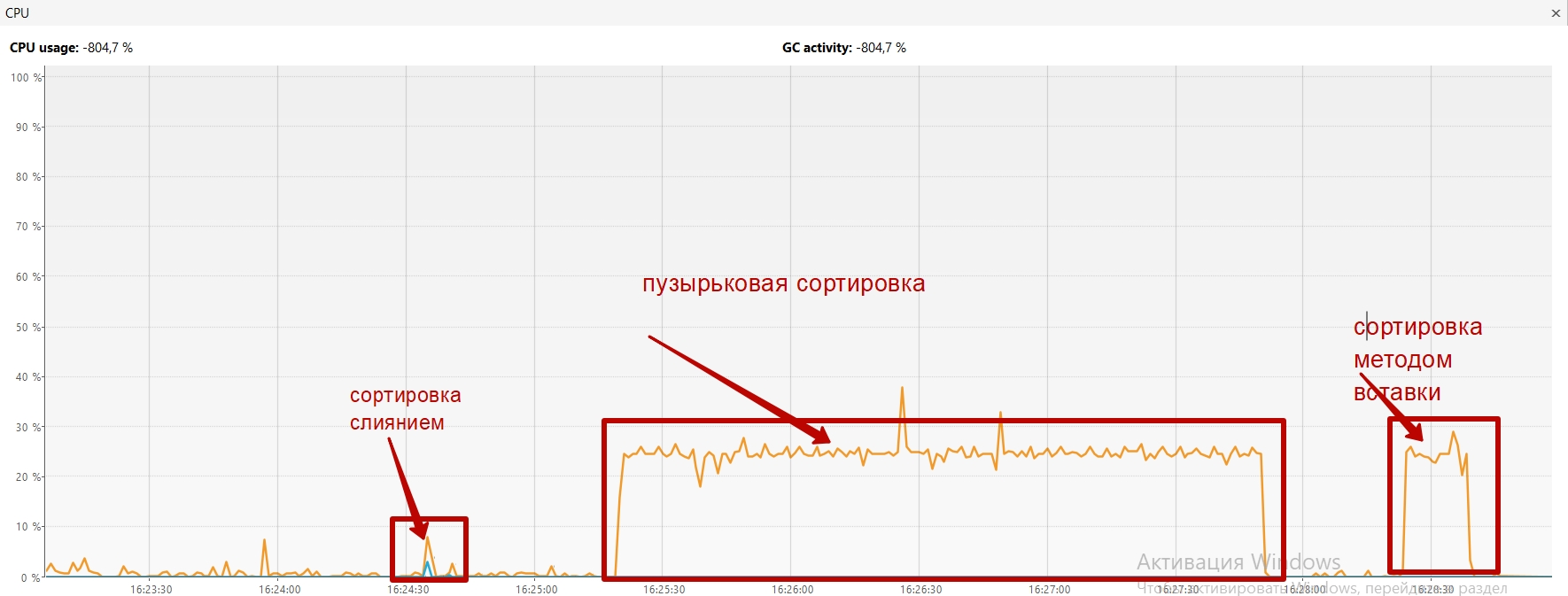
Parallel GC

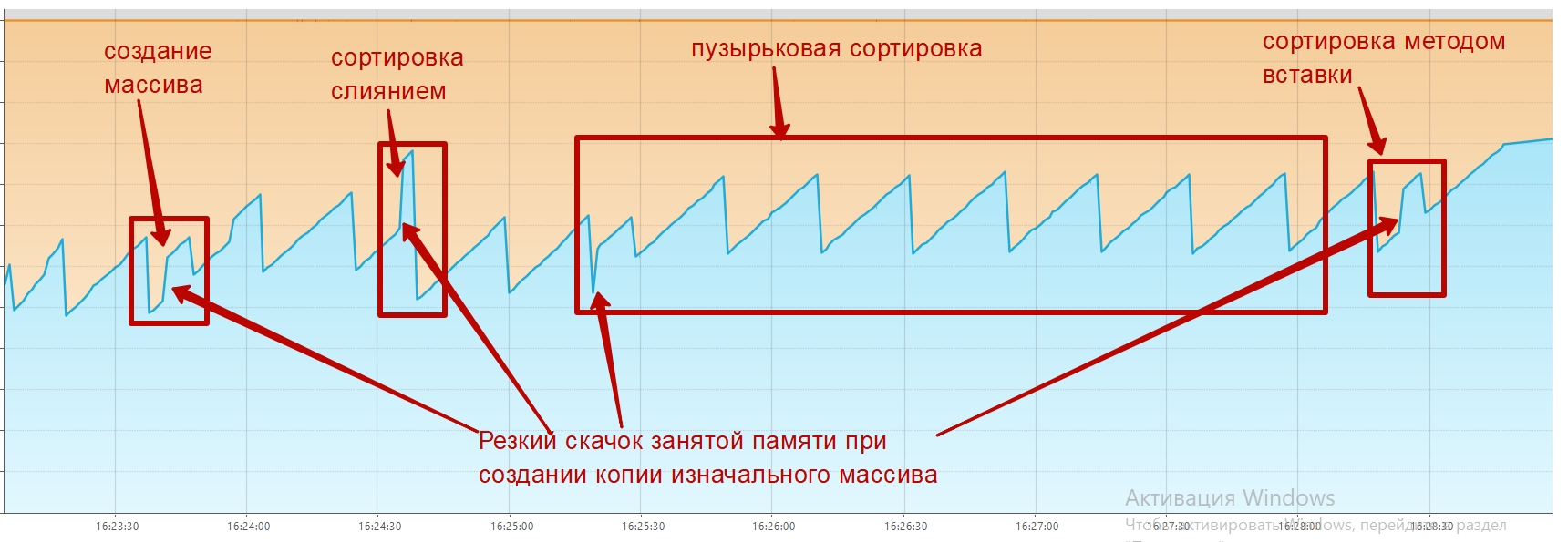
Время сортировки:  
слиянием 1 сек  
пузырьком 151 сек

методом вставки 15 сек

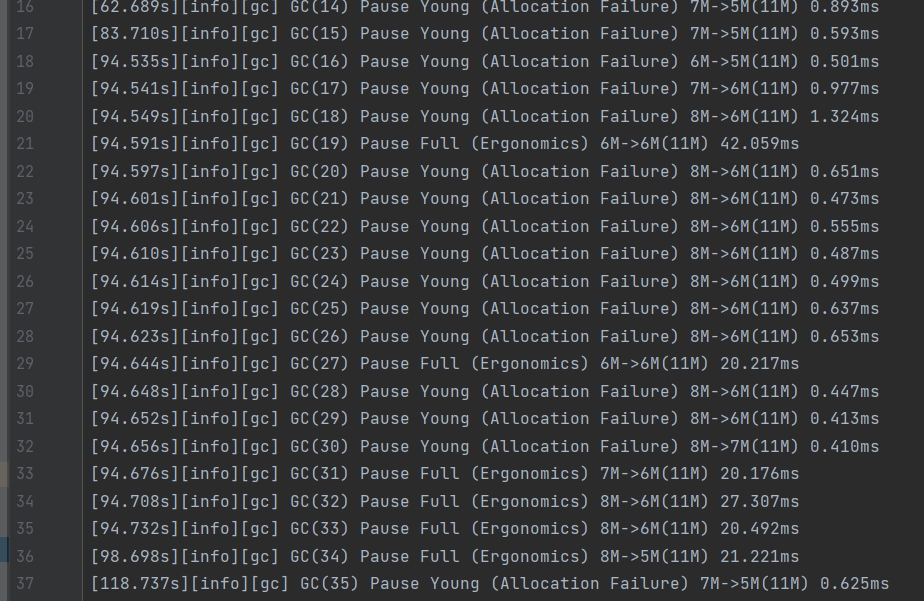
Нагрузка на процессор



Нагрузка на память, выглядит пилообразно, из за того что происходит частая очистка   
Тут видно что сортировка слиянием требует для своей работы много памяти, так как создает очень много малых массивов при своей работе, отчего young generation очень быстро переполняется и GC приходятся очень часто



вызывать сборку мусора «Pause Young» (очистка young generation).



