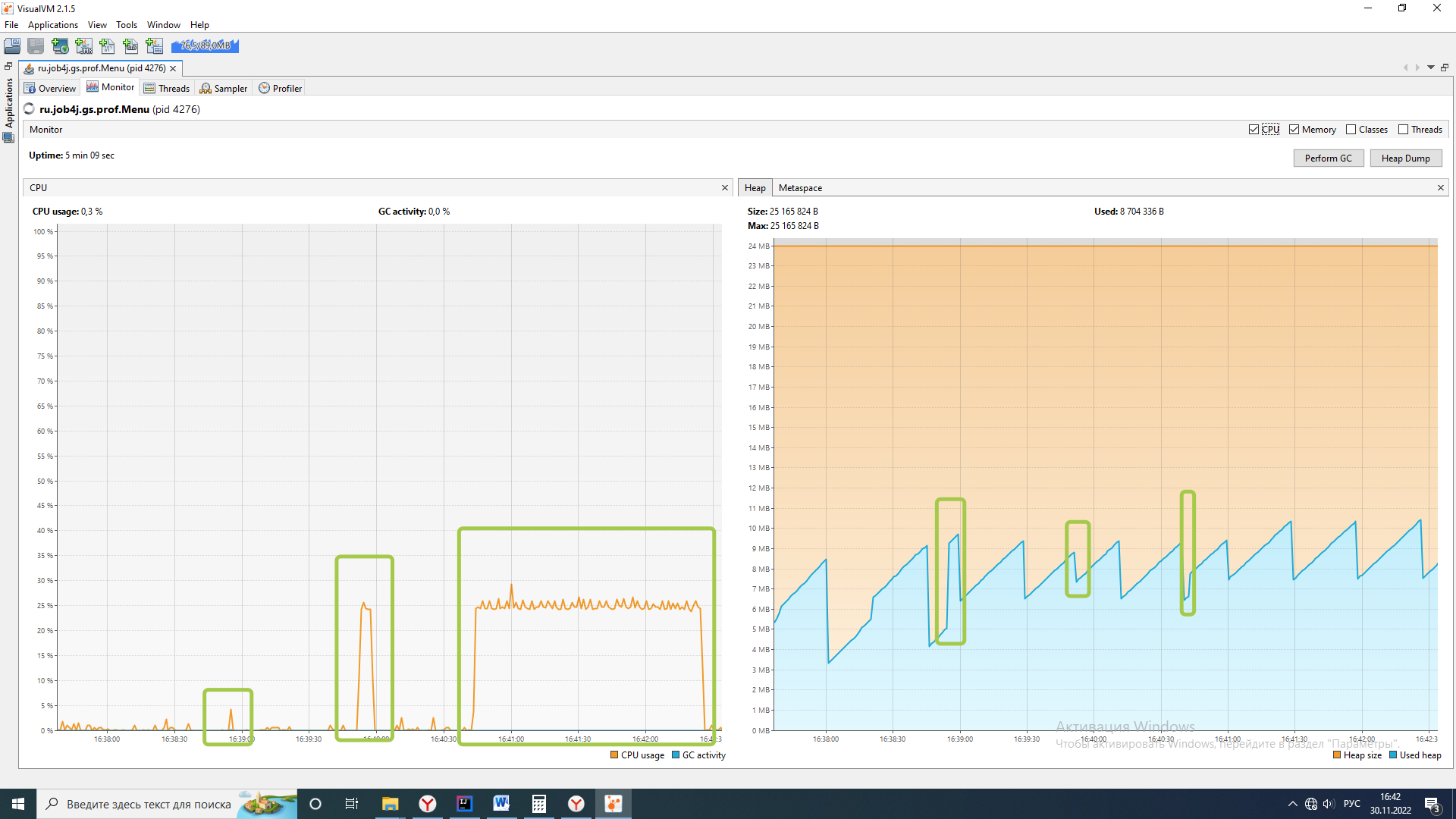
Сортировка пузырьком заняла 42 миллисекунды.



По полной диаграмме работы Parallel GC наблюдается стабильная работа .

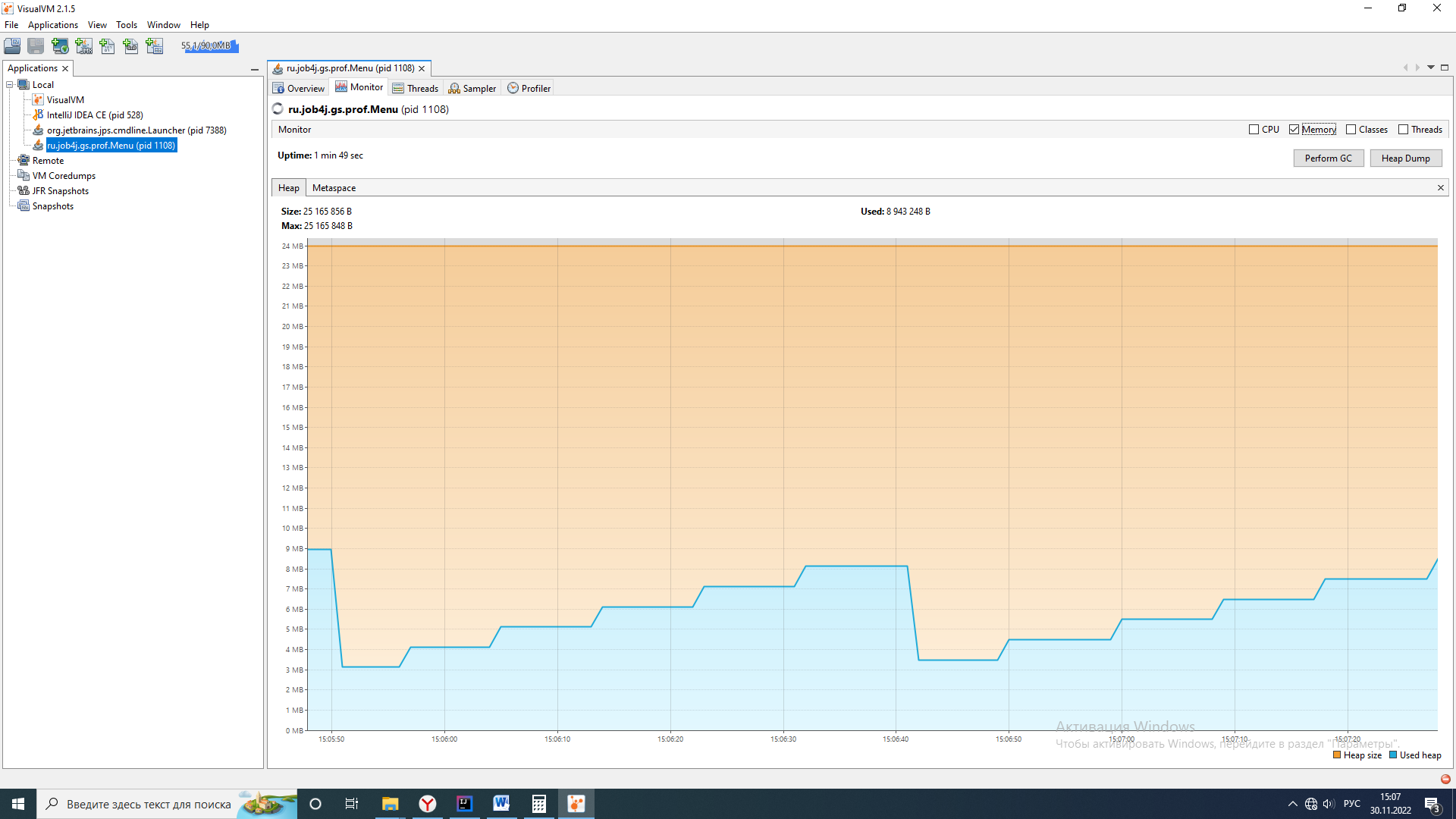


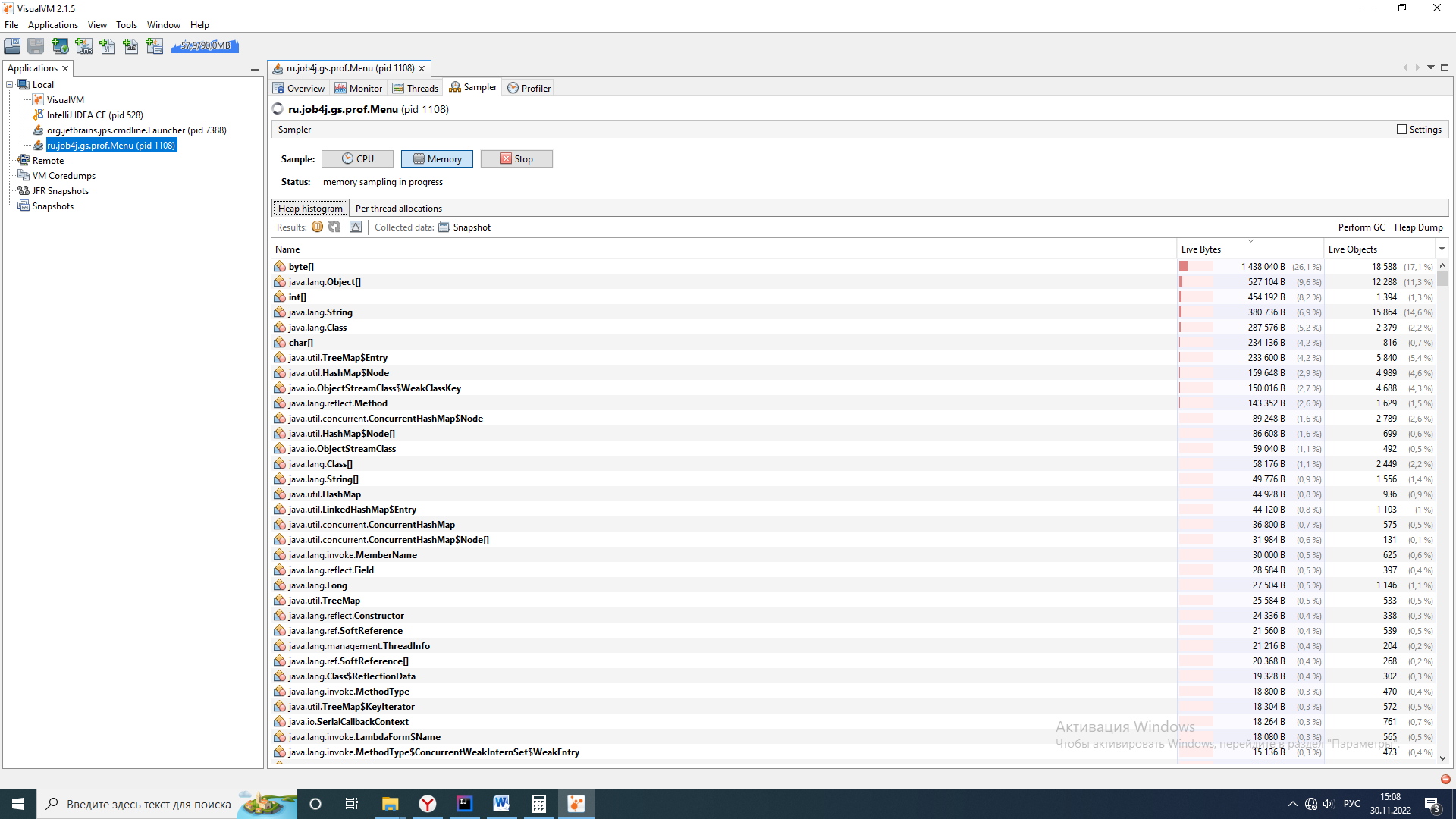
Загрузка ЦПУ. Сортировка пузырьком потребовала больше всего ресурсов процессора:



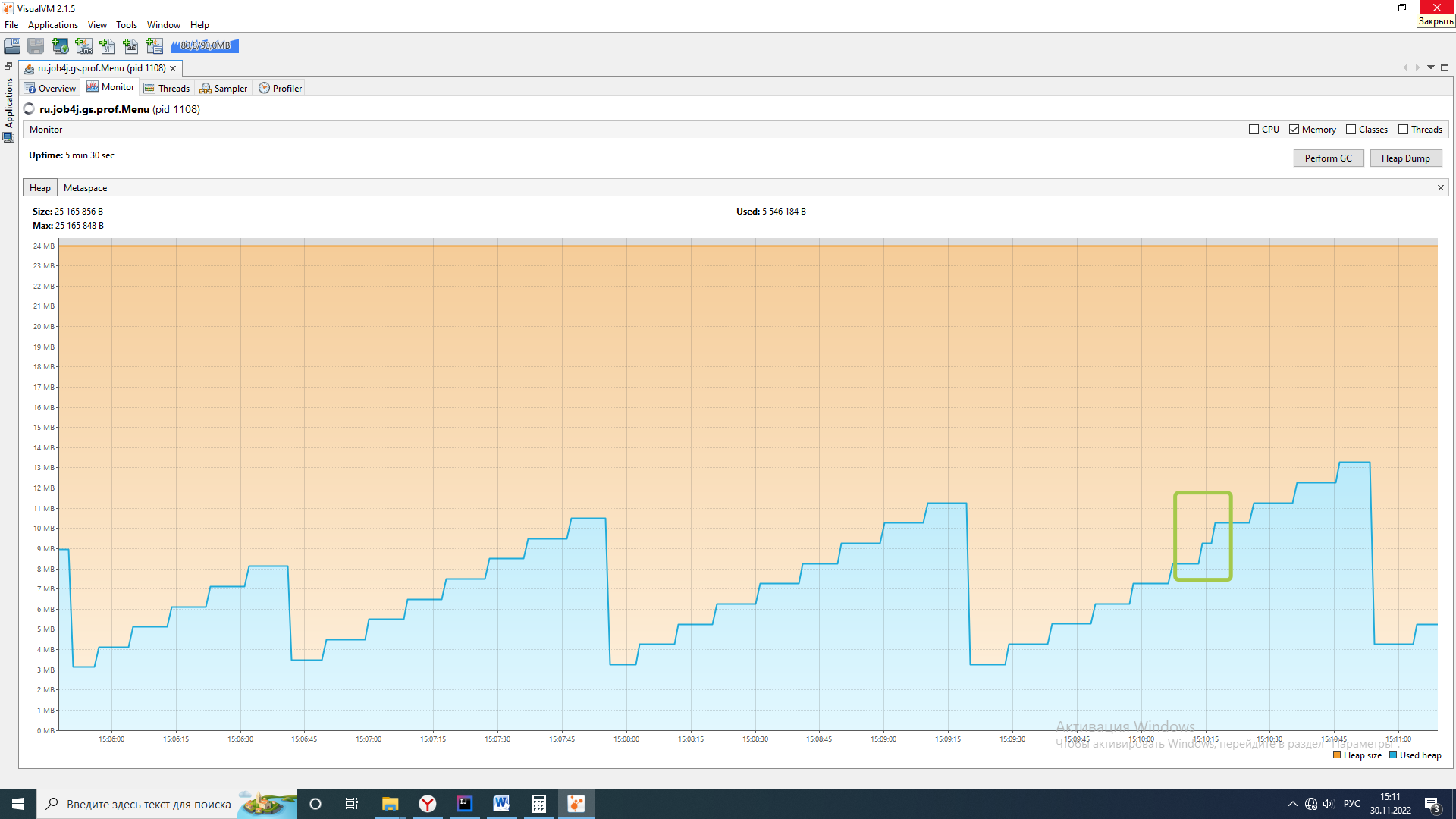
1. **Сборщик G1 GC**

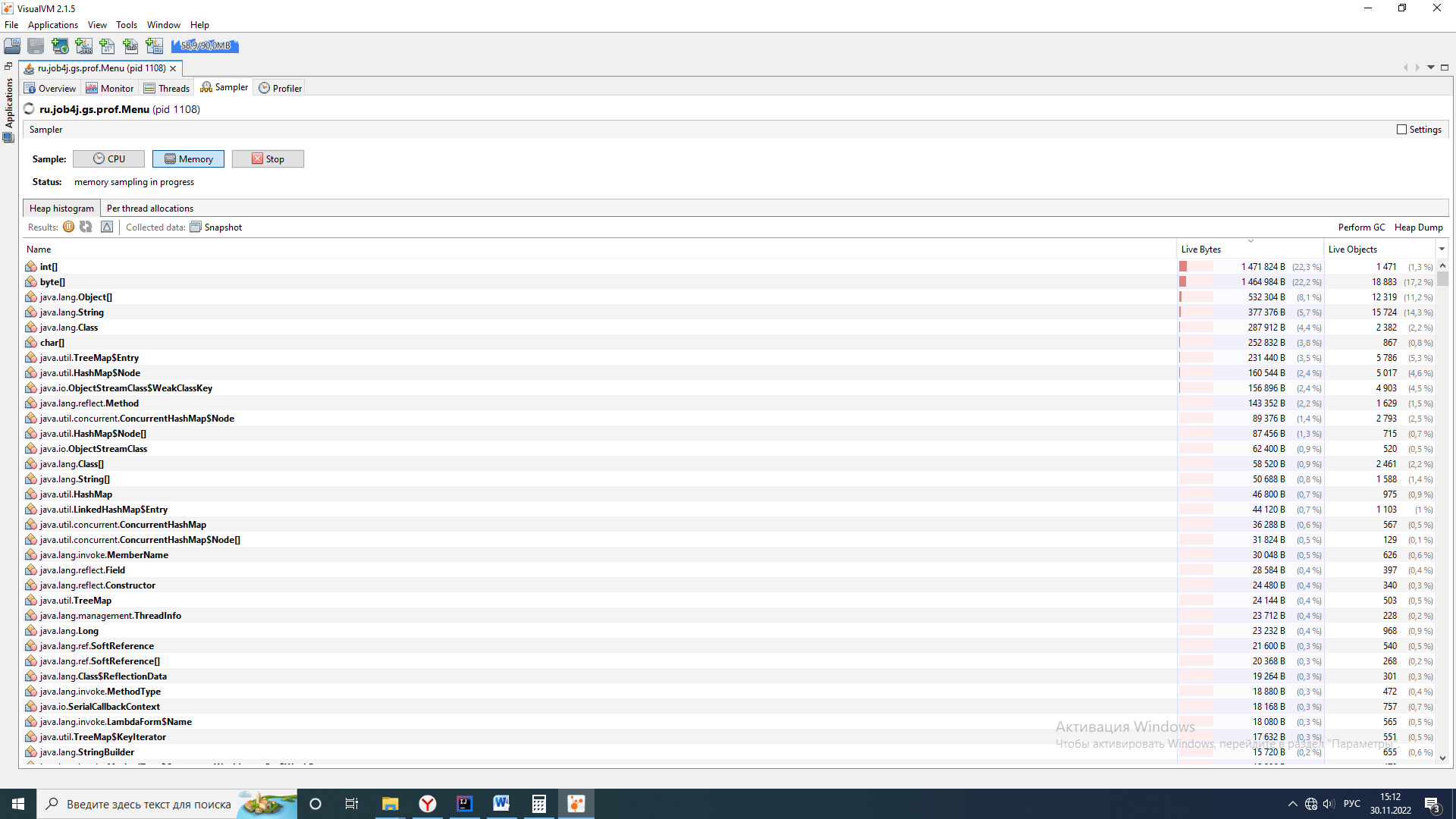
Запускаем программу. При запуске в хип поступило почти 9 мегабайт объектов. Затем произошла малая сборка мусора. Дальше можем наблюдать поднятие до 8 и снова сборку:



Открываем вкладку **Sampler**, отслеживаем количество объектов каждого типа: 

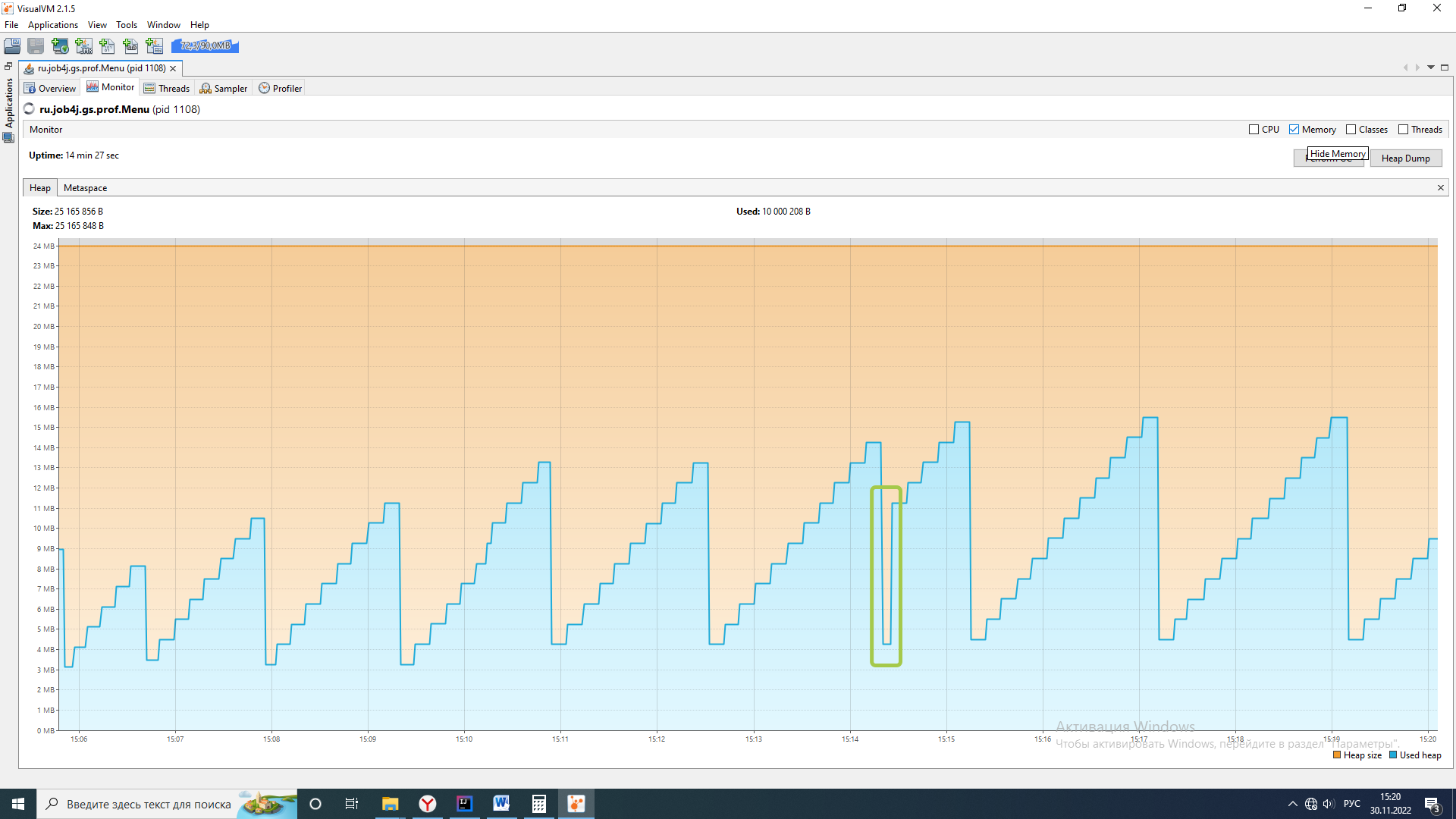
Создаем массив на 250000 элементов. Мы можем его заметить - int[] вырос на  размер массива мусора.

