Национальный Исследовательский Университет

«Московский Энергетический Институт»

Кафедра прикладной математики и искусственного интеллекта

Тема: Мониторинг производительности .NET приложения с помощью JetBrains dotTrace.

Студент: Ростовых Александра

Москва 2021

**Цель работы**

Научиться получать точную информацию об узких местах в производительности приложений, созданных на основе платформы .NET Framework. Получить навыки профилирования приложения в нескольких режимах, включая tracing (на основе подсчета числа вызовов), sampling (на основе подсчета времени исполнения) и построчный режим (детальный анализ производительности).

**1. Подготовить тестируемое приложение**

* Разработать с помощью Visual Studio консольное приложение
* Реализовать в приложении два алгоритма сортировки (быструю сортировку и сортировку пузырьком)

Быстрая сортировка:

static int Partition<T>(T[] m, int a, int b) where T : IComparable

{

int i = a;

for (int j = a; j <= b; j++) // просматриваем с a по b

{

if (m[j].CompareTo(m[b]) <= 0) // если элемент m[j] не превосходит m[b],

{

(m[i], m[j]) = (m[j], m[i]); // меняем местами m[j] и m[a], m[a+1], m[a+2] и так далее..

i++; // таким образом последний обмен: m[b] и m[i], после чего i++

}

}

return i - 1; // в индексе i хранится <новая позиция элемента m[b]> + 1

}

static void Quicksort<T>(T[] m, int a, int b) where T : IComparable// a - начало подмножества, b - конец

{ // для первого вызова: a = 0, b = <элементов в массиве> - 1

if (a >= b) return;

int c = Partition(m, a, b);

Quicksort(m, a, c - 1);

Quicksort(m, c + 1, b);

}

Сортировка пузырьком:

static void BubbleSort1<T>(T[] mas) where T : IComparable

{

for (int i = 0; i < mas.Length - 1; i++)

{

for (int j = i + 1; j < mas.Length - 1; j++)

{

if (mas[i].CompareTo(mas[j]) > 0)

{

(mas[i], mas[j]) = (mas[j], mas[i]);

}

}

}

}

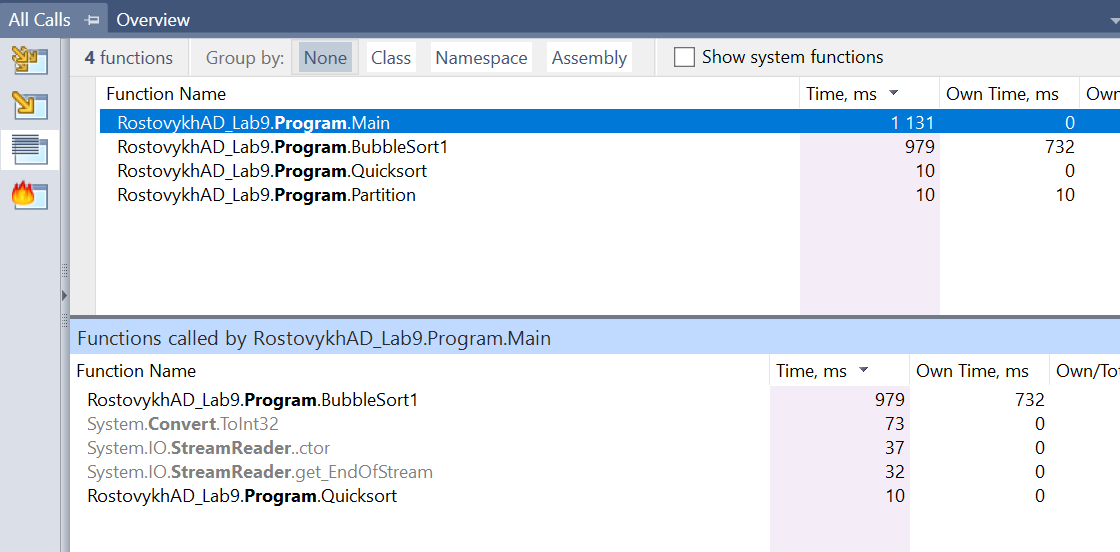
* Сортируемые данные должны загружаться из файла
* Создать несколько наборов несортированных тестовых данных (в т.ч. данные размером 1Кб, 2Кб)