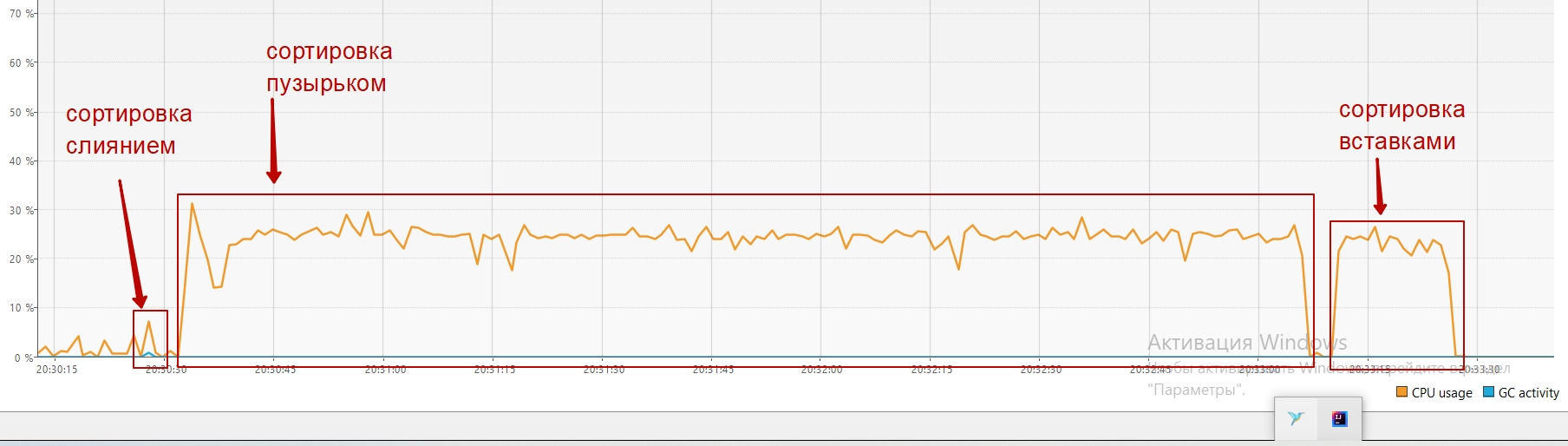
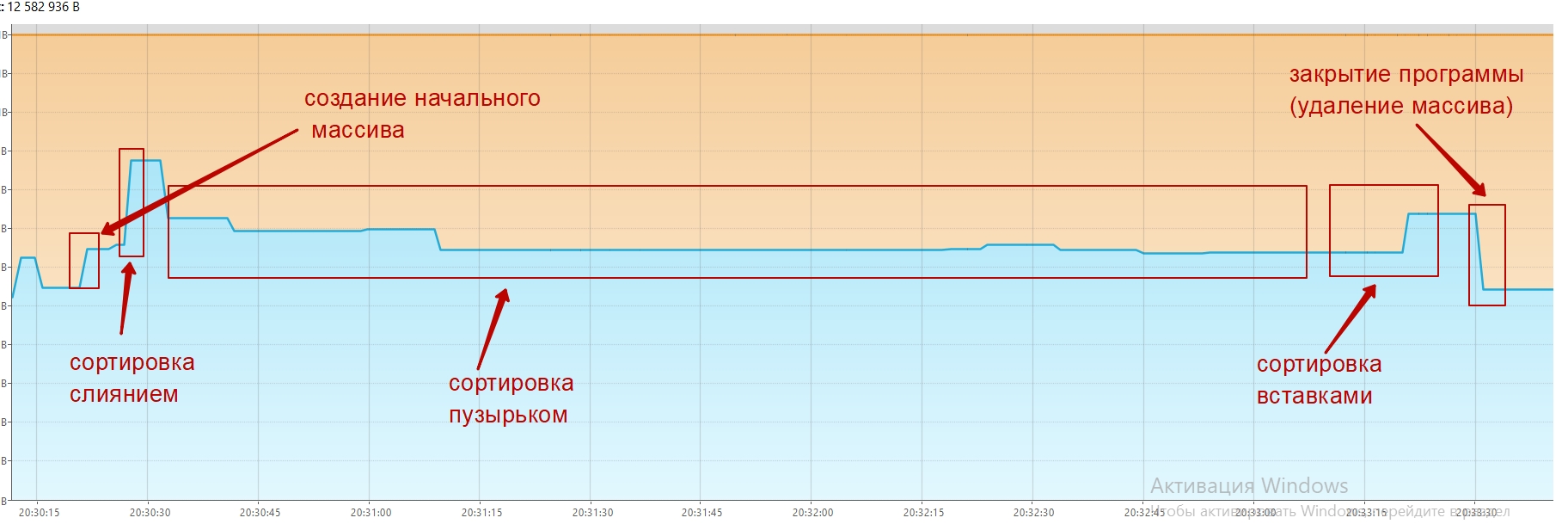
В конце работы метод сортировки создает большие массивы, и так как для них не хватает места в young generation их сразу же переводят в old generation, и поэтому часто вызывается сборка «Pause Full» (очистка old generation)