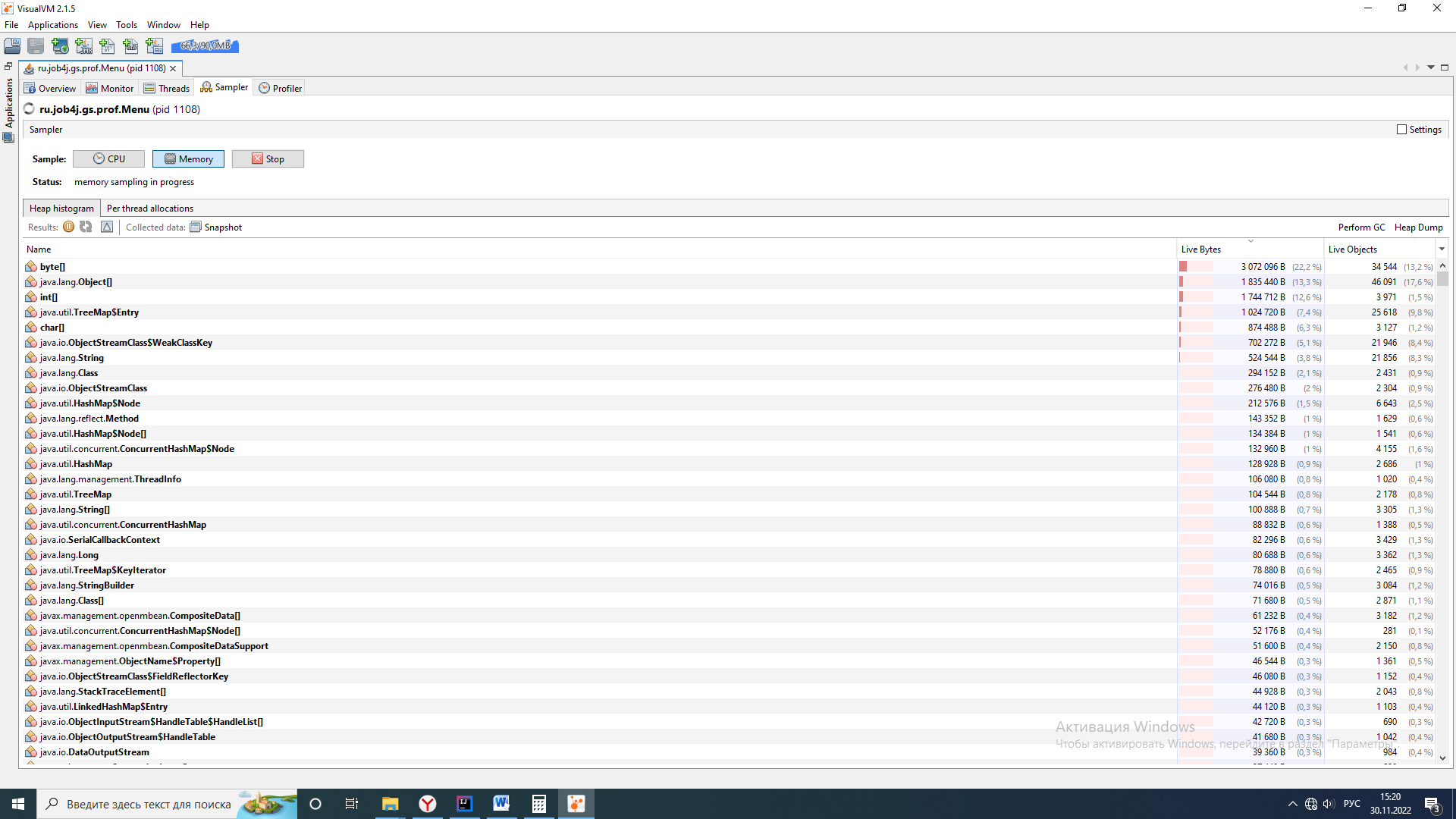




**Сортировка слиянием.**

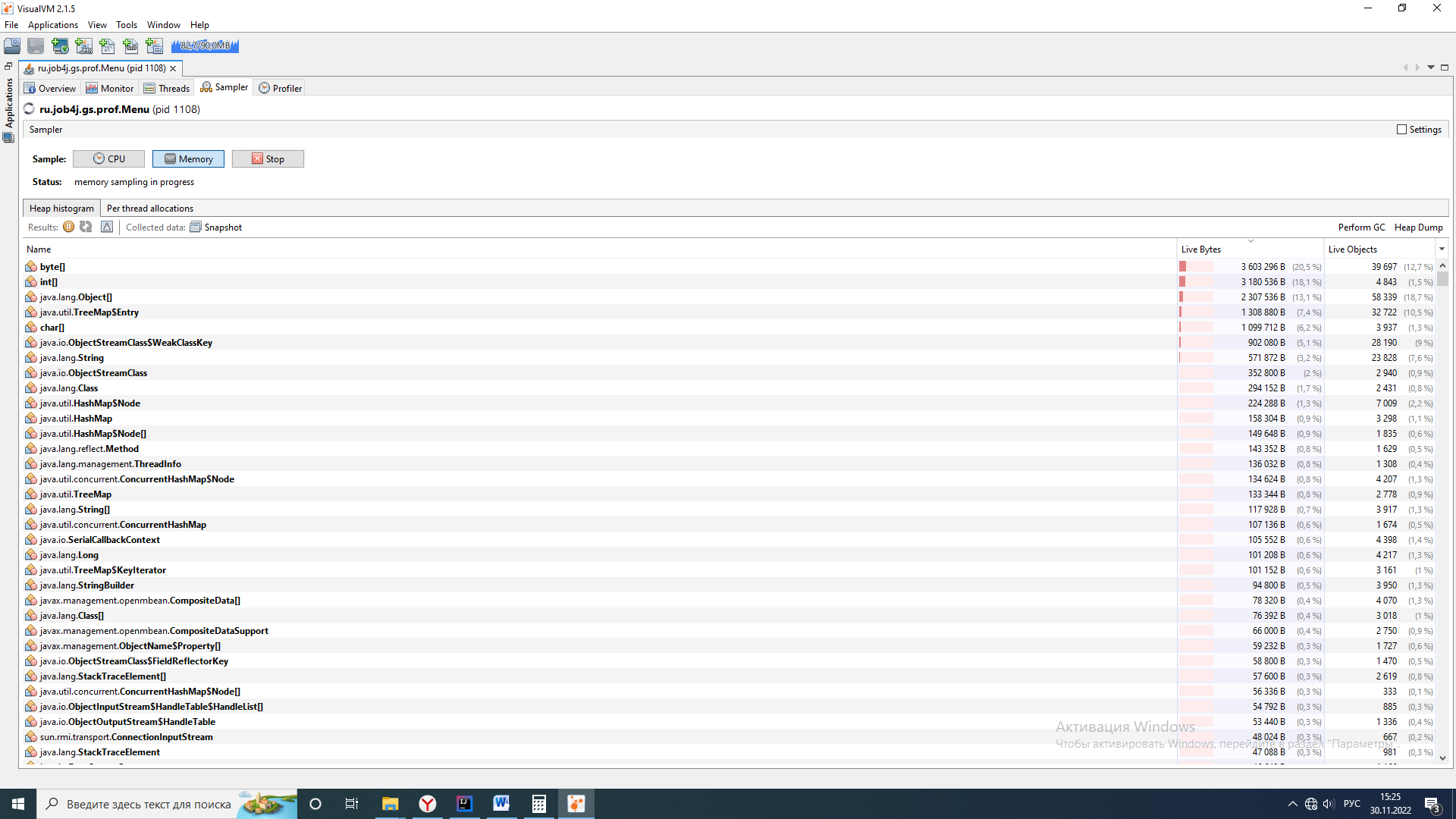
Делаем сортировку слиянием и видим, что объем int[] вырос незначительно. Вся сортировка заняла 6 миллисекунд.