Созданы файлы:

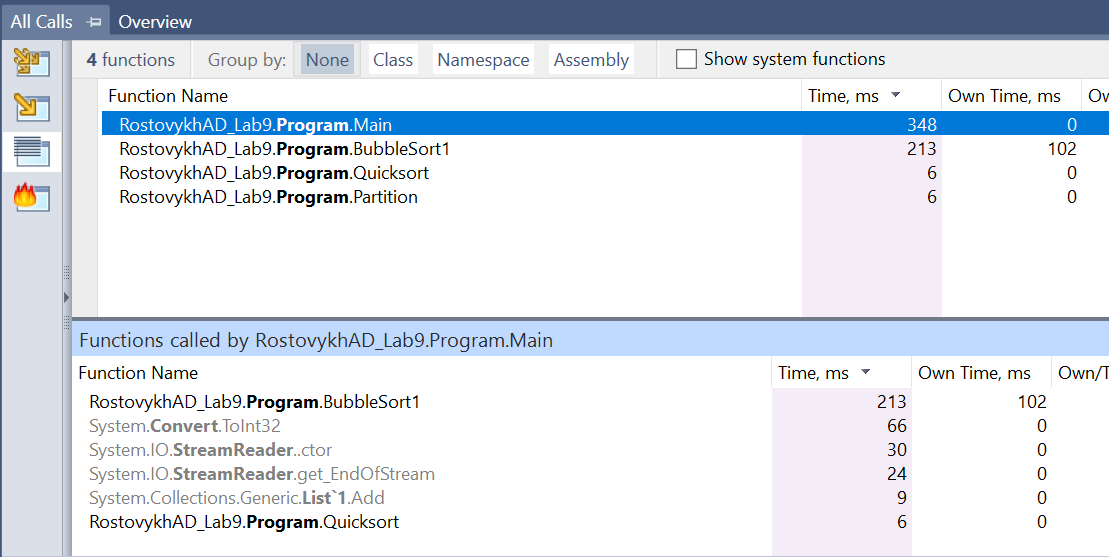
1. Файл с 7000 несортированных числовых данных (объем 40,2 Кб)
2. Файл с 3000 несортированных числовых данных (объем 17,2 Кб)
3. Файл с 1000 строковых несортированных данных (объем 977 Кб)