G1 GC

Время сортировки:  
слиянием 0,122 сек  
пузырьком 93 сек

методом вставки 15 сек

Особенностью данного сборщика является то что, молодое и старое поколение не разделены физически а находятся в одном пространстве, а сама куча разделена на небольшие сектора (от 1 по умолчанию и до 32 мегабайт), во вторых он производит фоновую (не останавливая работу приложения) индексацию недостижимых объектов. Потом он выделяет сектора с самым большим количеством мусора и копирует оттуда все нормальные объекты в последствии зачищая сектор к тому же делает это в нескольких потоках. Это позволяет работать приложению с минимальными задержками. Как можно увидеть на картинке ниже, сборщик не ждет переполнения памяти, а чистит его постоянно, отчего загрузка памяти «ровная»  
   
загрузка процессора:



ZGC

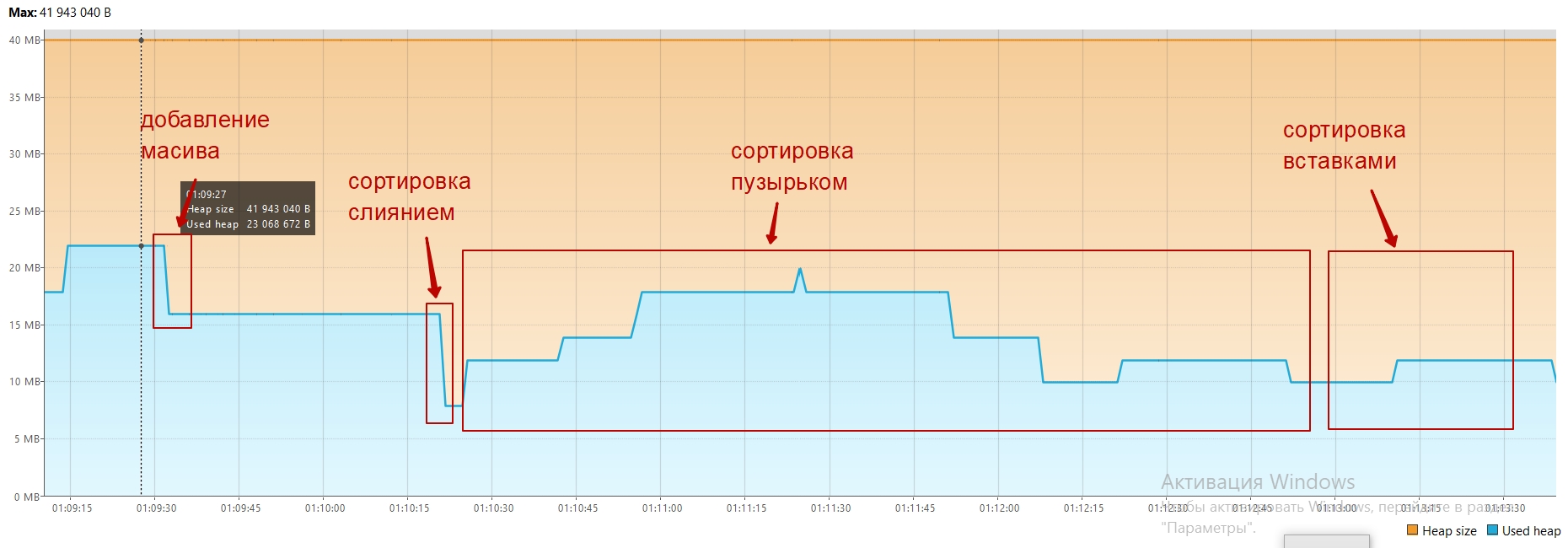
Время сортировки:  
слиянием 0,132сек  
пузырьком 154 сек

методом вставки 15 сек

при его проектировании ставились следующие цели:

1. Поддерживать паузы STW на уровне меньше одной миллисекунды.
2. Сделать так, чтобы паузы не увеличивались с ростом размера кучи, количества живых объектов или количества корневых ссылок.
3. Поддерживать кучи размером до 16 ТБ.

Это было достигнуто за счет того, что, отметкой о достижимости объектов и последующим копированием в новые сектора занимается не только сам сборщик мусора, но и само приложение что добавляет дополнительную сложность при работе с объектами для самого приложения. Поэтому чем больше приложение создает много мусорных объектов, тем больше размер кучи необходим сборщику для минимизации времени отклика. Это можно хорошо увидеть на картинке снизу:



Тут можно увидеть что в отличии от других видов сборщиков, объем занятой памяти при добавлении массива и работе сортировки не только не увеличивается а иногда и уменьшается. Тут дело в особенности реализации этого сборщика мусора. Это делается для минимизации времени отклика программы, но плата за это в усложнении работы программы и увеличении расхода ресурса процессора и памяти.