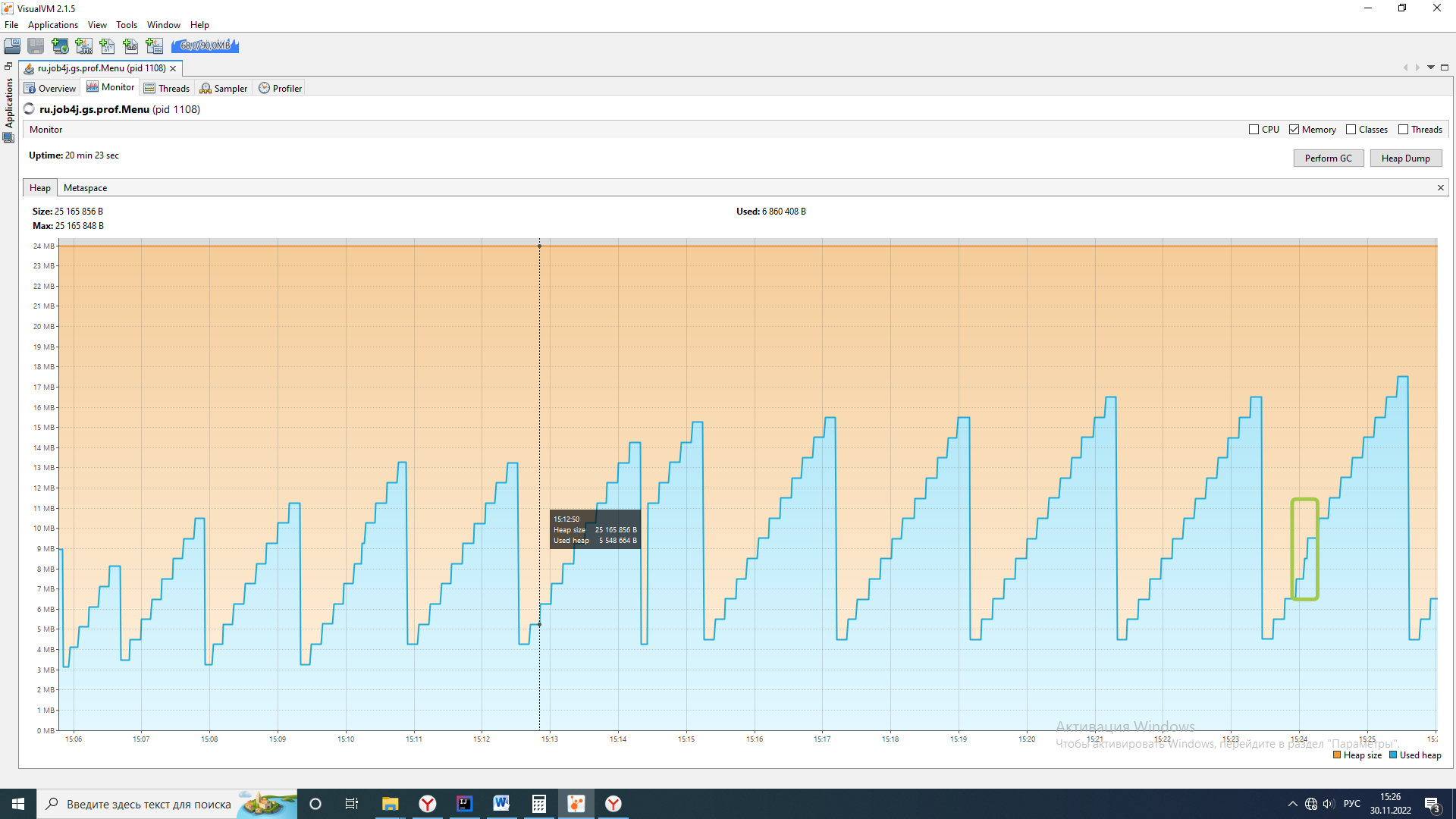




**Сортировка методом вставки.**

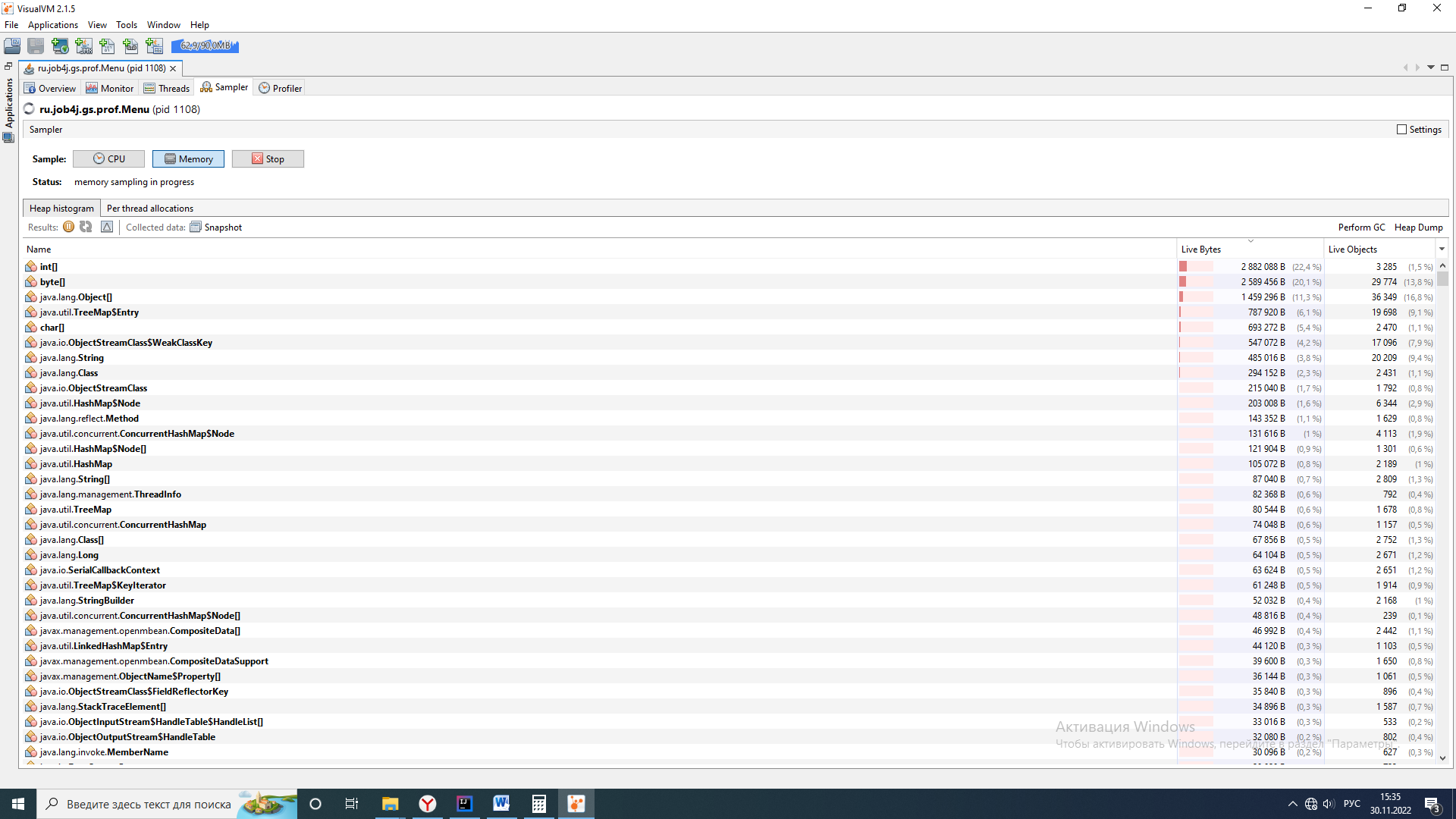
Теперь делаем сортировку методом вставками. Можем наблюдать, что хип вырос на 1 435 824‬ В.

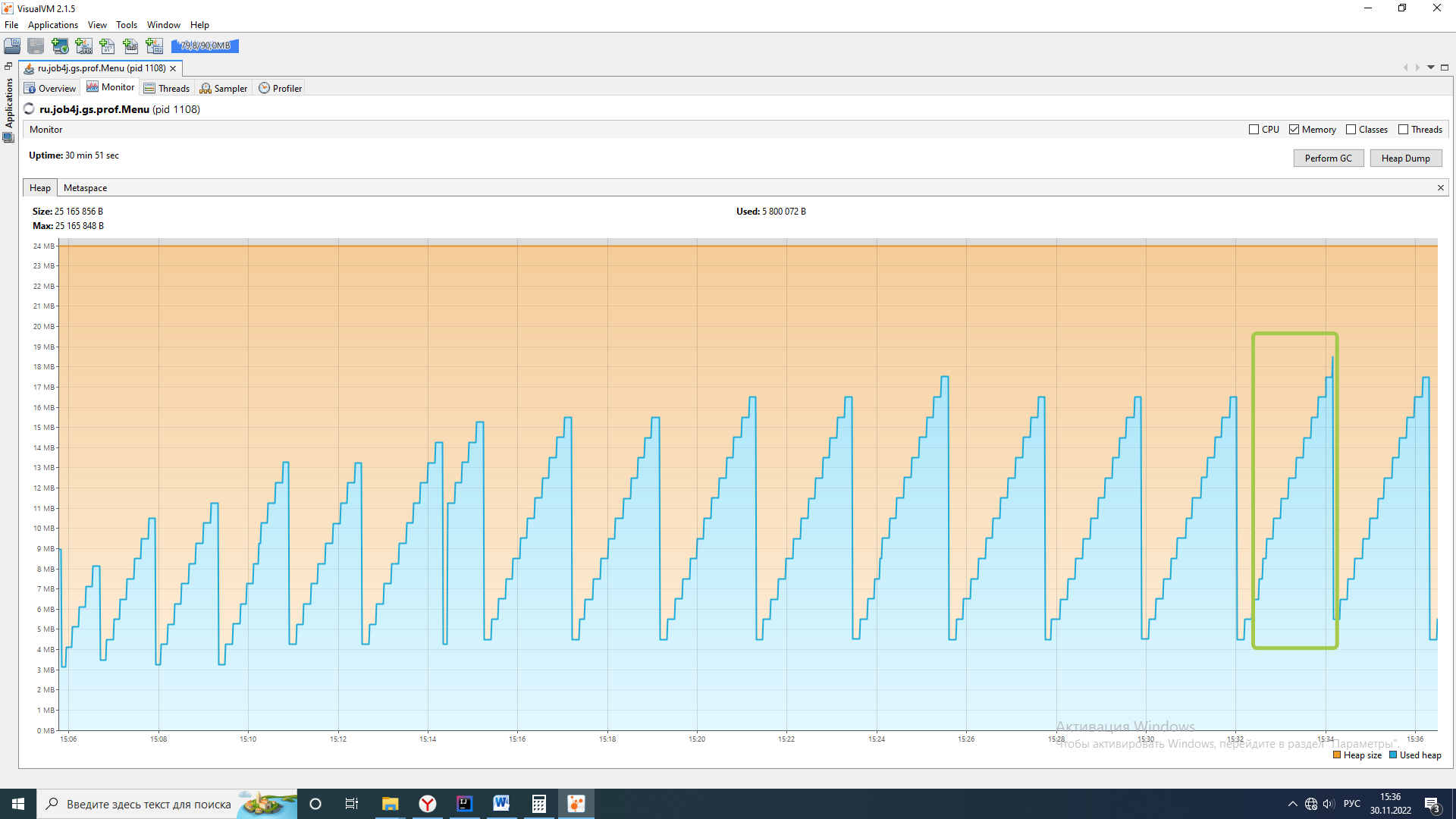




**Сортировка пузырьком.**

Начинаем сортировку пузырьком. В хип добавился клон массива:



Сортировка пузырьком заняла 1 минуту 39 секунд. 