**2. Запустить приложение в режиме профилирования sampling. Определить наименее производительные функции.**

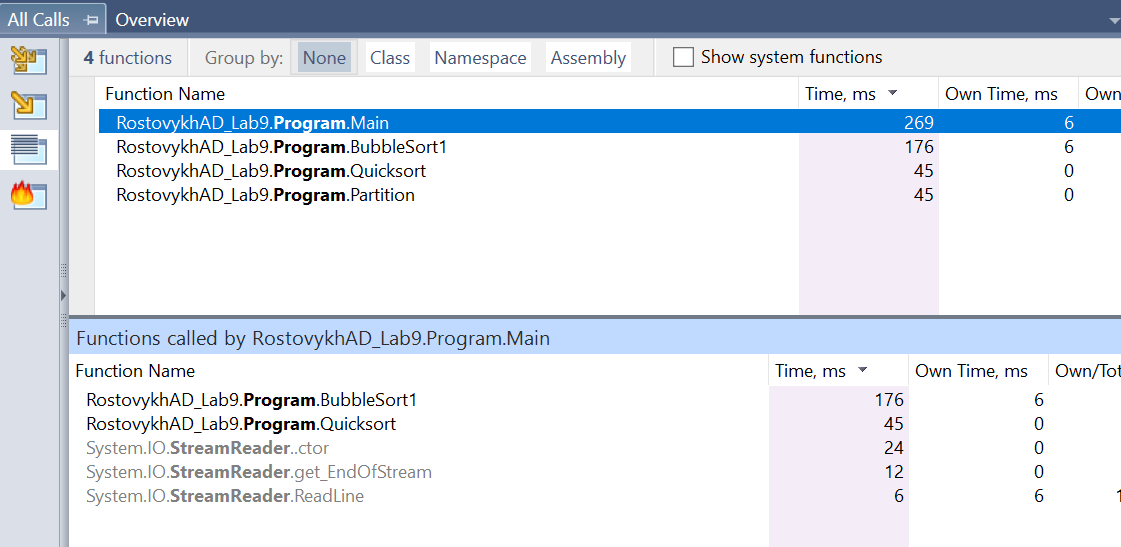
*Файл с 7000 числовых данных:*



*Файл с 3000 числовых данных:*



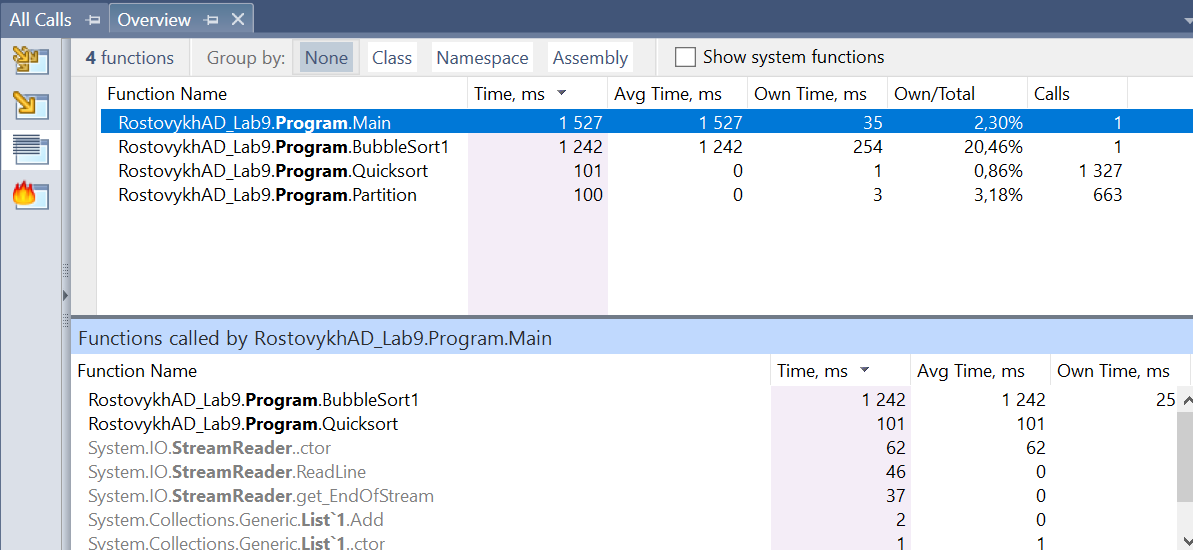
*Файл с 1000 строковых данных:*



