**1. Serial.**

Операции:

1. Создание массива – 7 миллисекунд;

2. Сортировка пузырьком – 1 мин 41 сек;

3. Сортировка вставками – 4,5 сек;

4. Сортировка слиянием – 80 миллисекунд.

После запуска программы на графике видны скачки показателя Used Heap. Это из-за подраздела eden (~3,3МБ) раздела Yong Generation (размер ~4МБ), который состоит из подразделов eden, survivor 1 и survivor 2. Подраздел eden заполняется данными, а после его заполнения происходит малая сборка мусора, т. е. удаление уже не нужных данных.

При создании массива, в хипе наблюдается скачёк. Скачёк связан с тем, что в хип добавляется созданный массив, который занимает некоторое количество памяти (~1МБ). Данный массив хранится в куче, поэтому куча будет всегда заполнена уже не на ~4МБ, как это было изначально видно на графике, а на ~5МБ. Как раз ~1МБ весит созданный массив ((250000 \* 4 + 16) / 1048576). Сборщик мусора при малой сборке данные массива не удаляет, так как массив используется программой (ссылка на массив не потеряна). Сейчас массив после каждой малой сборки кочует из подраздела survivor 1 в подраздел survivor 2, где сборщик мусора запоминает, сколько раз массив перешёл из одного подраздела в другой.

При пузырьковой сортировке создаётся клон массива, в результате чего в куче появляются ещё данные размером ~1МБ (размер клонированного массива). Это видно по скачку на графике. Данный клон массива также хранится в куче, и куча будет заполнена всегда уже не на ~5МБ, а на ~6МБ. Т. е. сборщик мусора при малых сборках не удаляет уже не только массив созданный методом insert(int elements) класса RandomArray, но и массив созданный методом sort(Data data) класса сортировки.

При сортировке вставками опять создаётся клон массива, в результате чего в куче появляются ещё данные размером ~1МБ (размер клонированного массива). Это видно по скачку на графике. Так как ссылка на клон массива, созданный пузырьковой сортировкой, уже не используется, то данные его будут стёрты при первой же сборке мусора. Так что куча останется заполнена всё так же на ~6МБ.

При сортировке слиянием создаётся не только клон массива, но и массивы при разбиении массива. Поэтому скачок на графике больше чем у пузырьковой сортировки и у сортировки вставками. Клон массива, который был создан во время сортировки вставками, удаляется. Так что куча остаётся заполненной на ~7МБ.

Те же выводы можно сделать исходя из файла логов log\_serial.txt. По строчкам:

[40.506s][info][gc] GC(7) Pause Young (Allocation Failure) 7M->4M(11M) 1.453ms

[59.516s][info][gc] GC(8) Pause Young (Allocation Failure) 7M->5M(11M) 1.578ms

видно, что как раз таки перед малой сборкой мусора GC(8) и был создан массив размером ~1МБ.

По строчкам:

[87.525s][info][gc] GC(9) Pause Young (Allocation Failure) 8M->5M(11M) 1.402ms

[107.533s][info][gc] GC(10) Pause Young (Allocation Failure) 8M->6M(11M) 1.032ms

видно, что перед малой сборкой мусора GC(10) был создан клон массива для пузырьковой сортировки размером ~1МБ.

По строчкам:

[253.611s][info][gc] GC(16) Pause Young (Allocation Failure) 9M->7M(11M) 0.918ms

[253.621s][info][gc] GC(17) Pause Young (Allocation Failure) 10M->10M(11M) 0.038ms

[253.637s][info][gc] GC(18) Pause Full (Allocation Failure) 10M->6M(11M) 16.246ms

видно, что перед малой сборкой мусора GC(17) был создан клон массива для сортировки вставками, после чего произошла первая полная сборка в результате заполнения памяти.

По строчкам:

[253.669s][info][gc] GC(24) Pause Young (Allocation Failure) 10M->7M(11M) 0.235ms

[253.672s][info][gc] GC(25) Pause Young (Allocation Failure) 9M->9M(11M) 0.035ms

[253.686s][info][gc] GC(26) Pause Full (Allocation Failure) 9M->6M(11M) 13.395ms

[268.606s][info][gc] GC(27) Pause Young (Allocation Failure) 10M->7M(11M) 1.136ms

видно, что перед малой сборкой мусора GC(25) был создан клон массива для сортировки вставками, после чего произошла вторая полная сборка в результате заполнения памяти.

Для малой и полной сортировки происходит остановка работы приложения. Остановка для малой сборки мусора, которая касается только раздела Yong Generation достаточно мала: от 0,156 до 6,576 миллисекунд. Остановка же для полной сборки мусора гораздо дольше: 13,395 до 16,246 миллисекунд.

**2. Parallel.**

1. Создание массива – 7 миллисекунд;

2. Сортировка пузырьком – 1 мин 41 сек;

3. Сортировка вставками – 4,5 сек;

4. Сортировка слиянием – 76 миллисекунд.

Так как единственное отличие Serial GC от Parallel GC, это то, что первый выполняет сборку мусора в одном потоке, а второй выполняет сборку мусора в нескольких потоках, графики у данных GC схожи по своему виду, за тем исключением, что у Parallel GC на графике присутствуют пики, которые говорят нам о том, что сбор мусора происходит в несколько потоков.

При создании массива на графике происходит скачок на размер создаваемого массива ~1МБ. Данный скачок достигает предельного размера раздела Yong Generation, в результате в той части eden, которая принадлежит одному из потоков, происходит малая сборка. А в той части eden, которая принадлежит другому потоку, сборка мусора не происходит и по графику видно, что куча продолжает заполняться данными до малой сборки.