Полная диаграмма работы G1 GC

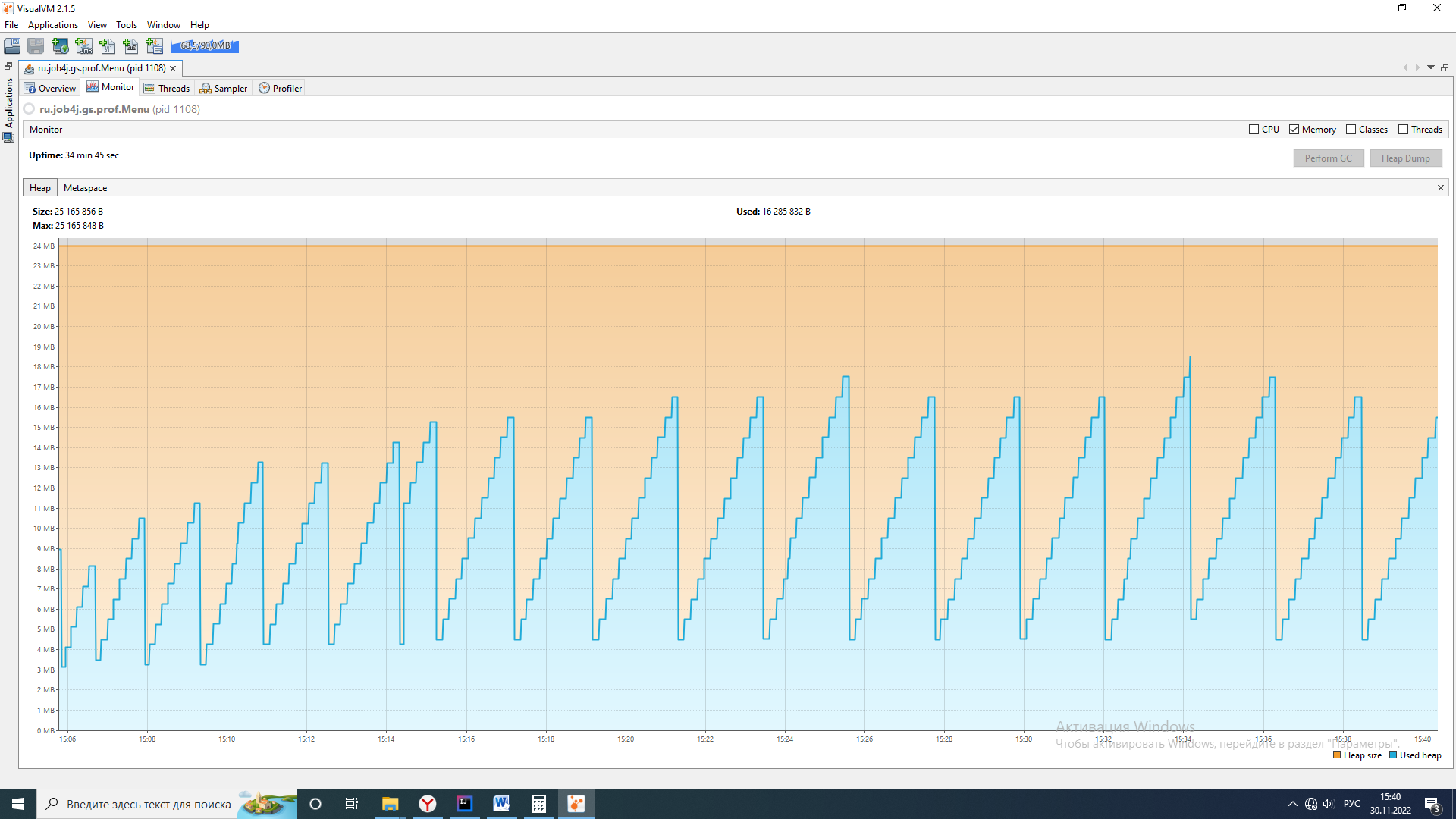
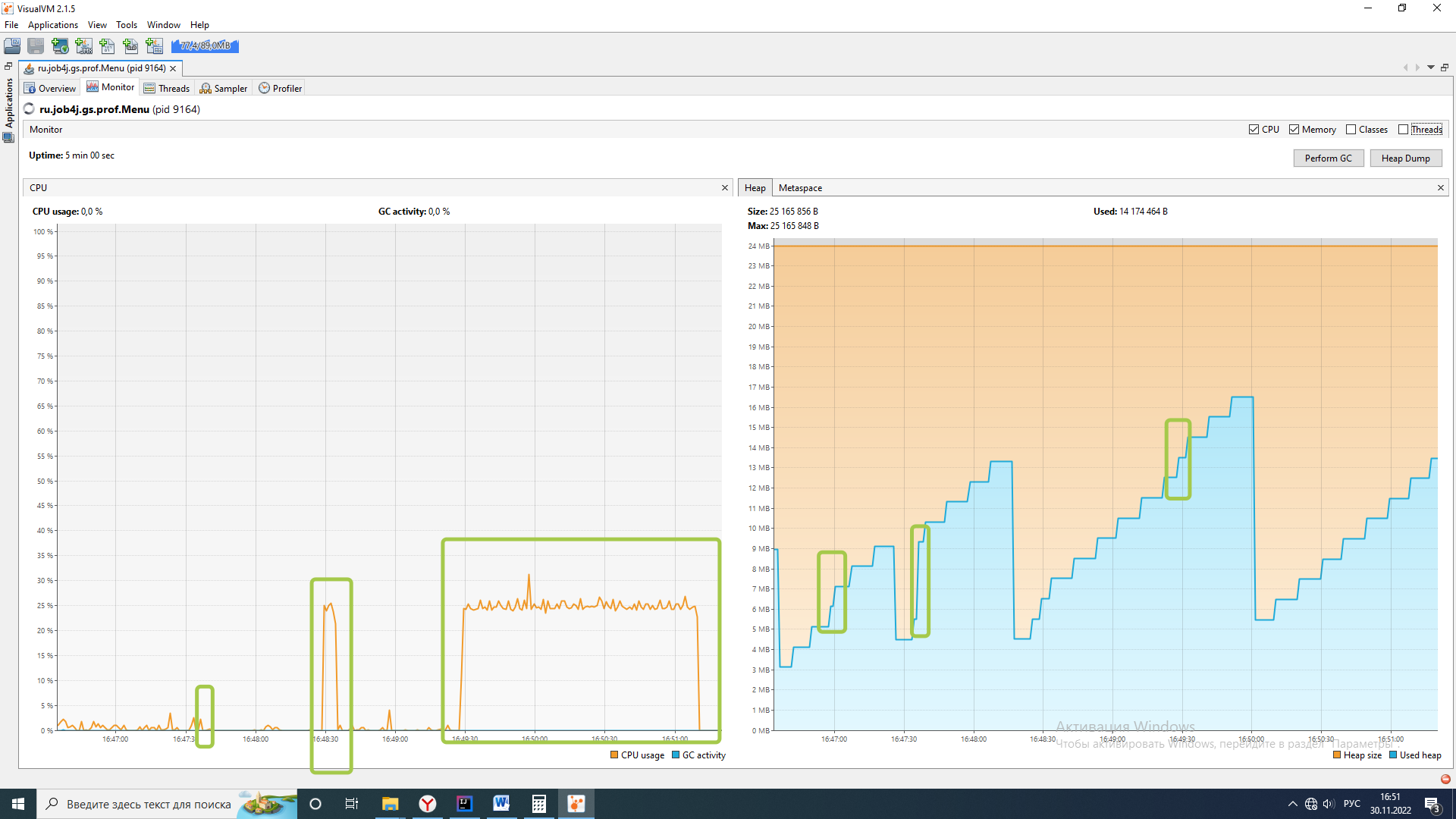


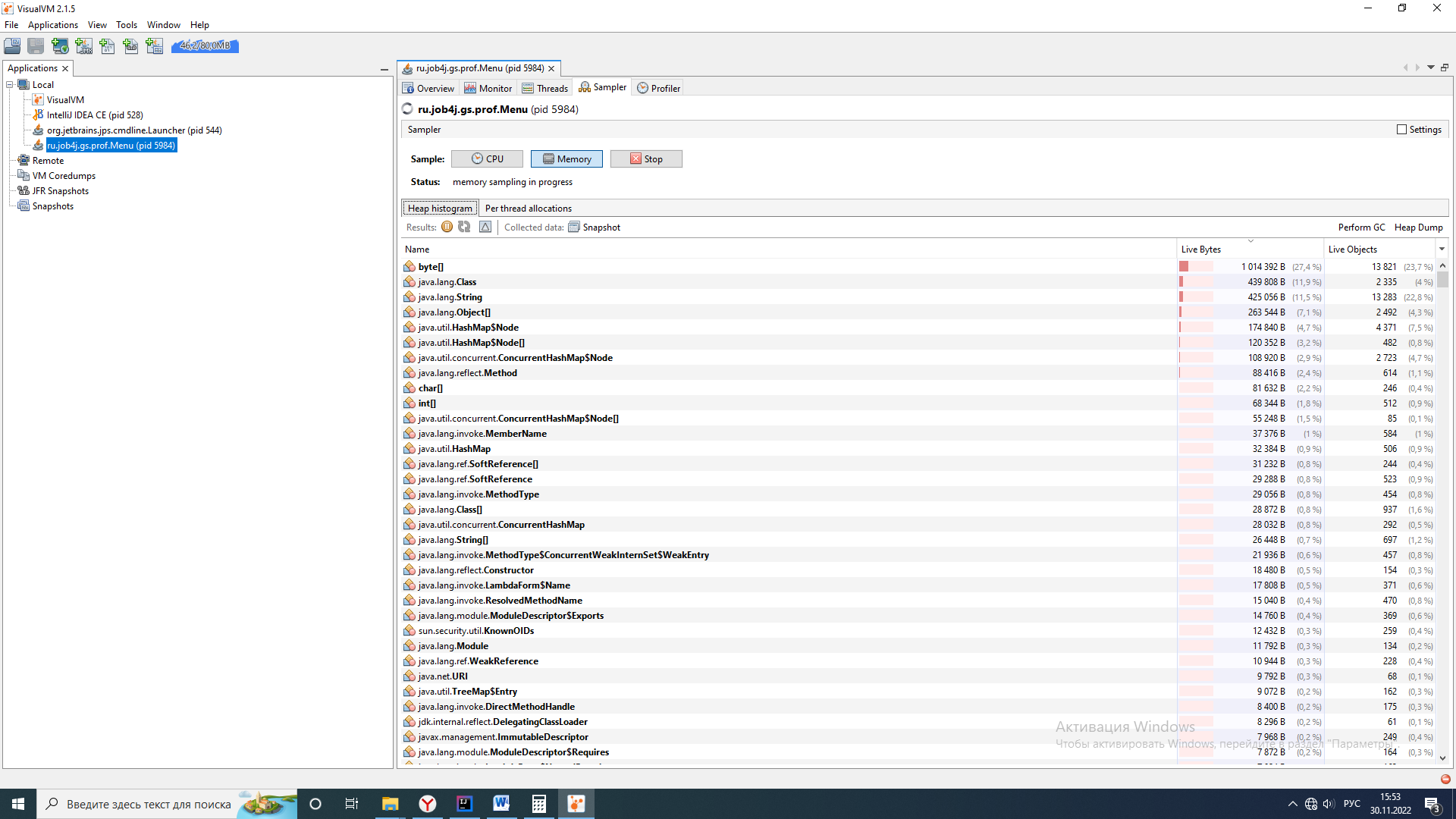
График загрузки процессора на каждой сборке. Сортировка пузырьком потребовала больше всего ресурсов процессора:



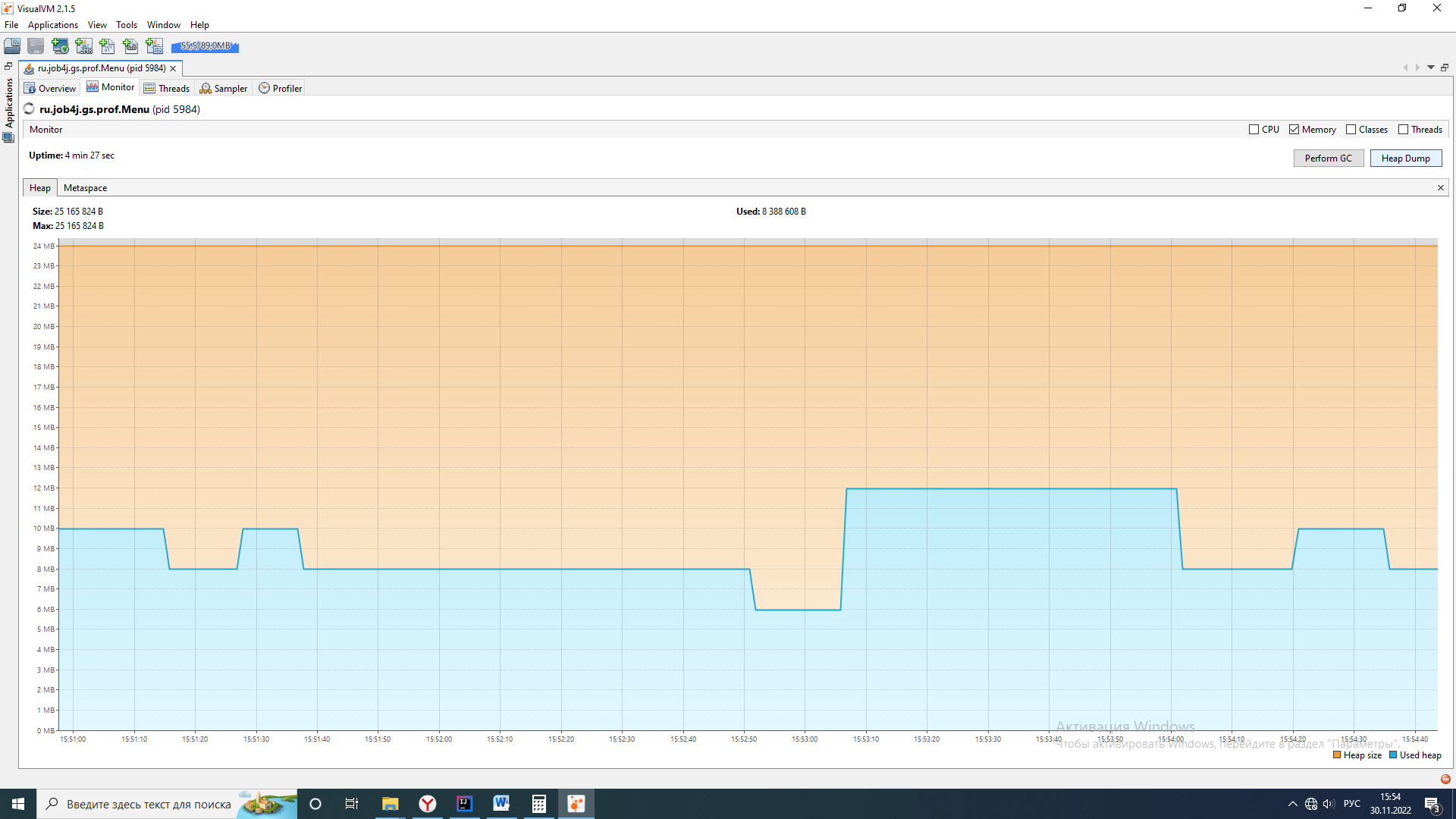
1. **Сборщик ZGC GC**

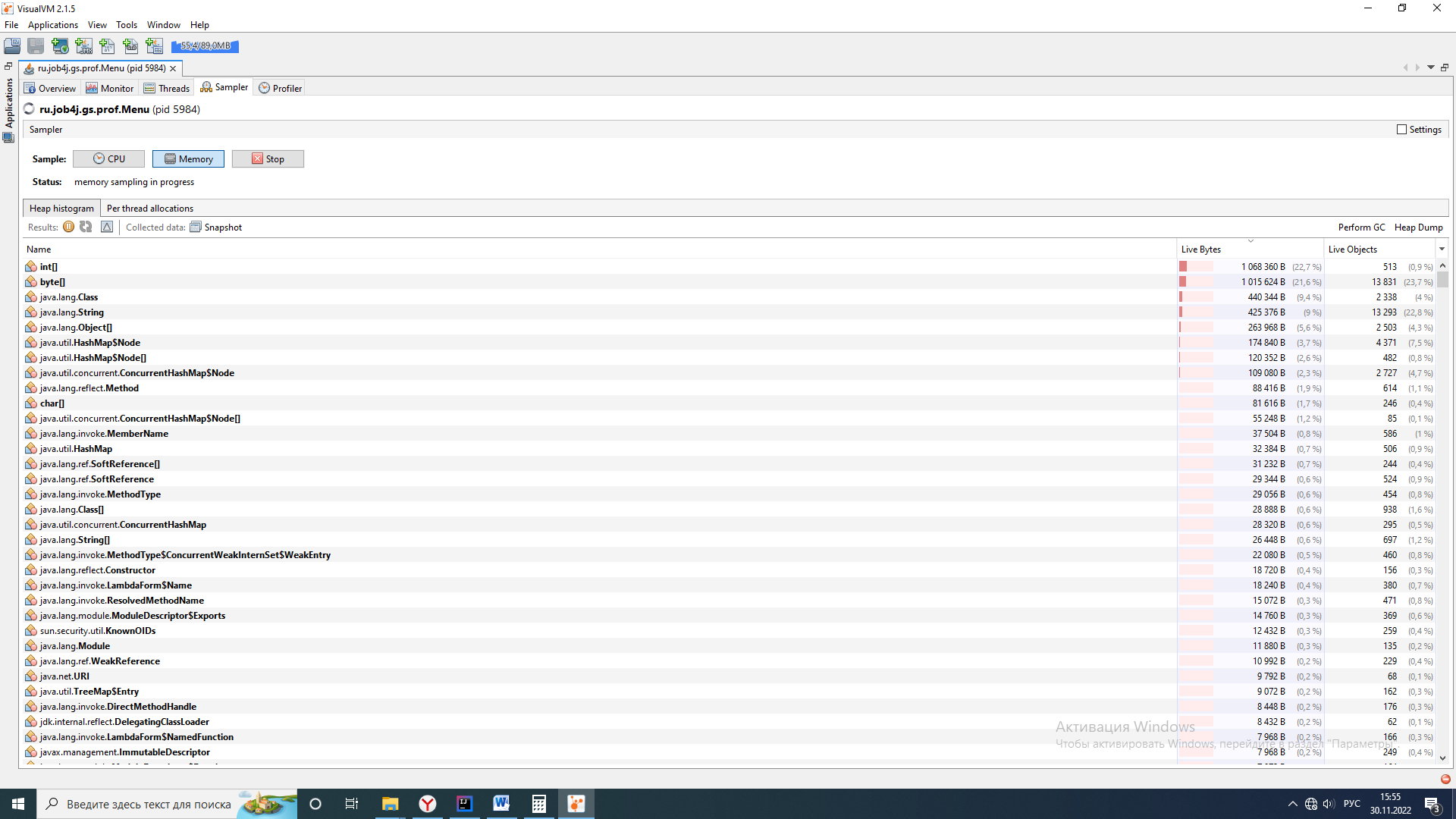
Запускаем программу. Можно заметить, что при запуске в хип поступило почти 10 мегабайт объектов. Затем произошла малая сборка мусора. Дальше можем наблюдать поднятие до 10 и снова сборку:

****



Далее создаем массив на 250000 элементов. Мы можем его заметить - int[] вырос на  размер массива (1\_000\_000).