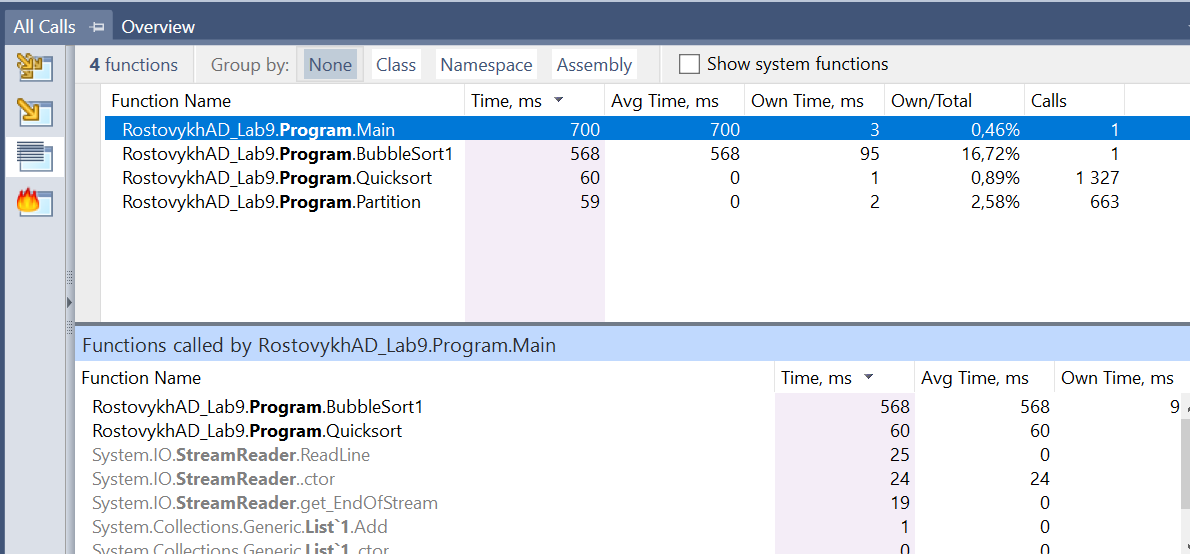
Видим, что сортировка пузырьком работает во много раз медленнее, чем быстрая сортировка во всех трех случаях.

**3. Запустить приложение в режиме профилирования tracing. Получить результат. Определить узкие места в реализации программы.**

