|  |  |
| --- | --- |
| Размер | n |
| Основная операция | сравнение |
| Зависимость только от размера входных данных | нет |
| Сумма основных операций |  |
| Упрощение формулы | ,то =(n-1)n ≈∈Θ() |

IsReflectMatrix

MinEl

|  |  |
| --- | --- |
| Размер | n |
| Основная операция | сравнение |
| Зависимость только от размера входных данных | да |
| Сумма основных операций | A(n)=A(n-1)+1=…=A(n-n)+n≈n ∈Θ(n) |
| Начальные условия | A(0)=0,  А(1)=1 |

MinEl2

|  |  |
| --- | --- |
| Размер | n |
| Основная операция | сравнение |
| Зависимость только от размера входных данных | да |
| Сумма основных операций | A(n) = 2A(n/2)+1=…=A(1)+n≈n ∈Θ(n), где n > 1 |
| Начальные условия | A(0)=0,  А(1)=1 |
| По дереву рекурсивных вызовов могу предположить что рекуррентная функция | Fn(n)=2n-1 |

Скорость алгоритма будет зависеть от машины на которой он выполняется. И текущая реализация одна из самых быстрых. Можно рассмотреть следующие варианты улучшения алгоритма:

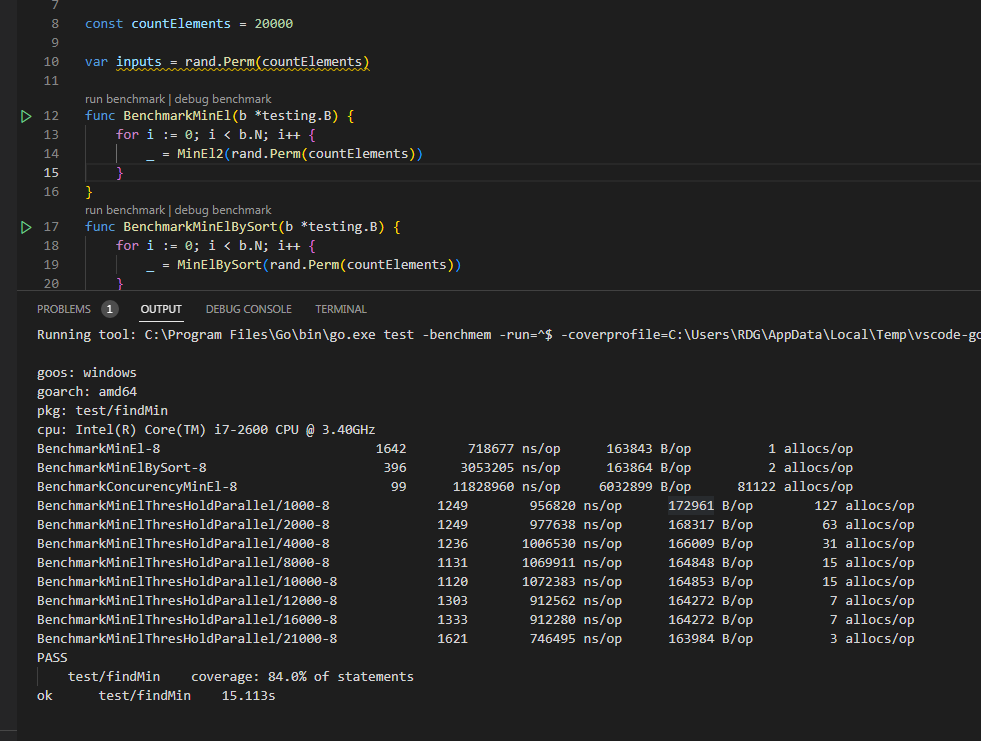
- считать половинки среза конкурентно, но рабочая нагрузка основной операции(сравнение) слишком мала – время на создание горутин и переключения между контекстом выше. Самый медленный вариант ;

-считать половинки среза конкурентно с порогом(если количество элементов в половинке меньше этого значения, то будем обрабатывать последовательно).

Второй вариант приближен по скорости к MinEl2, однако требуется подбора порога в зависимости от машины. На 4х ядернике Intel Core i7-2600 такой алгоритм проигрывает последовательному в два раза.

Сложность алгоритма не изменяется от изменения скорости.

В main\_test.go бэнчмарки для разных вариантов алгоритма + для разных порогов, а так же алгоритм с сортировкой среза и возвращением первого элемента.



На 16 ядерном AMD Ryzen картина схожа, отличается значение погрогов.