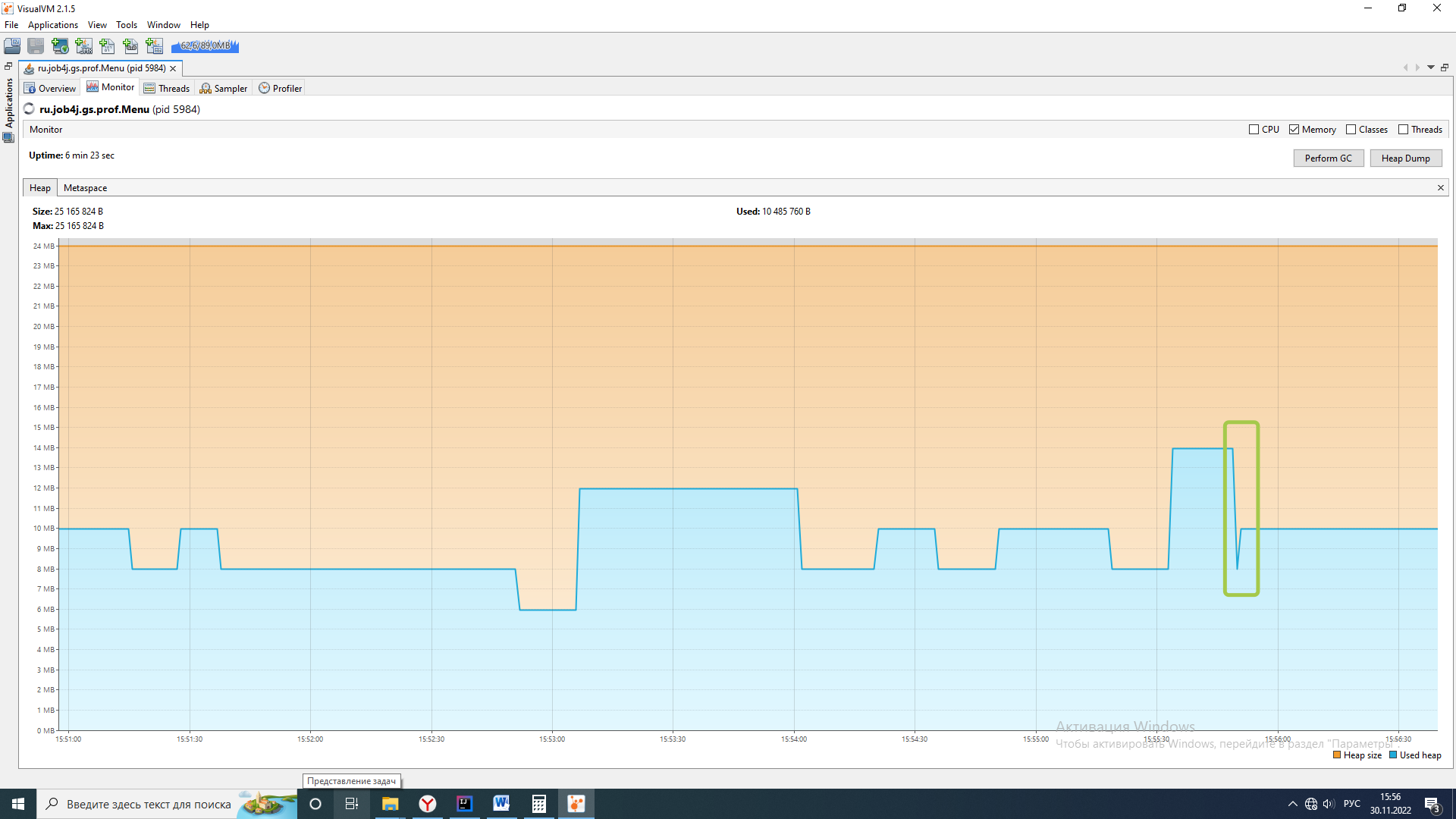




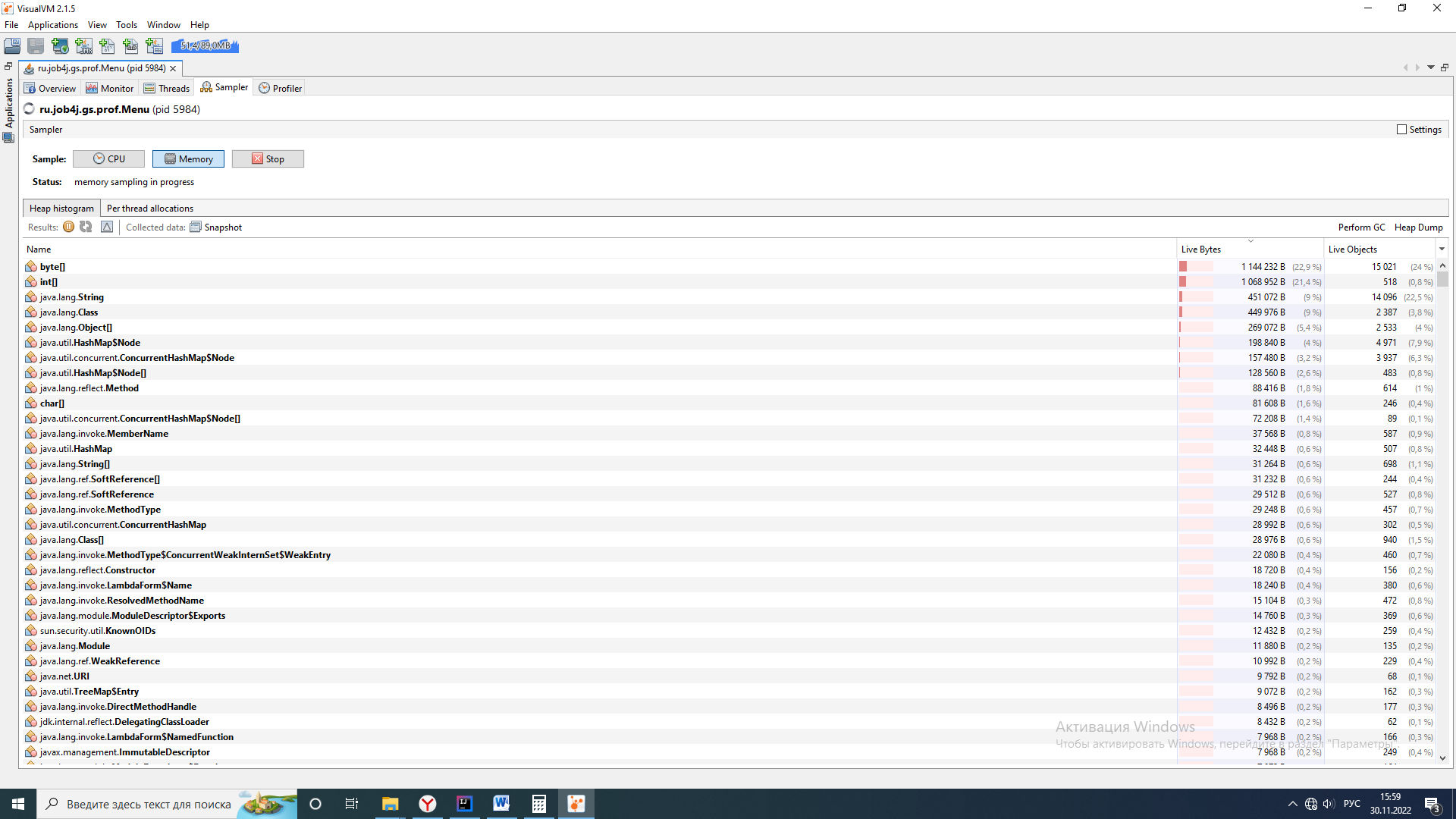
**Сортировка слиянием.**

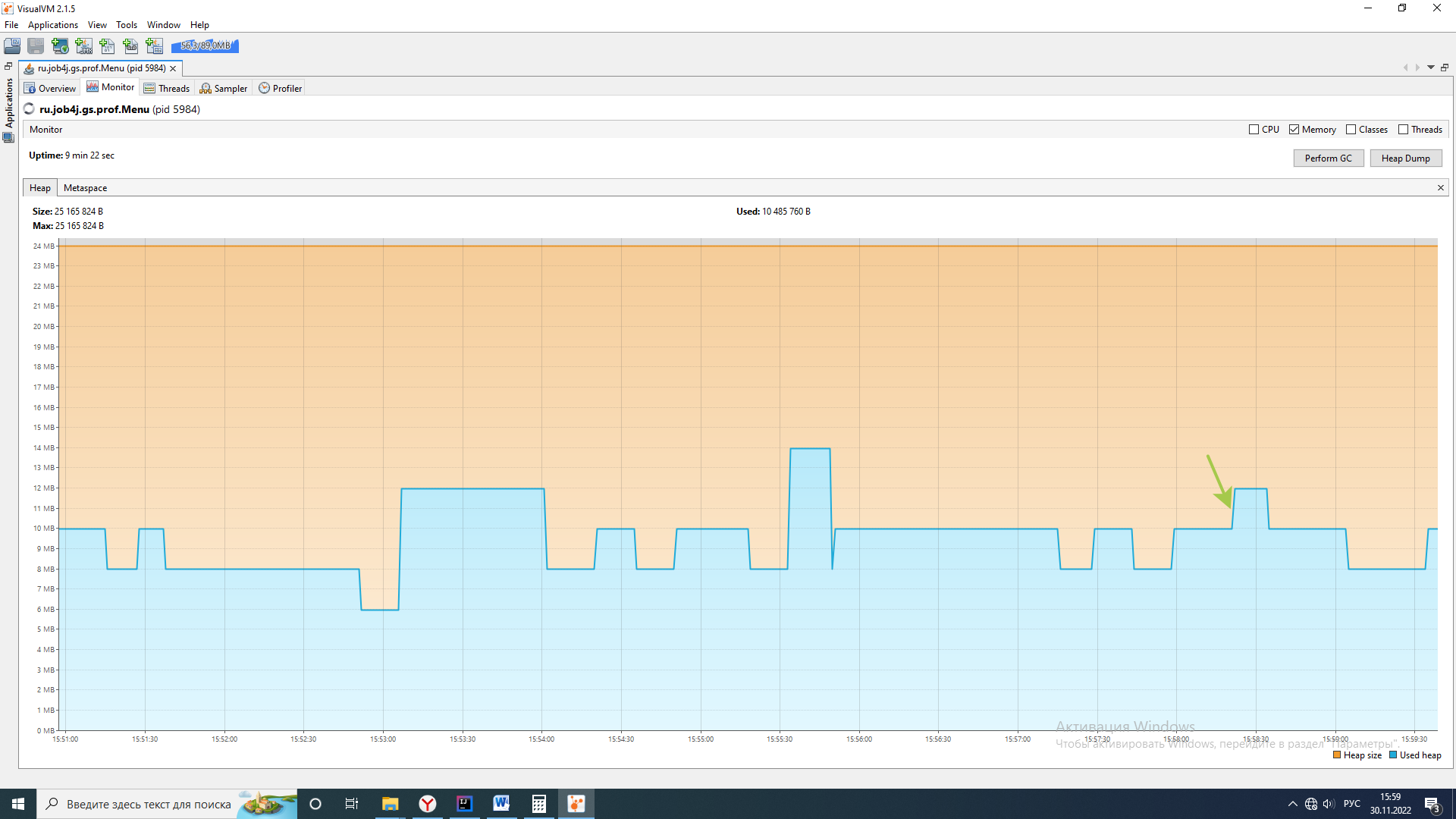
Делаем сортировку слиянием и видим, что объем int[] не возрос. Вся сортировка заняла менее1 миллисекунды.





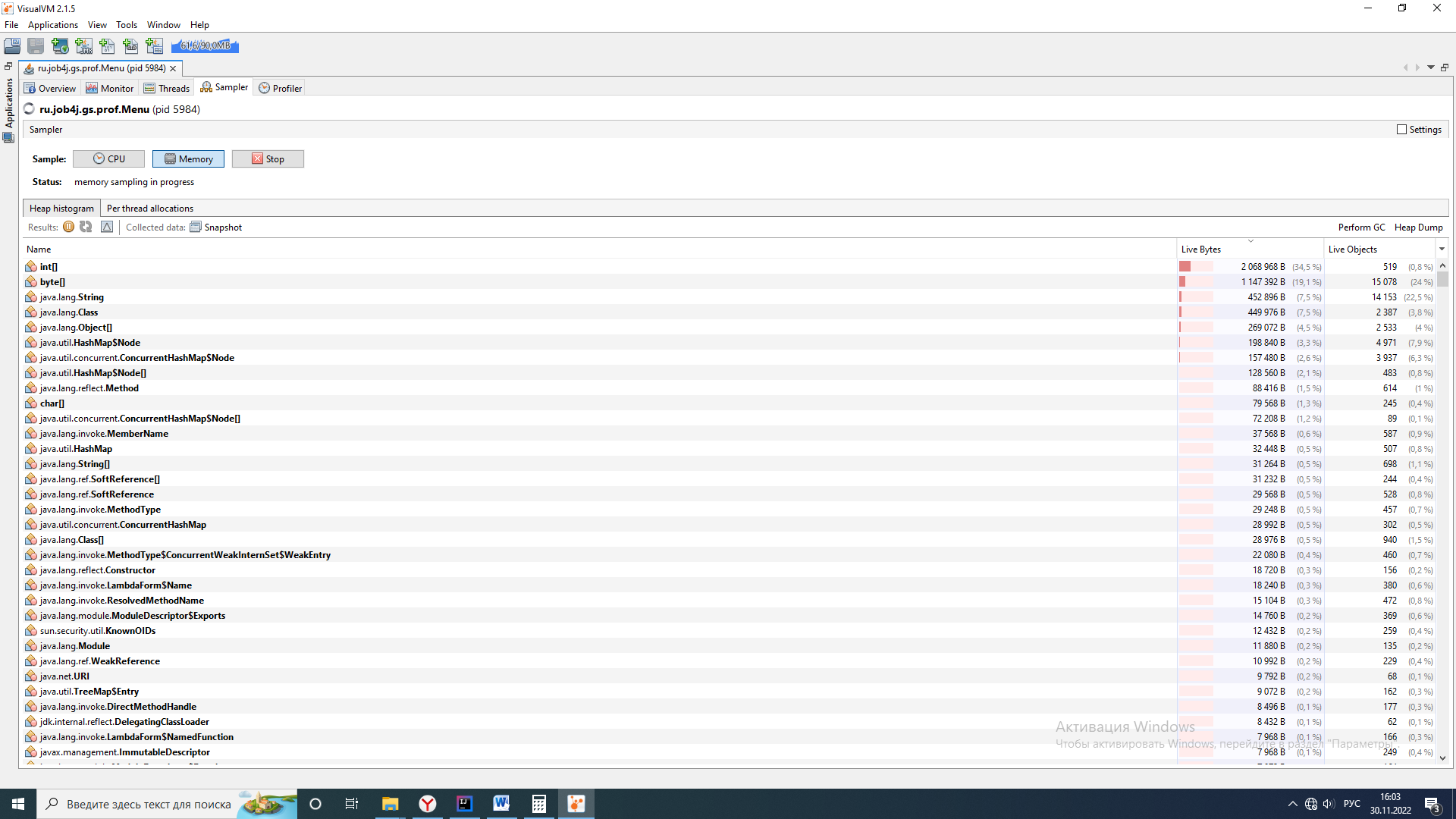
**Сортировка методом вставки.**

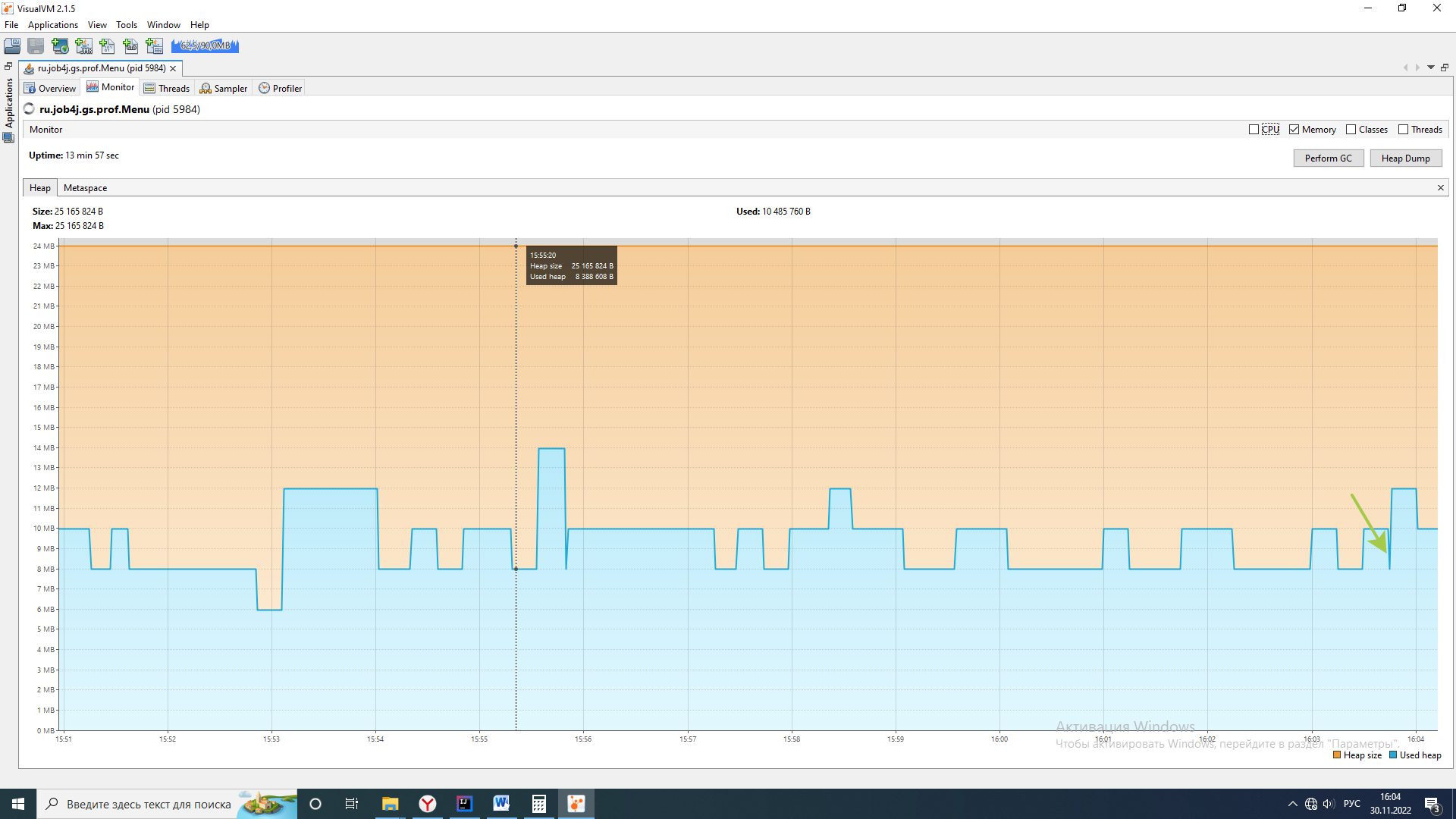
Теперь делаем сортировку методом вставками. Можем наблюдать, что хип вырос и сразу же произошла сборка до старого значения. Время сортировки составило 6 мсек. 