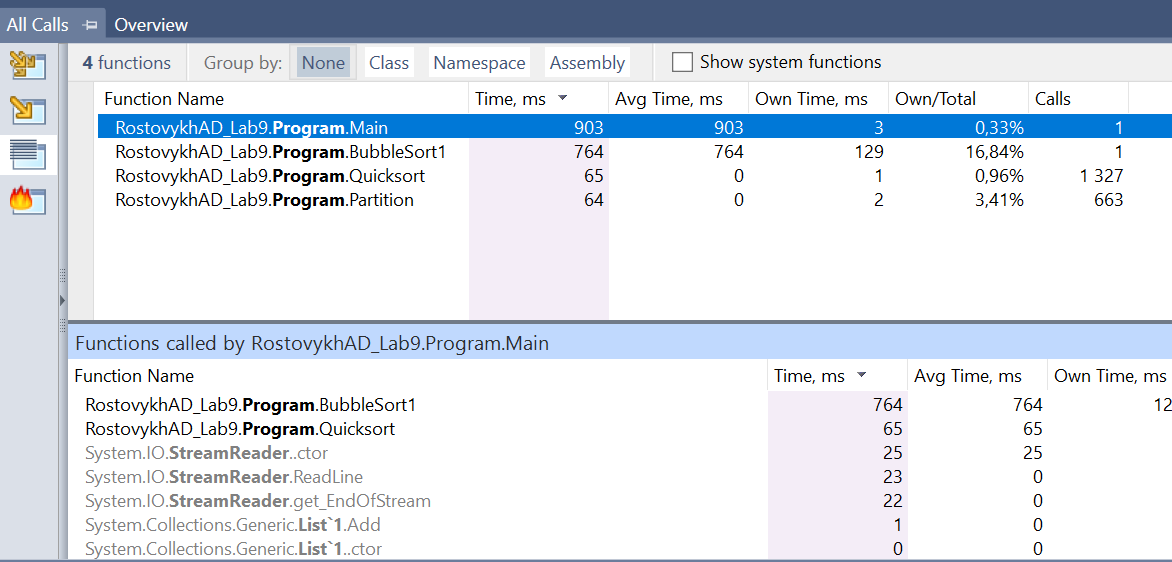
*Файл с 7000 числовых данных:*



*Файл с 3000 числовых данных:*



*Файл с 1000 строковых данных:*

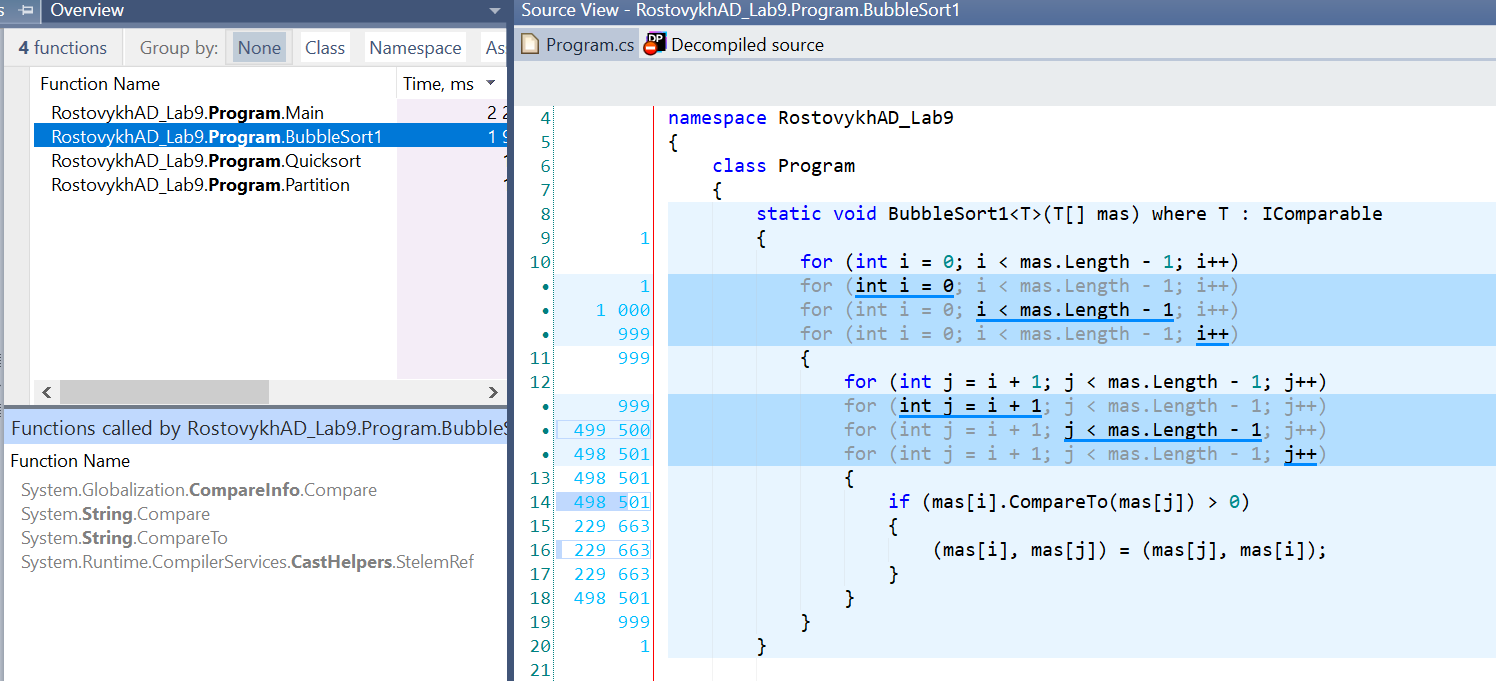


Наблюдаем в последнем столбце количество вызовов каждой функции. Так как в быстрой сортировке мы рекурсивно вызываем эту же функцию, видим, что количество вызовов Quicksort больше всего.