По строкам лога можно определить операции производимые программой. Из данных срок:

[67.409s][info][gc] GC(12) Pause Young (Allocation Failure) 6M->5M(11M) 0.433ms

[84.419s][info][gc] GC(13) Pause Young (Allocation Failure) 7M->5M(11M) 0.405ms

видно, что создание массива произошло перед малой сборкой мусора GC(13)

Из-за многопоточности резкого скачка при пузырьковой сортировке и сортировке вставками не наблюдается, но по логу отследить их можно:

Сортировка пузырьком произошла перед сборкой GC(16):

[103.395s][info][gc] GC(15) Pause Young (Allocation Failure) 7M->6M(11M) 0.252ms

[120.432s][info][gc] GC(16) Pause Young (Allocation Failure) 8M->6M(11M) 0.322ms

[120.453s][info][gc] GC(17) Pause Full (Ergonomics) 6M->5M(11M) 20.451ms

[133.435s][info][gc] GC(18) Pause Young (Allocation Failure) 7M->5M(11M) 0.499ms

Сортировка вставками произошла перед сборкой GC(28):

[269.653s][info][gc] GC(27) Pause Young (Allocation Failure) 7M->5M(11M) 0.202ms

[269.655s][info][gc] GC(28) Pause Young (Allocation Failure) 7M->6M(11M) 0.218ms

[269.659s][info][gc] GC(29) Pause Young (Allocation Failure) 8M->7M(11M) 0.341ms

[269.675s][info][gc] GC(30) Pause Full (Ergonomics) 7M->5M(11M) 16.287ms

У сортировки слиянием скачок на графике ярко выражен. Связано это с тем, что создаётся не только клон массива, но ещё и происходит разбиение массива. В логах это произошло перед сборкой GC(42):

[269.724s][info][gc] GC(41) Pause Young (Allocation Failure) 8M->6M(11M) 0.113ms

[269.727s][info][gc] GC(42) Pause Young (Allocation Failure) 8M->7M(11M) 0.114ms

[273.512s][info][gc] GC(43) Pause Young (Allocation Failure) 9M->7M(11M) 0.534ms

[291.520s][info][gc] GC(44) Pause Young (Allocation Failure) 9M->7M(11M) 0.826ms

Судя по логам у Parallel GC в отличии от Serial GC полная и малая сборка происходят гораздо чаще и при этом они не происходят при полном заполнении кучи. Связано это с тем, что каждому потоку принадлежит свой участок памяти.

Остановка для малой сборки мусора: от 0,107 до 0,826 миллисекунд. Остановка для полной сборки мусора: 16.287 до 20,451 миллисекунд.

В сравнении с остановками Serial GC у Parallel GC остановки продолжительнее.

**3. G1.**

1. Создание массива – 9 миллисекунд;

2. Сортировка пузырьком – 1 мин 42 сек;

3. Сортировка вставками – 4,5 сек;

4. Сортировка слиянием – 87 миллисекунд.

GC G1 по графику уже сильнее отличается от предыдущих GC, хотя общие принципы так же прослеживаются.

Из графика так же видны скачки в хипе в результате создания новых объектов, но в отличие от предыдущих графиков скачки не в виде пиков, а в виде «пеньков». Можно сделать вывод, что сборка происходит не сразу, как только достигается предел памяти. G1 может себе это позволить благодаря высокой пропускной способности.

В логах можно проследить особенность данного G1:

[4.420s][info][gc] GC(9) Concurrent Mark Cycle

[4.424s][info][gc] GC(9) Pause Remark 5M->5M(12M) 1.140ms

[4.426s][info][gc] GC(9) Pause Cleanup 5M->5M(12M) 0.024ms

Сначала происходит пометка живых объектов, далее дополнительный поиск не учтённых ранее живых объектов, а потом очистка и поиск пустых регионов.

Остановка для малой сборки мусора: от 0,290 до 3,581 миллисекунд. Остановка для полной (смешанной) сборки мусора: 1,248 до 1,673 миллисекунд.

Полная остановка значительно меньше чем у Serial GC и Parallel GC.

**4. ZGC.**

1. Создание массива – 7 миллисекунд;

2. Сортировка пузырьком – 1 мин 44 сек;

3. Сортировка вставками – 5,5 сек;

4. Сортировка слиянием – 87 миллисекунд.

График ZGC GC схож с графиком G1 GC. Так же как и G1 сборщик мусора ZGC может позволить себе не выполнять сборку сразу, как будет достигнут предел памяти благодаря высокой пропускной способности и возможности работать параллельно с приложением.

Из логов видна особенность данного сборщика мусора – отсутствие пауз на малую сбоку:

[3.040s][info][gc] GC(4) Garbage Collection (Allocation Rate) 20M(100%)->8M(40%)

[3.153s][info][gc] GC(5) Garbage Collection (Allocation Stall) 10M(50%)->8M(40%)

[3.249s][info][gc] GC(6) Garbage Collection (Allocation Rate) 10M(50%)->8M(40%)

[3.347s][info][gc] GC(7) Garbage Collection (Allocation Rate) 10M(50%)->8M(40%)

На самом деле паузы есть, но они очень малы. Паузы при таких сборках составляют менее одной миллисекунды. Но в ситуациях, когда у JVM не осталось свободных блоков для размещения новых объектов в логах можно найти STW в привычных для GC временных рамках:

[2.844s][info][gc] GC(3) Garbage Collection (Proactive) 6M(30%)->6M(30%)

[3.038s][info][gc] Allocation Stall (Attach Listener) 13.667ms

[3.040s][info][gc] GC(4) Garbage Collection (Allocation Rate) 20M(100%)->8M(40%)

Остановка для малой сборки мусора: менее одной миллисекунды. Остановка для полной сборки мусора: 1,873 до 15,106 миллисекунд. У ZGC GC самые маленькие паузы во время малой сборки в сравнении с упомянутыми GC.