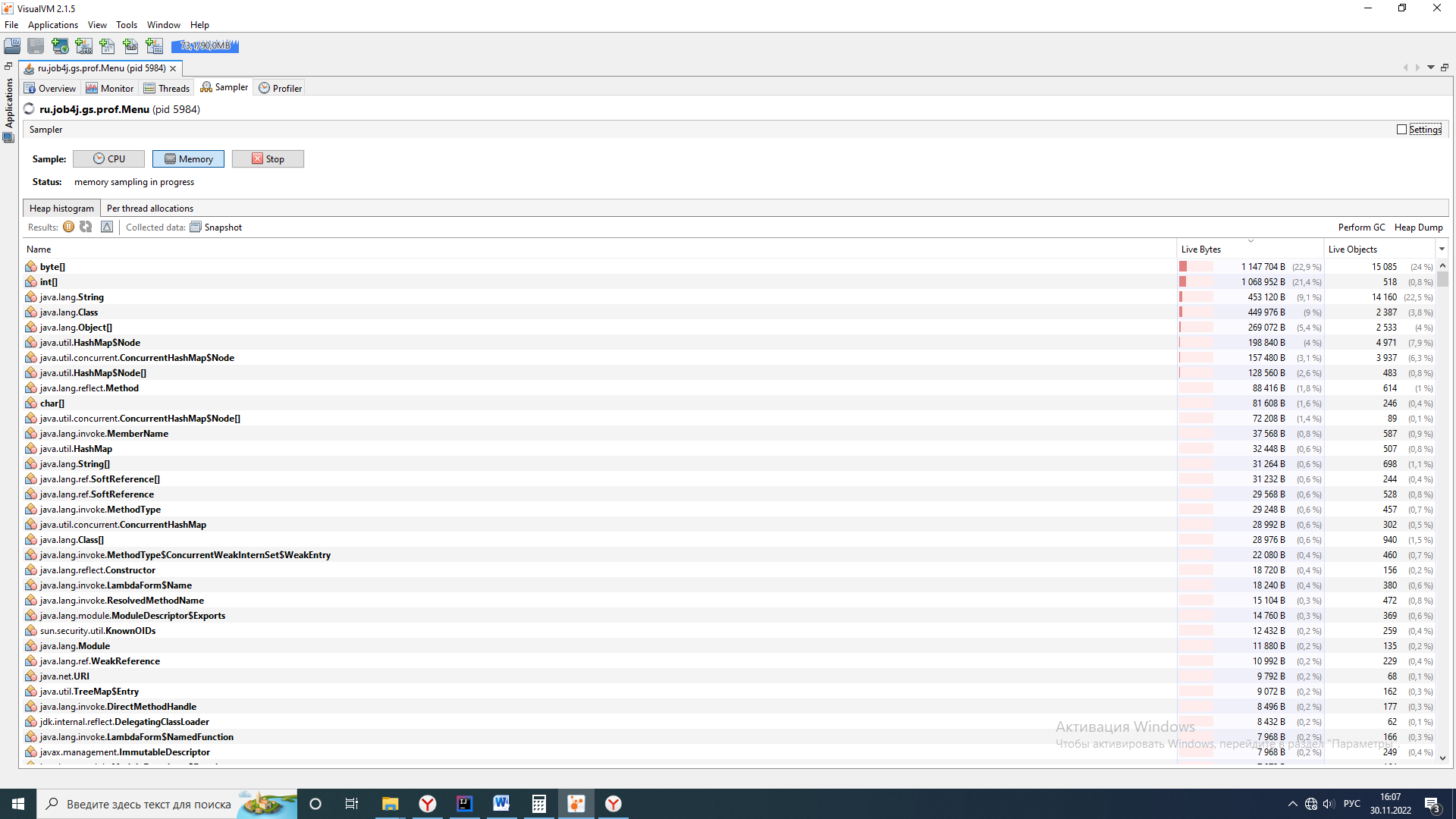
**Сортировка пузырьком.**

Начинаем сортировку пузырьком. В хип добавился клон массива:

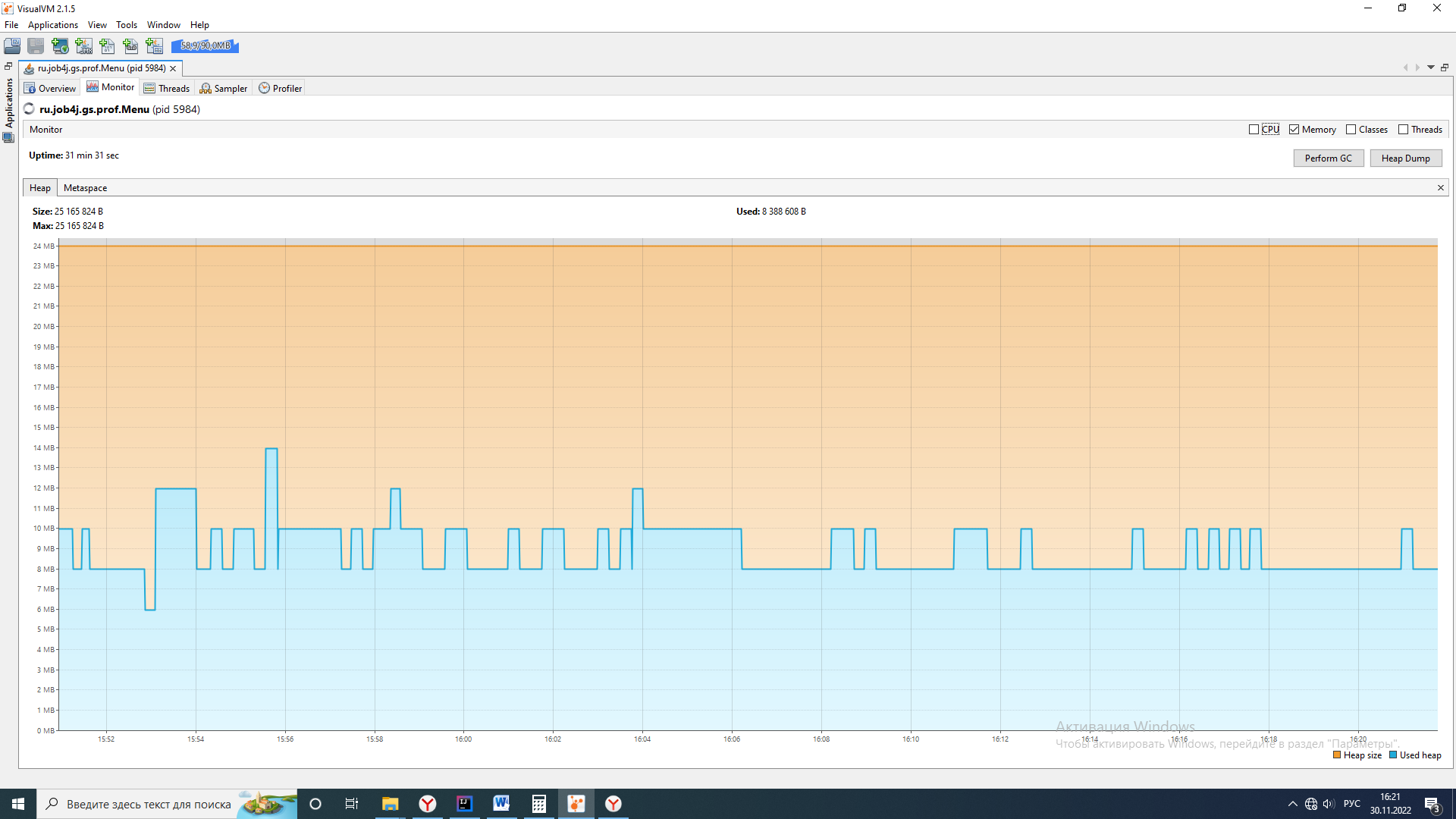




Затем происходит полная сборка.



Полная диаграмма работы приложения.



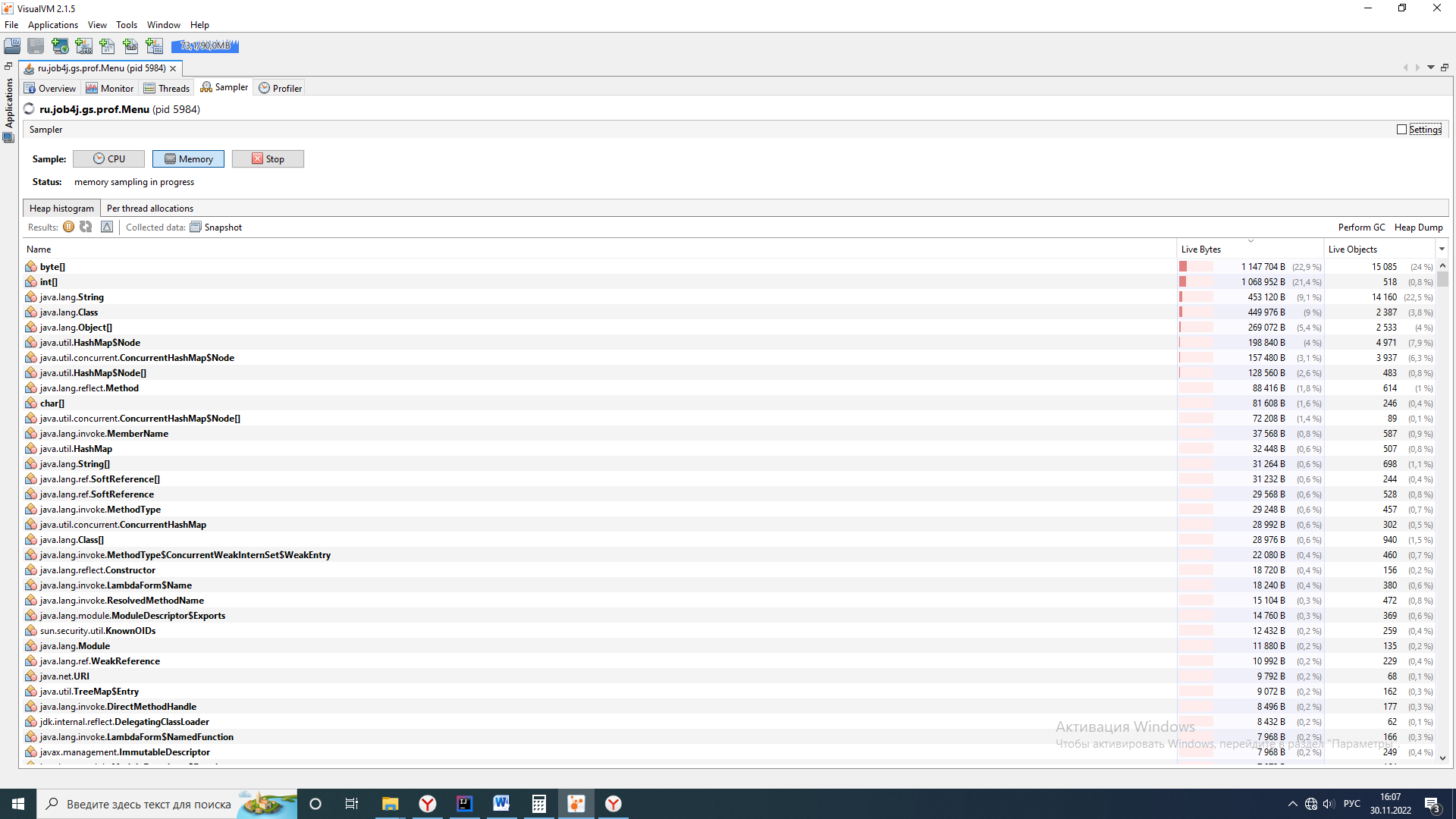


График работы CPU. Сортировка пузырьком заняла максимальное количество ресурсов.

