Отчет: Анализ и улучшение метода ConvertToList для HashSet

1. Выявленные недостатки исходной реализации

При анализе метода ConvertToList для HashSet<T> обнаружены следующие проблемы:

1. *Неэффективное использование памяти:*
   Создание списка без предварительного указания емкости (Capacity) приводит к многократным переаллокациям внутреннего массива при добавлении элементов. Для 3 млн элементов это вызовет 22 переаллокации.
2. *Высокие накладные расходы:*
   Суммарно будет скопировано ~6 млрд элементов (в процессе переаллокаций), что негативно скажется на производительности.
3. *Нагрузка на GC:*
   Временные массивы от переаллокаций увеличат давление на сборщик мусора, вызывая паузы.

2. Решение проблемы

Вместо создания нового списка предлагаю использовать обертку, которая предоставляет доступ к данным HashSet как к списку без копирования элементов.

Принцип работы: Элементы получаются через ElementAt по требованию.
Для HashSet это O(1) в среднем (используется внутреннее индексирование). Данные не копируются — обертка использует исходный HashSet (Zero-Copy). Гарантирует, что список не изменяется после создания.

Подходит для сценариев, где список используется для чтения (без модификаций), и обработки больших данных (экономия памяти).

Не подходит, если требуется частое обращение по индексу и когда нужна независимая копия данных.

Решим и эту проблему: для частых обращений по индексу можно добавить кеширование.
Итоговый изменённый код по ссылке https://github.com/sanvvvu/Vspomogatelni-metod

3. Итог

Предложенная реализация решает проблемы исходного метода:

1. Устраняет переаллокации и нагрузку на GC.
2. Снижает потребление памяти с 3n до O(1).
3. Сохраняет совместимость с IReadOnlyList<T>..