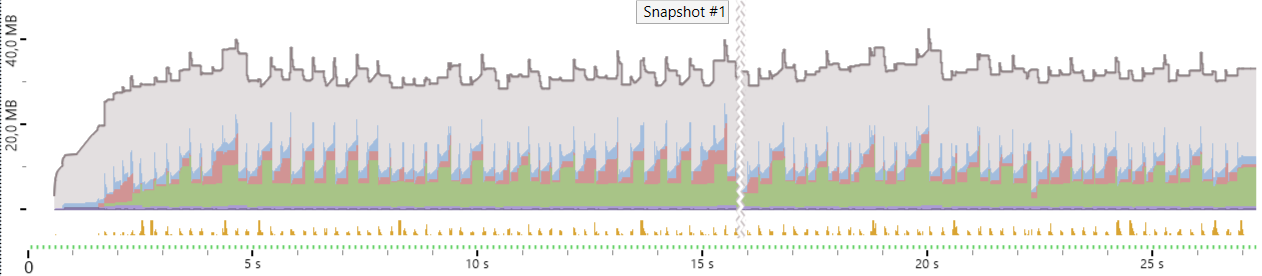
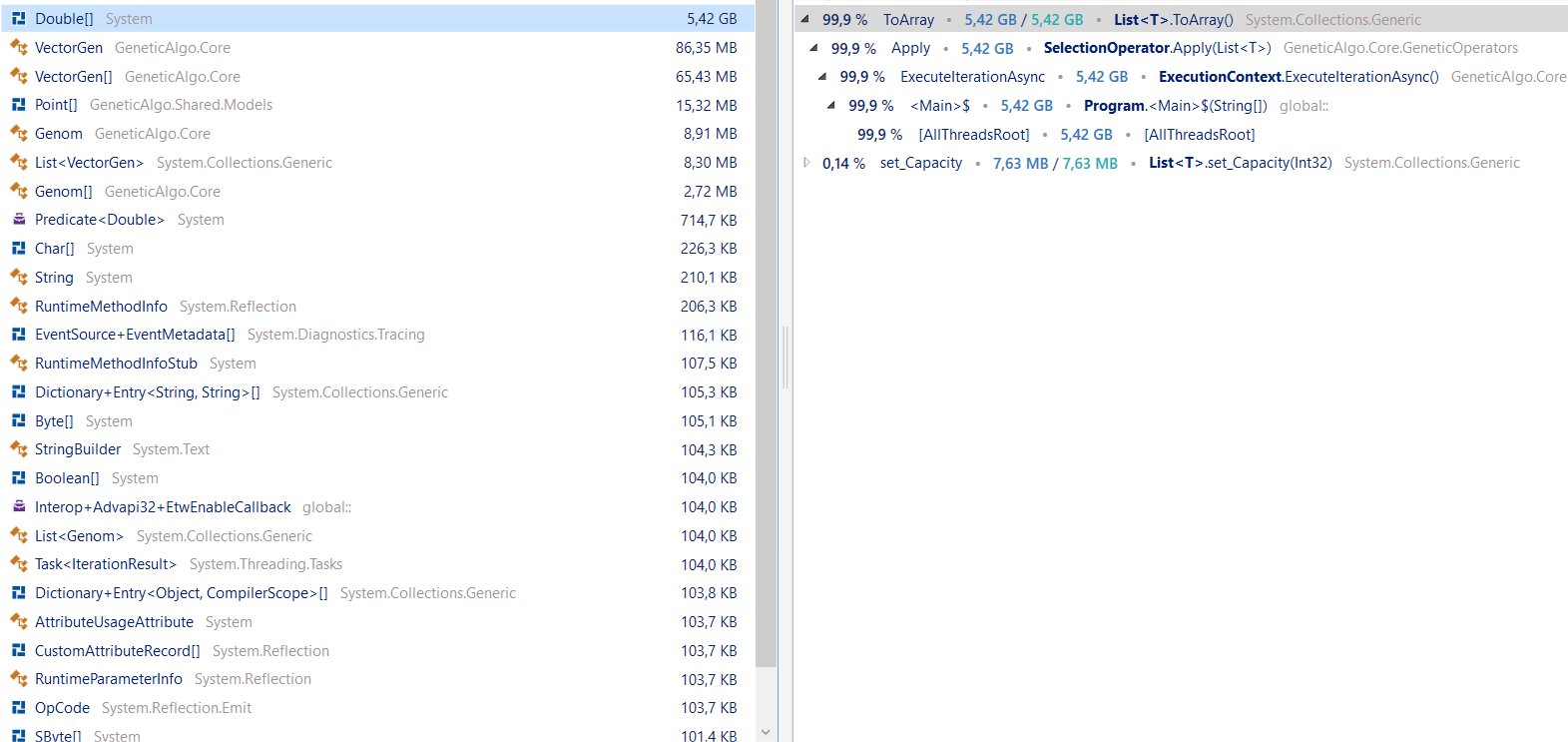
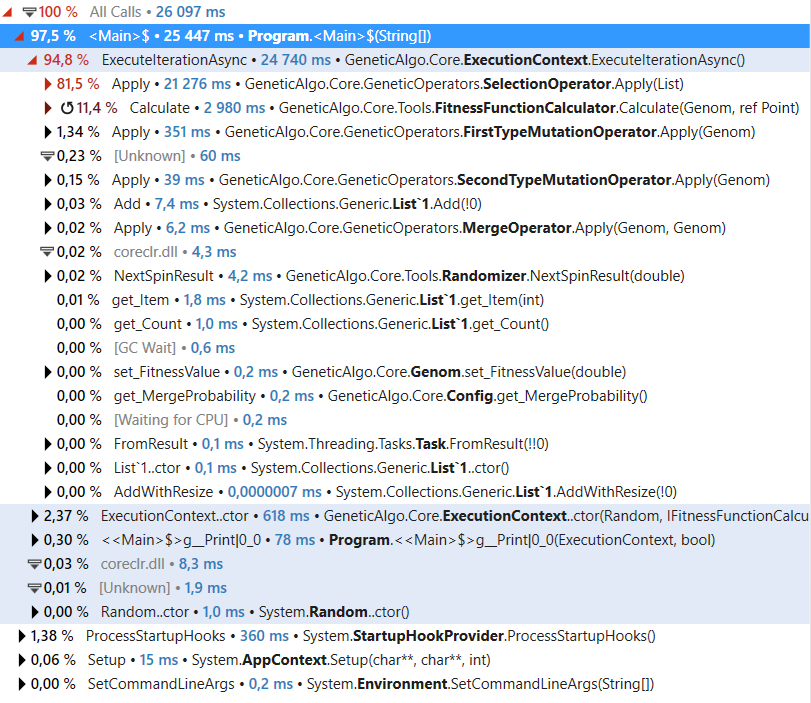
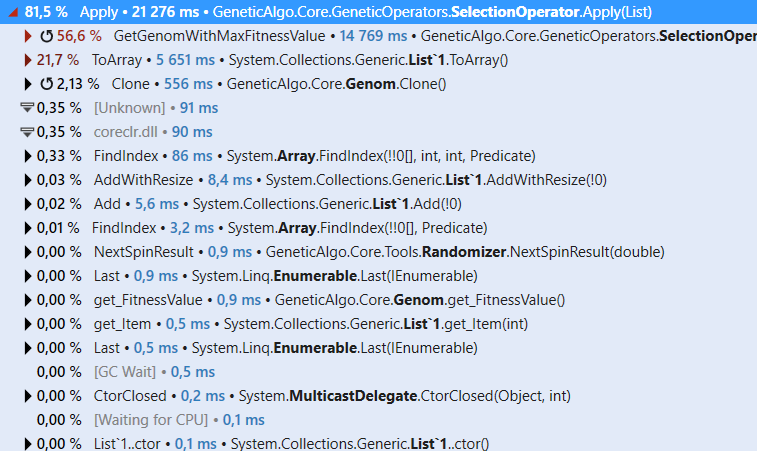
Первая реализация:







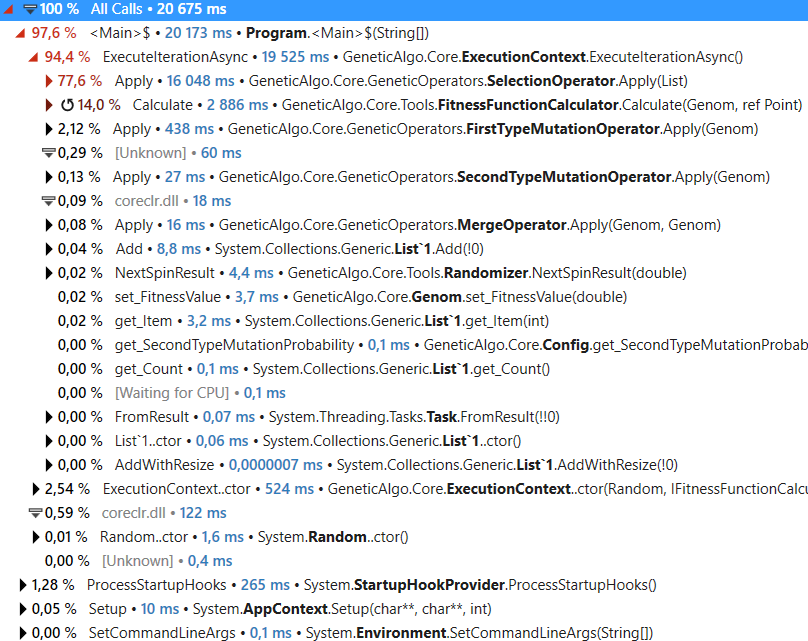
Можно заметить, что большую часть времени занимает вызов функции Apply класса SelectionOperator (а также, на нём выделяется большая часть памяти):



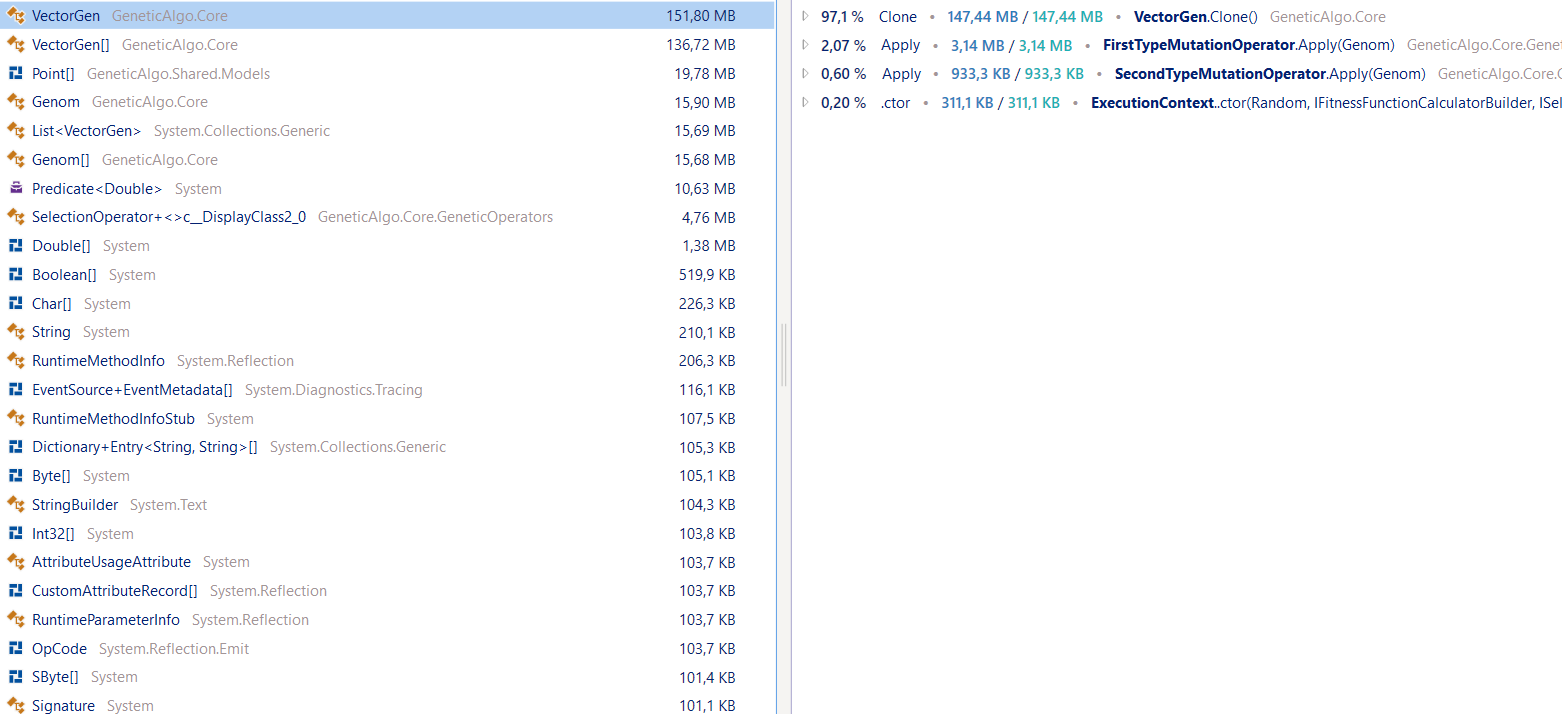
Около четверти времени занимает вызов метода ToArray

Первая оптимизация:

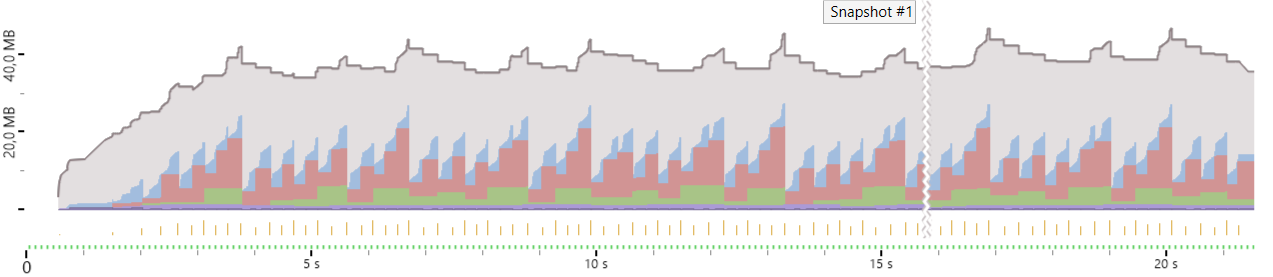
Заменим List на Array и уберём вывод в консоль, в итоге получаем ускорение на 20%



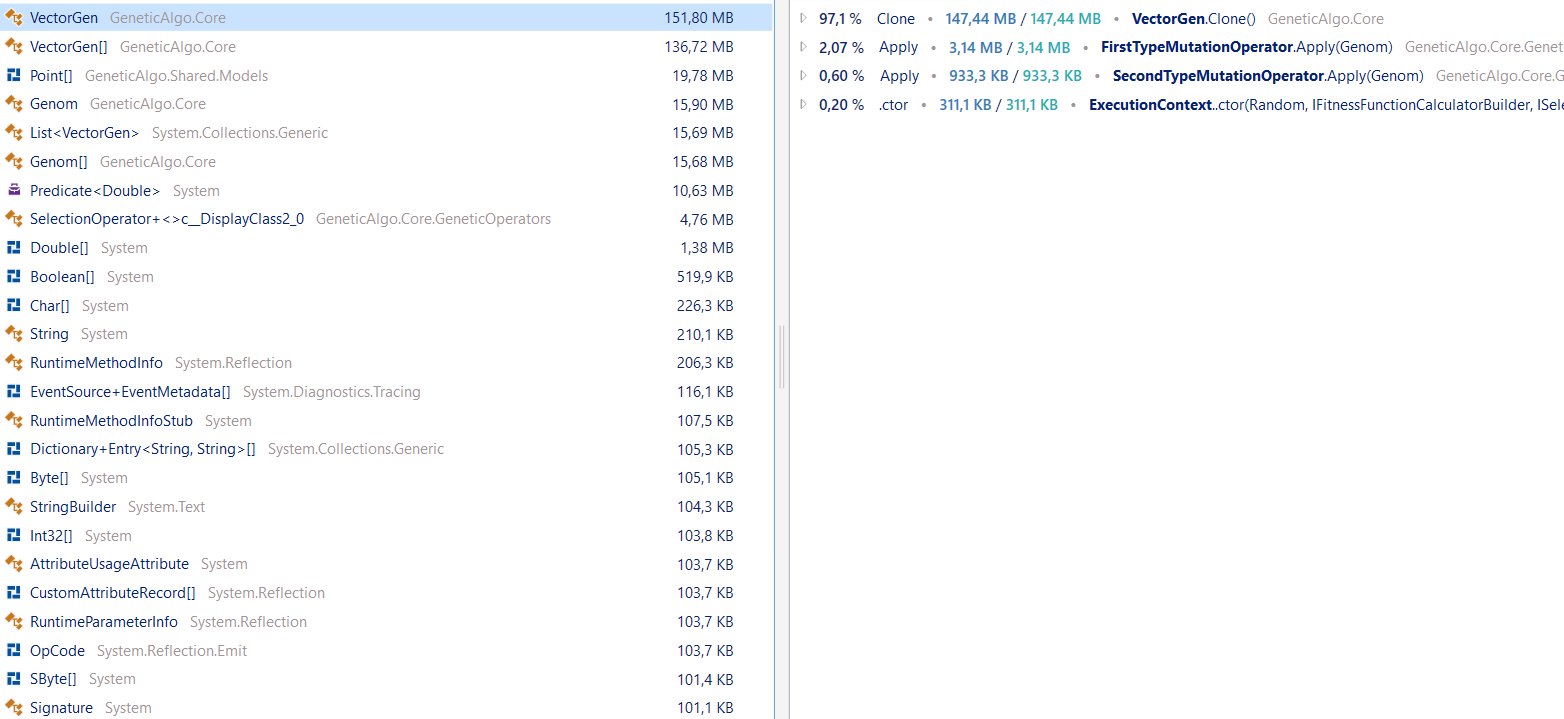
И оптимизацию более 5 Гб памяти (более 95%):



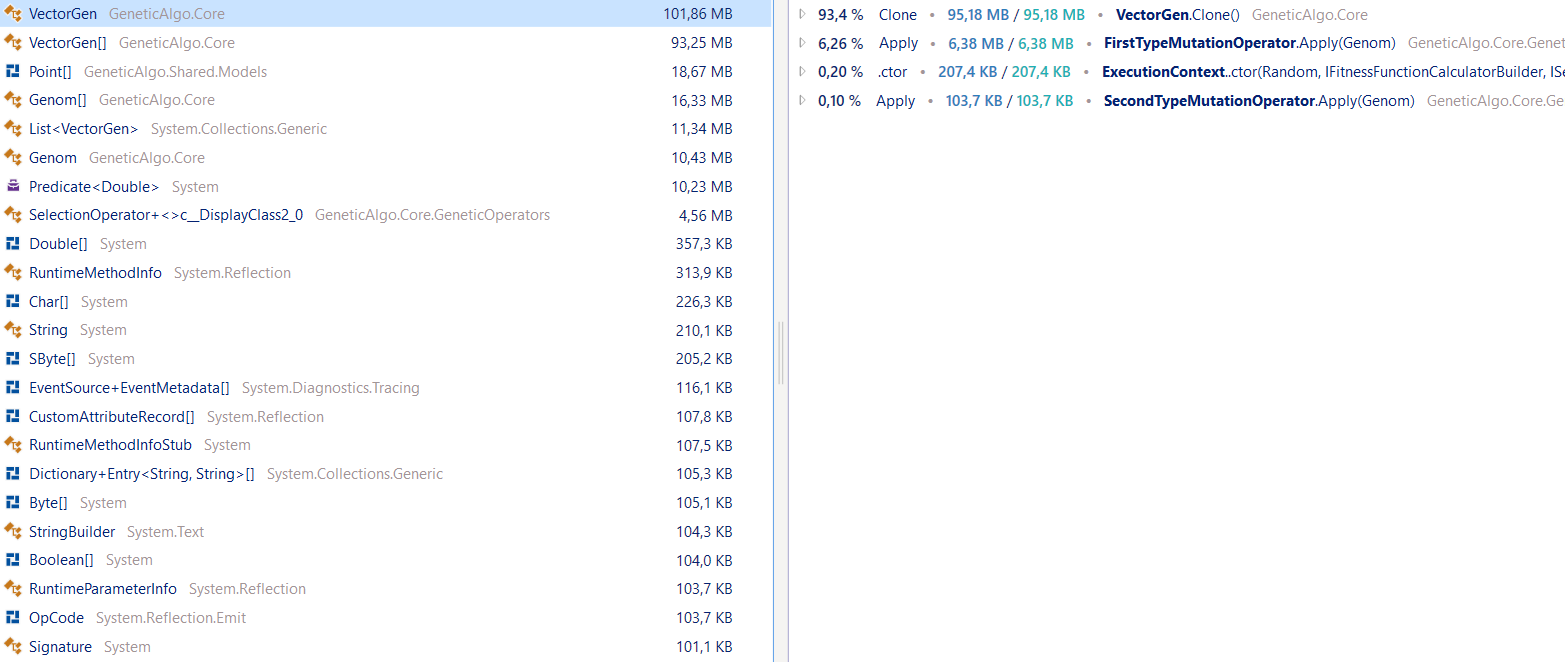
По-прежнему остаётся проблема пилообразности графика выделения памяти:



Проблема может возникать из-за выделения на каждой итерации новых экземпляров List.

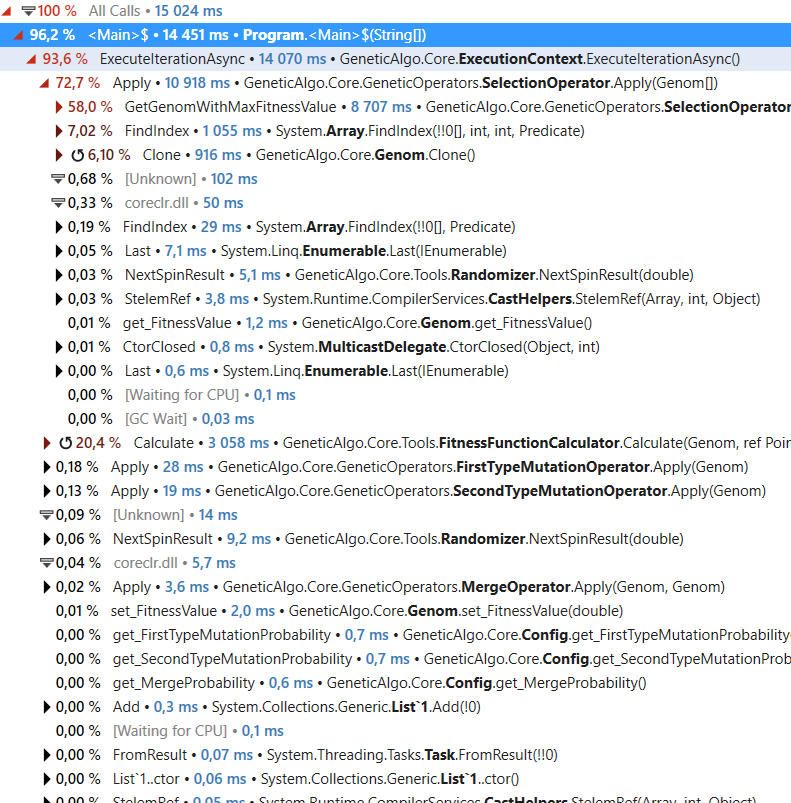


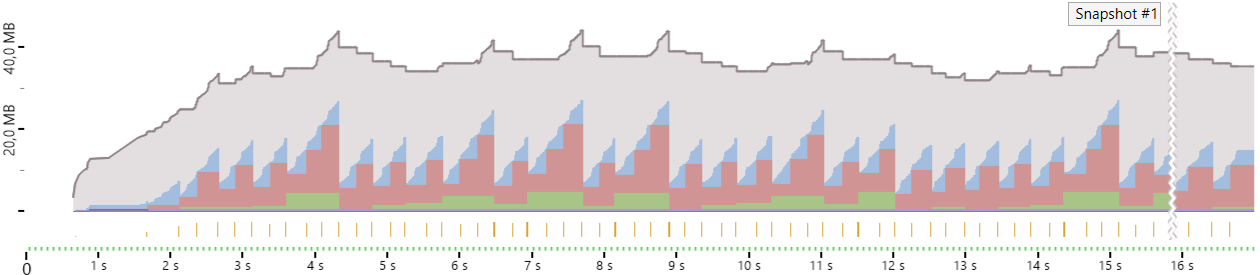
Заменим клонирование при мутациях на изменение исходного массива, в итоге получаем оптимизацию ещё 90 Мб (около 25%) памяти:

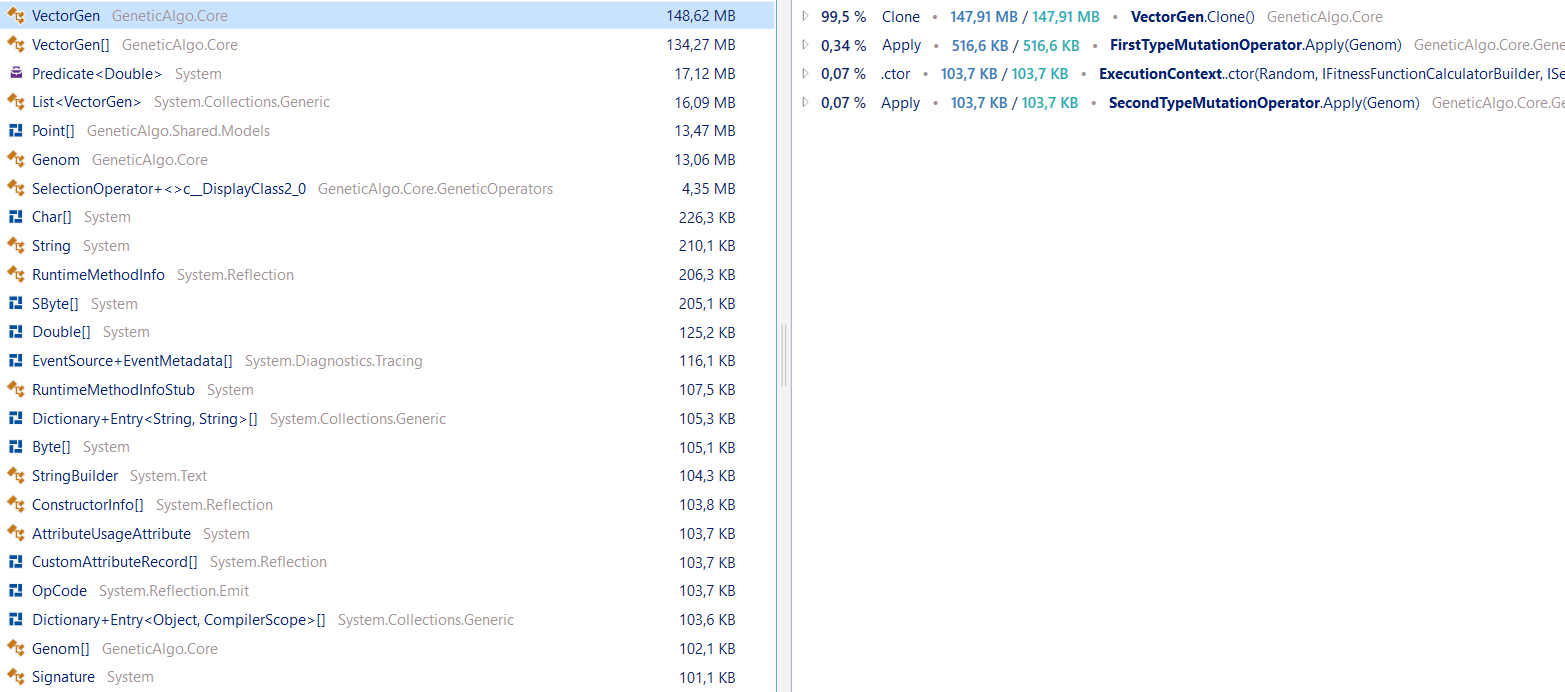


Оптимизация 3:

Так же существует потенциальная проблема с расширением List методом Add. Так как размеры массивов известны заранее, будем использовать Array вместо List, это даёт небольшое ускорение и устраняет данную проблему, а также ускоряет время выполнения ещё на 25%:

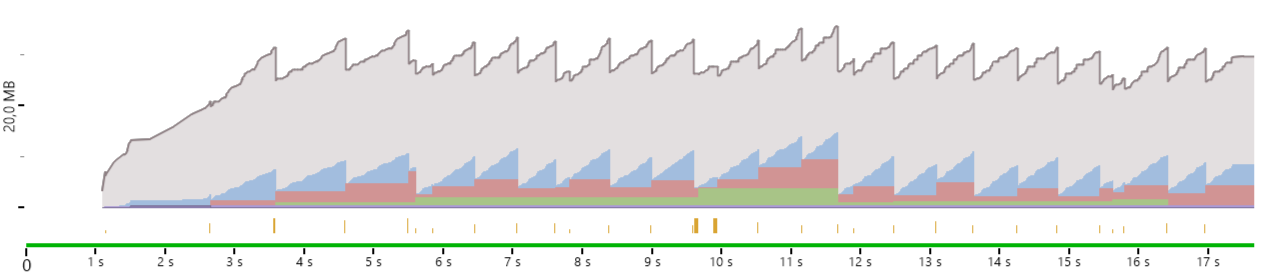


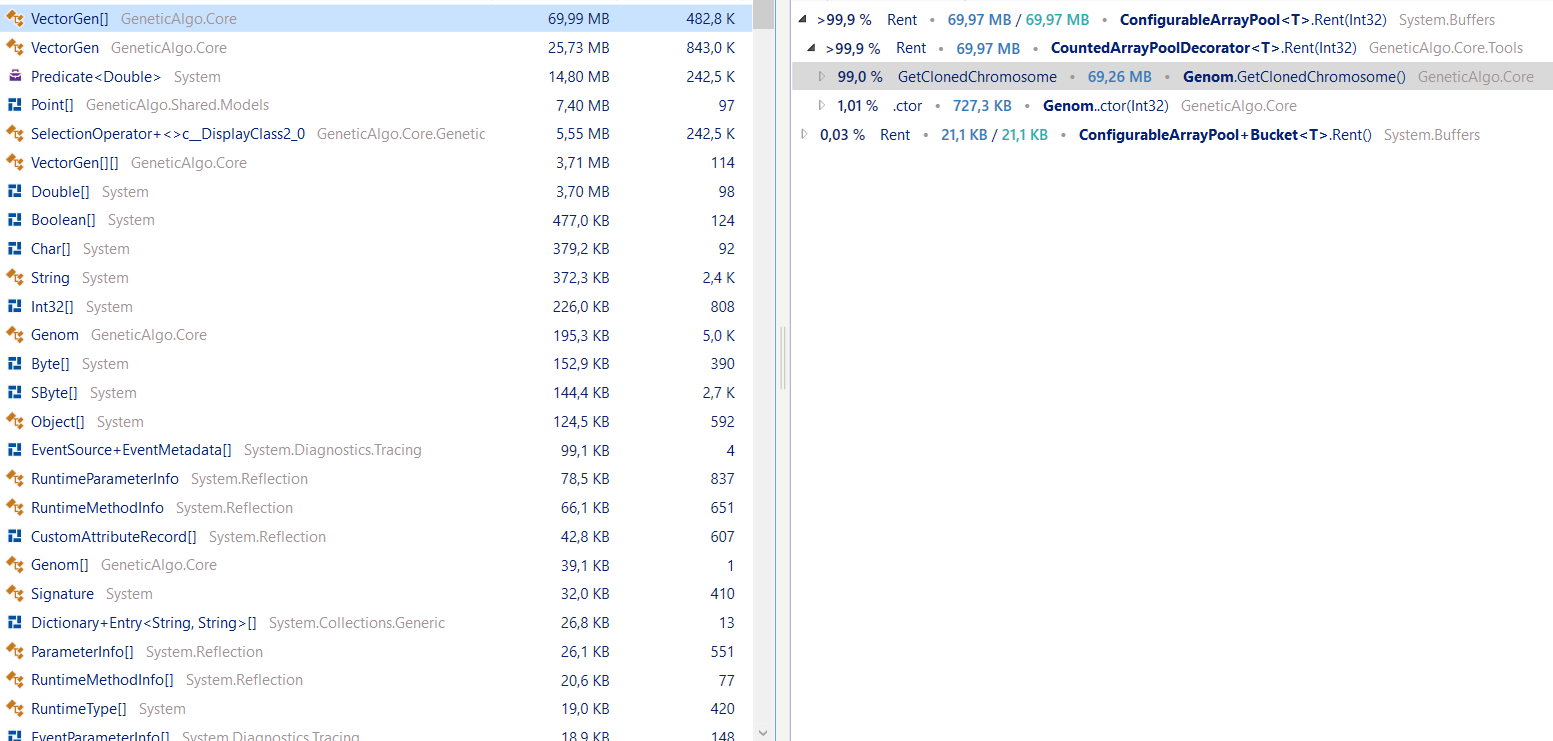




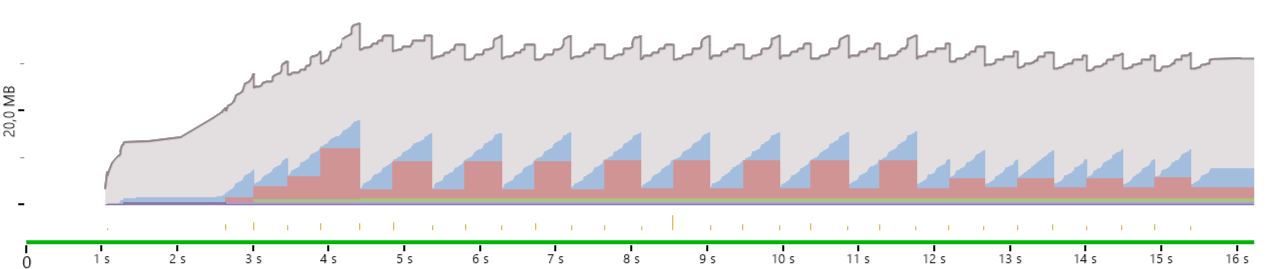
Оптимизация 4:

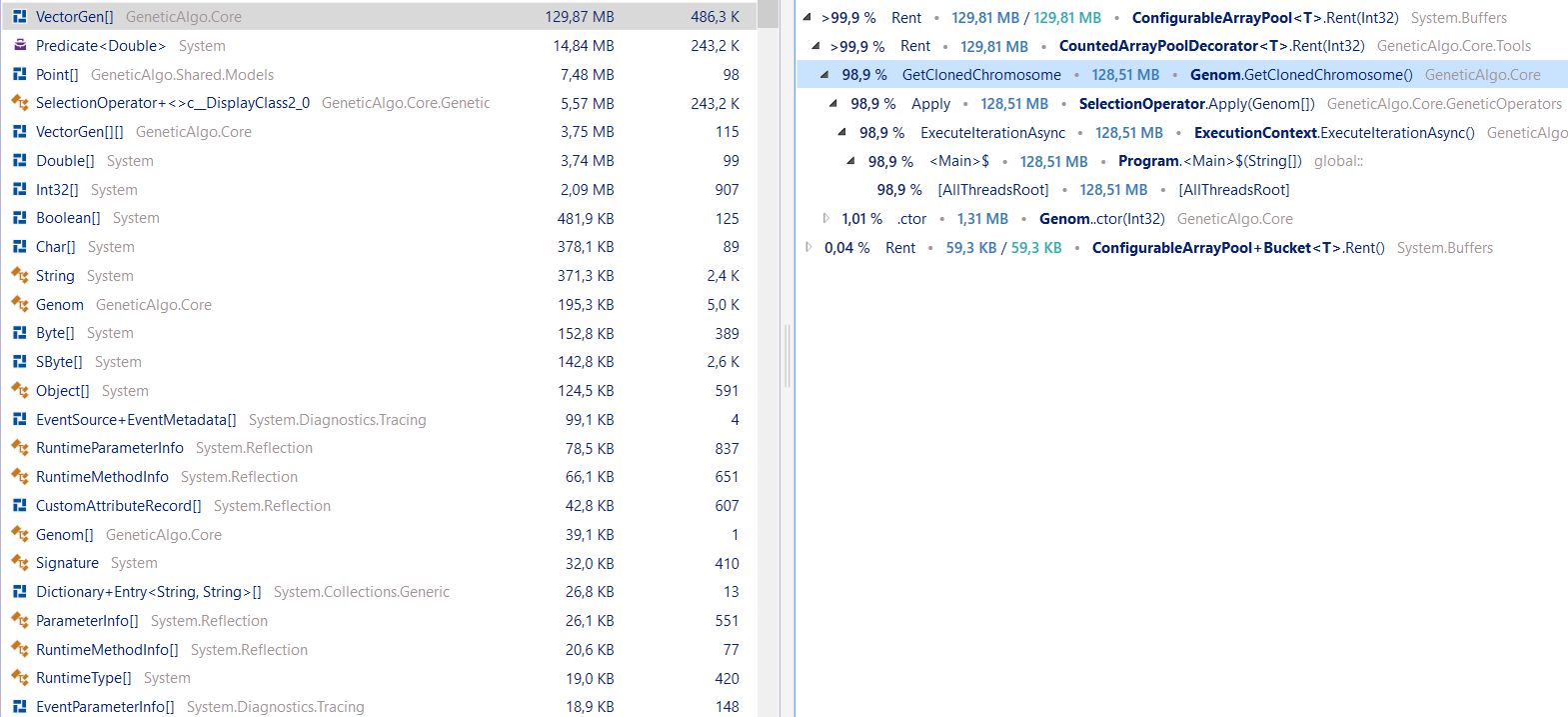
Заметим, что выделяется огромное количество объектов VectorGen[]. Они создаются на каждой итерации, при этом объекты с предыдущей итерации становятся ненужными. Используем ArrayPool для переиспользования этих объектов:



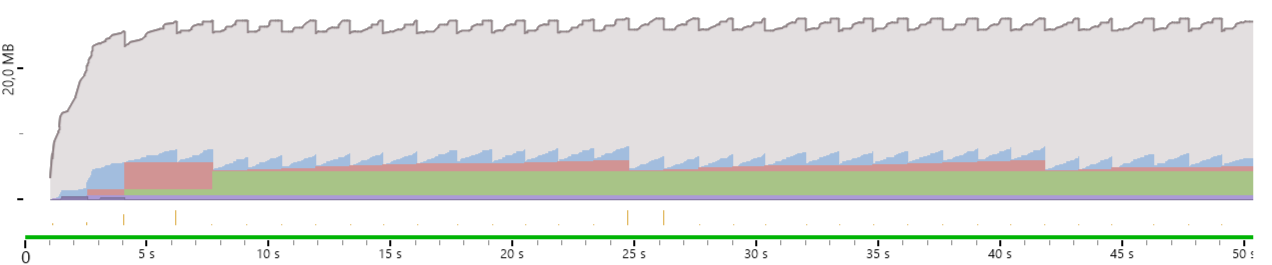


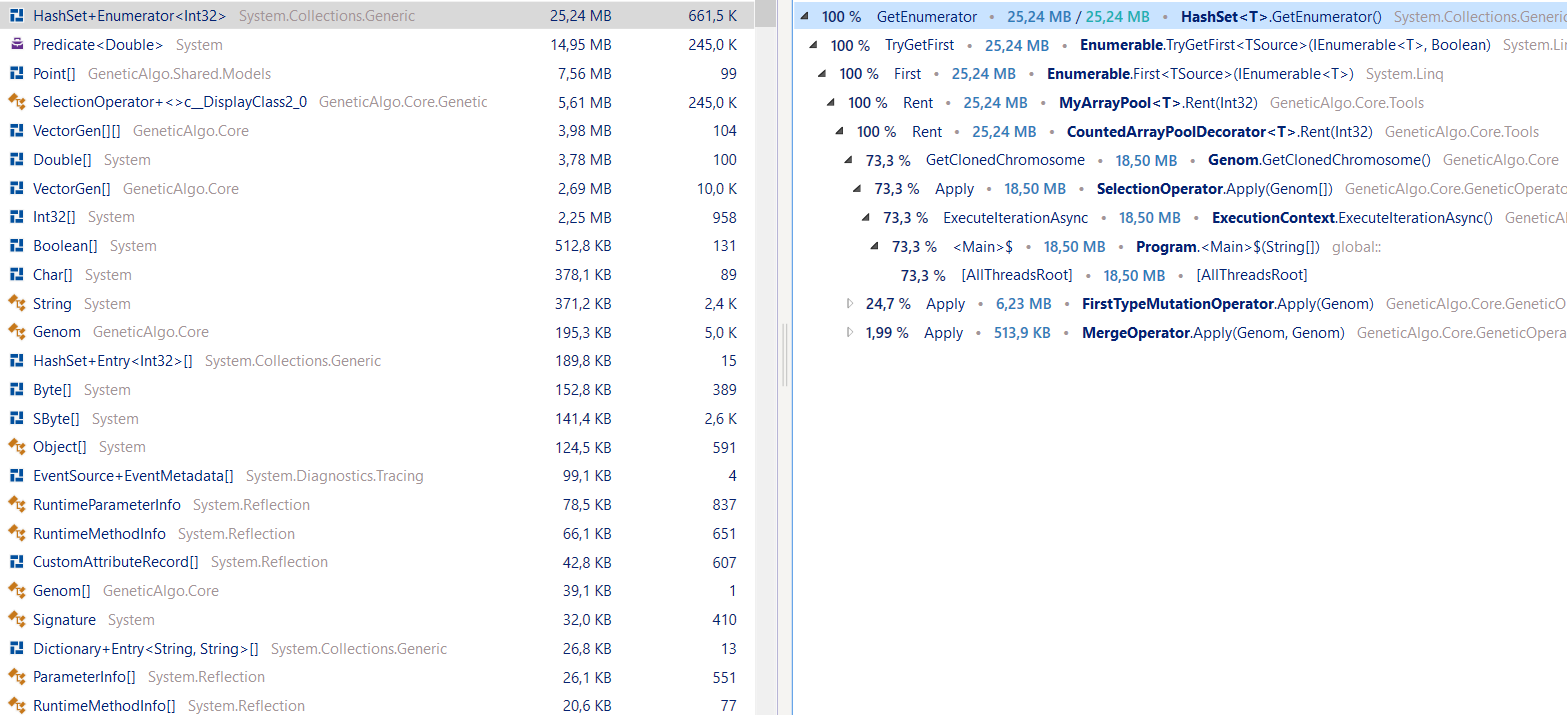
VectorGen – класс, сделаем его структурой:





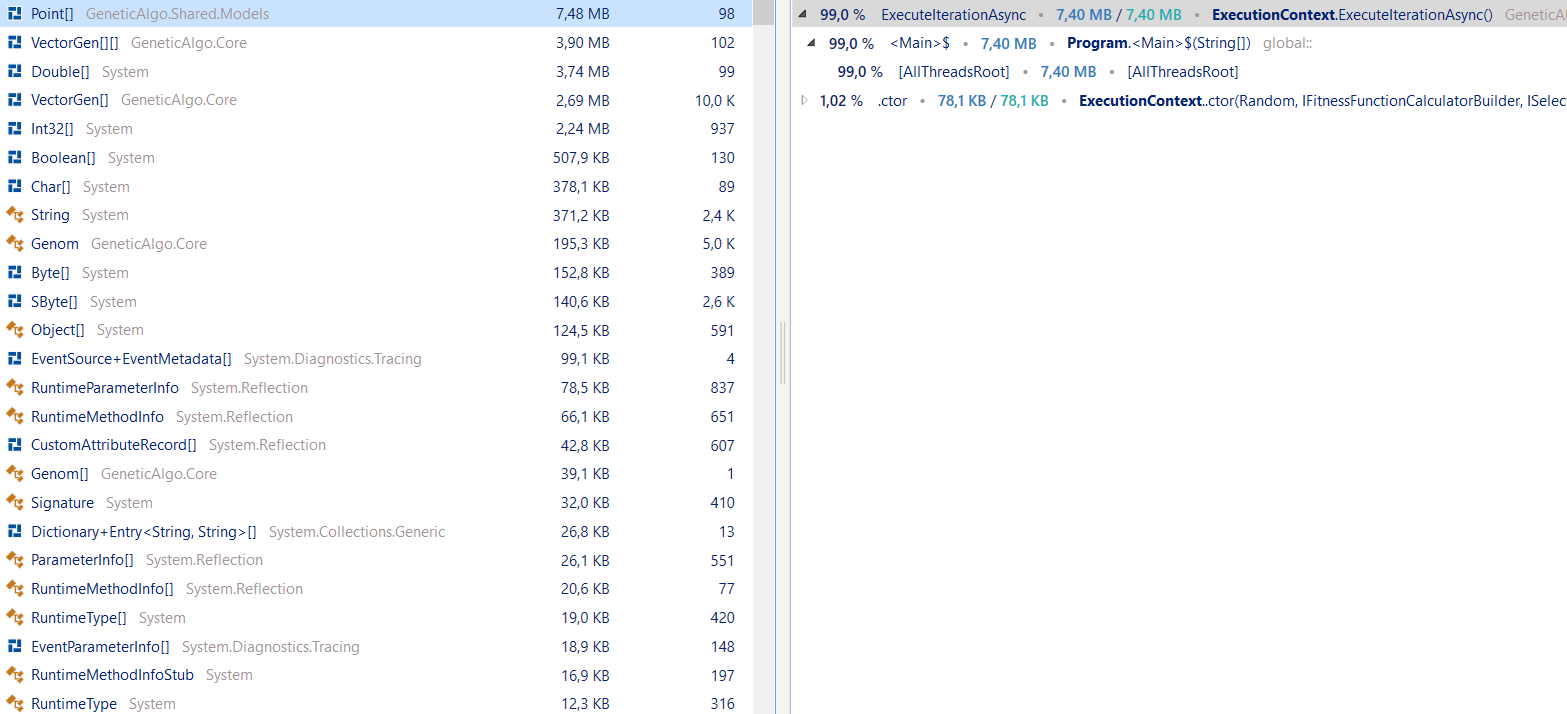
Встроенный ArrayPool всё ещё создаёт много объектов, напишем для него свою реализацию:





Оптимизация 5:

Теперь создаётся только необходимое количество объектов VectorGen[]. Но появилось много объектов HashSet+Enumerator<Int32>, связанных с вызовом функции First у HashSet, и объектов Predicate<Double>, связанных с лямбда функцией x => x > spinResult при поиске в массиве с помощью Array.FindIndex. Заменим HashSet на Queue и вынесем предикат, содержащий условие на x и переменную spinResult как поля класса:

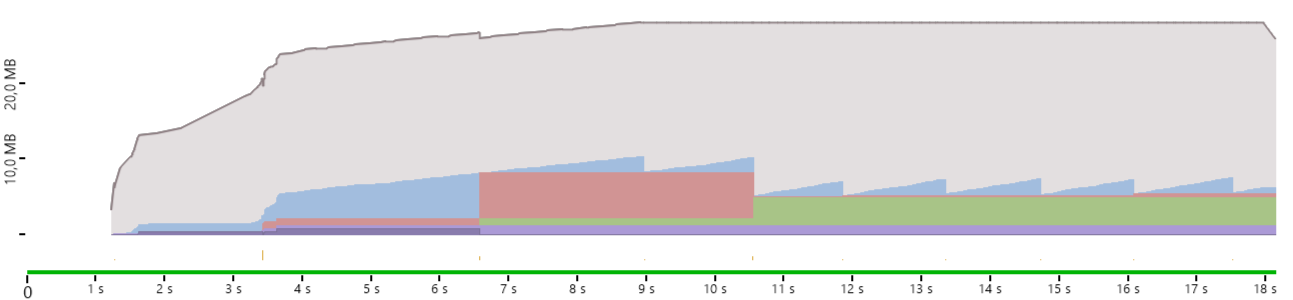


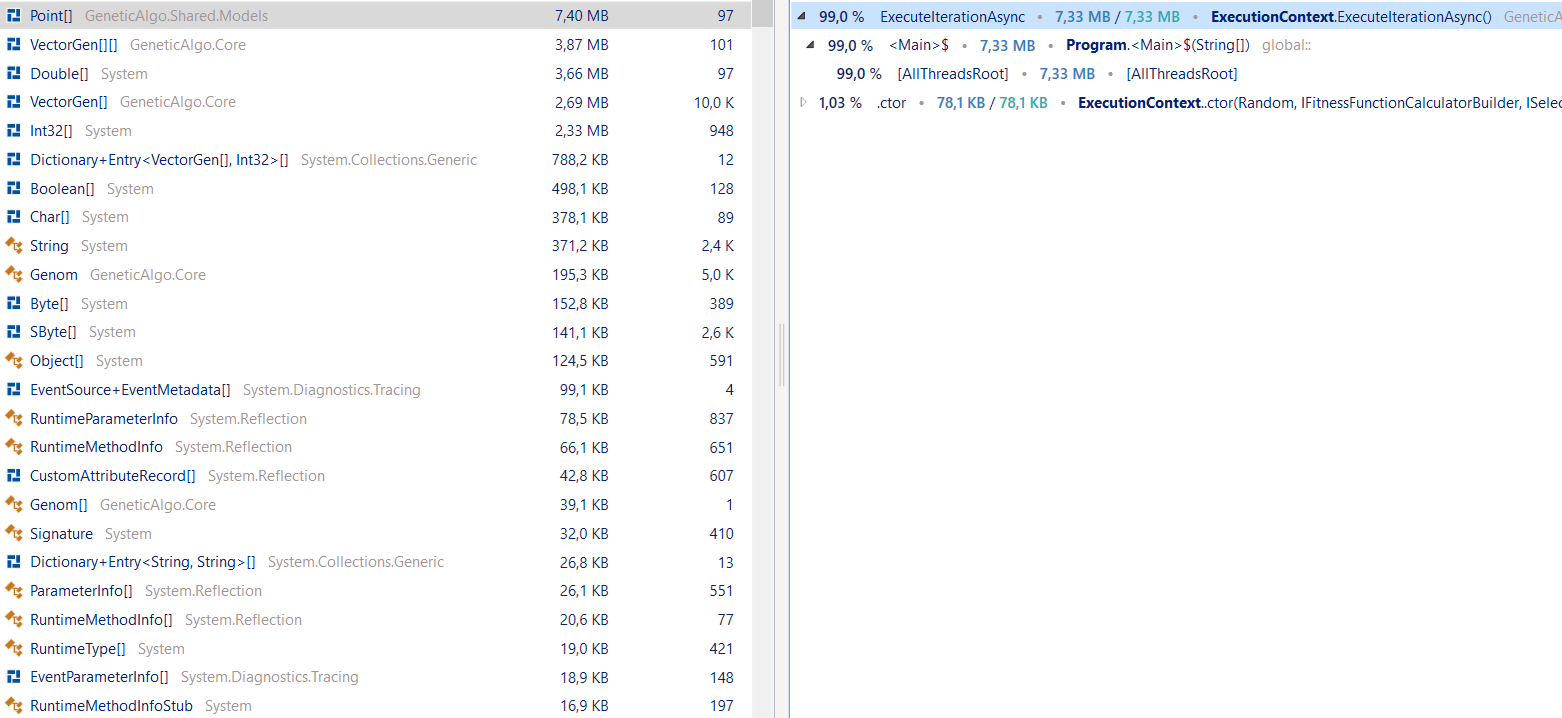
Это разрешает проблемы с памятью, но можно заметить, что произошло резкое увеличение времени работы:



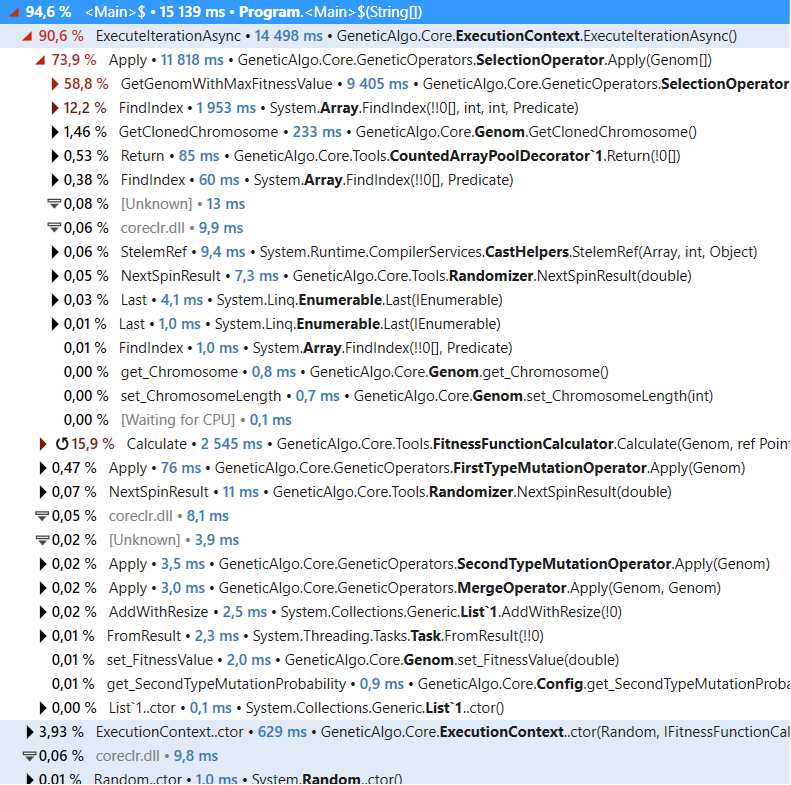
50% всего времени занимает вызов метода Return объекта MyArrayPool. Это связано с линейным поиском массива в пуле при его возврате. Добавим в MyArrayPool поле Dictionary<T[], int>, в котором будем хранить индексы для всех выделенных массивов из пула, после этого операция Return будет выполняться за , как и операция Rent.

Итоговые замеры dotMemory и dotTrace:





Память выделяется только под необходимое число объектов



Большую часть времени занимает поиск максимума в массиве, поэтому дальнейшие оптимизации бессмысленны либо требуют значительных изменений в логике работы.

В результате удалось сократить объём выделяемой памяти с до , а время выполнения с до .