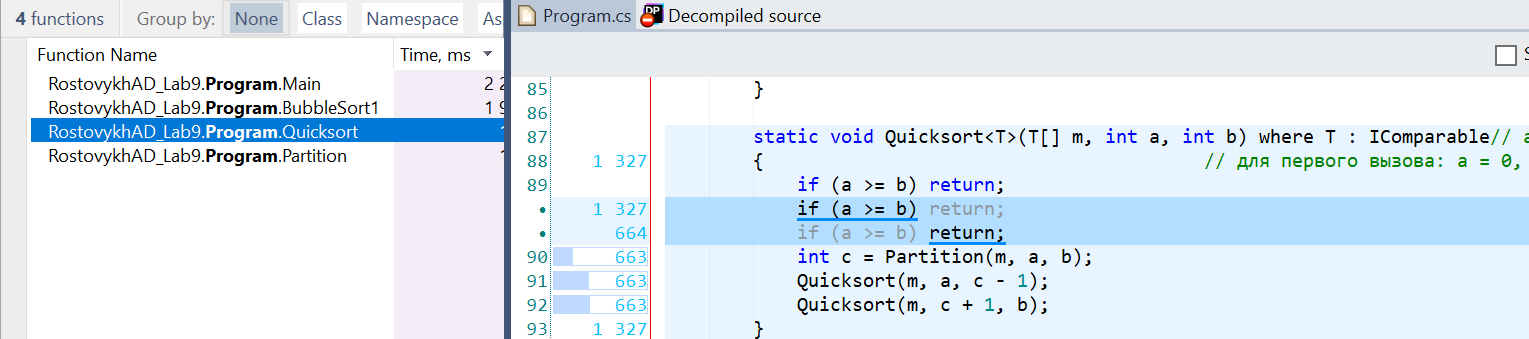
**4. Запустить приложение в режиме профилирования line by line. Получить информацию какие строки кода исполняются чаще всего.**

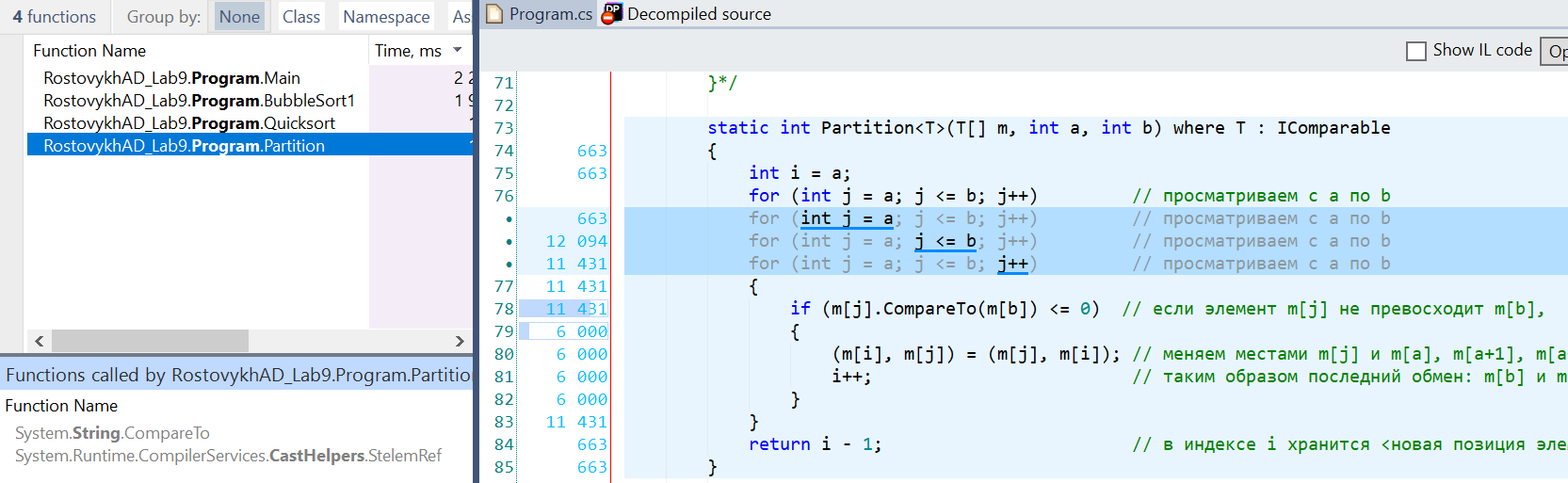
*Файл с 7000 числовых данных:*

Сортировка пузырьком:



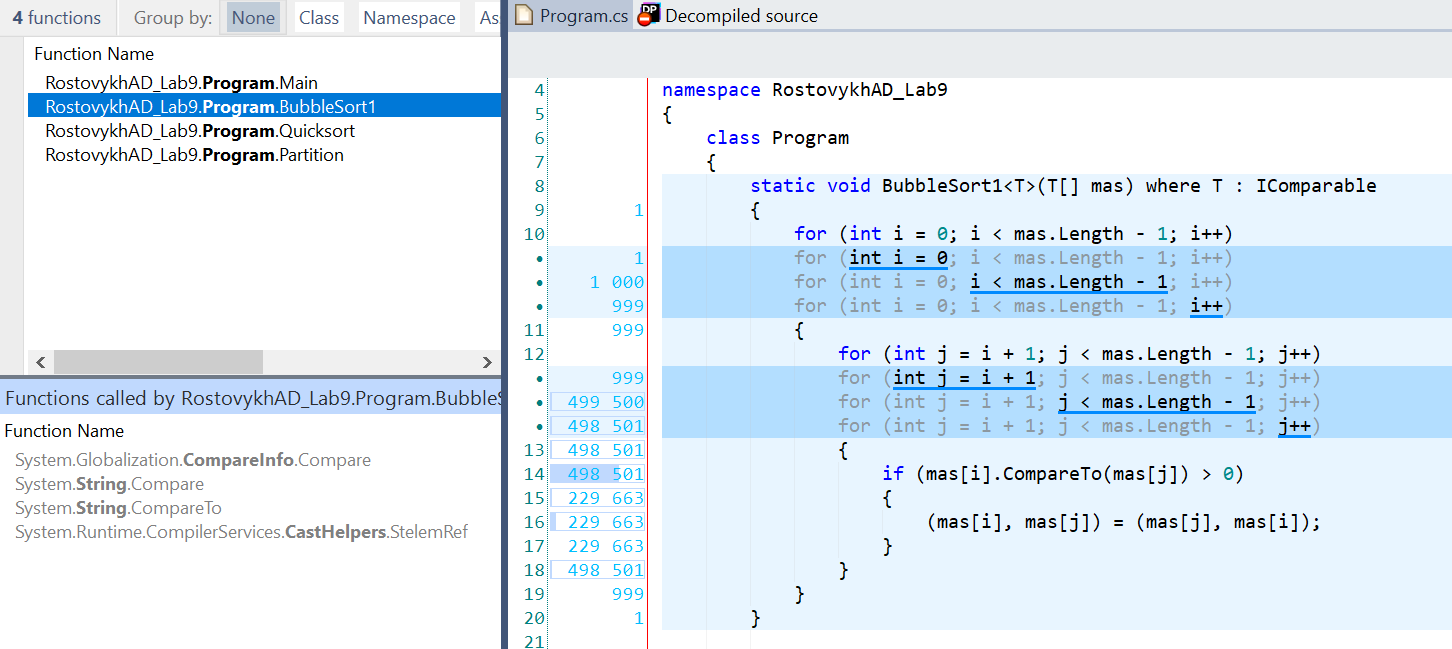
Функции для быстрой сортировки:



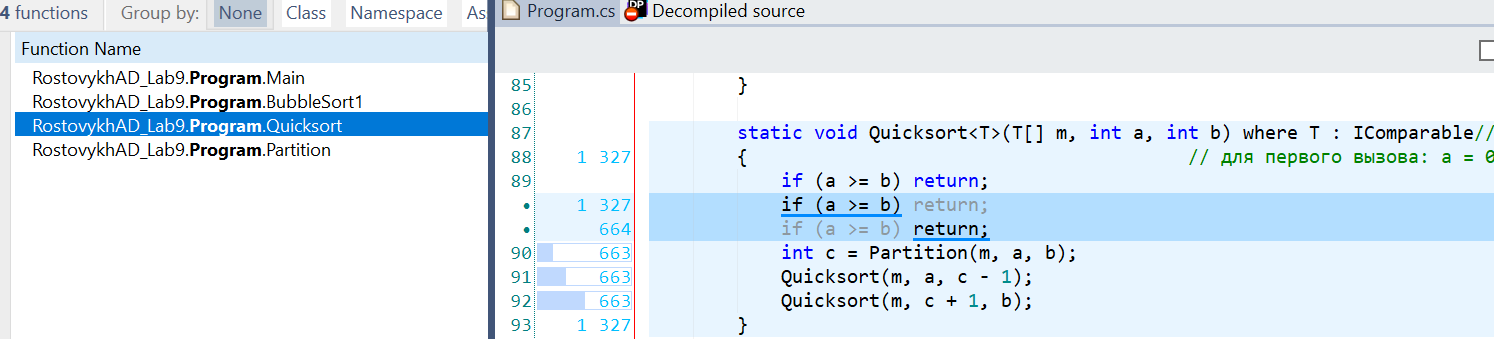


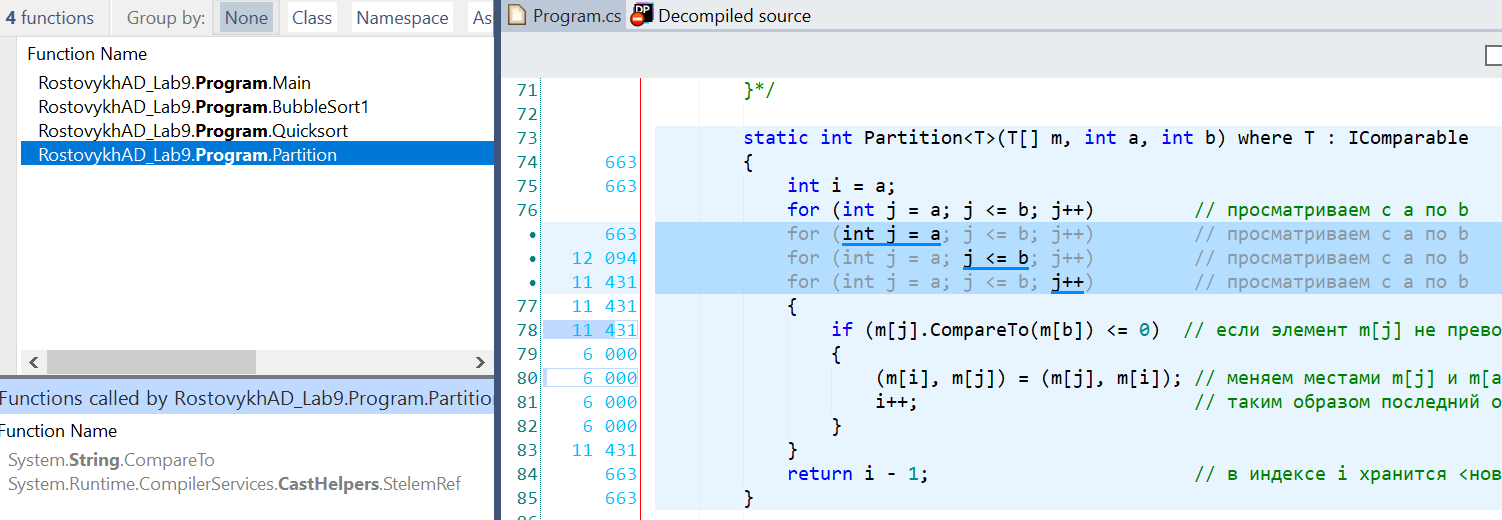
*Файл с 3000 числовых данных:*

Сортировка пузырьком:



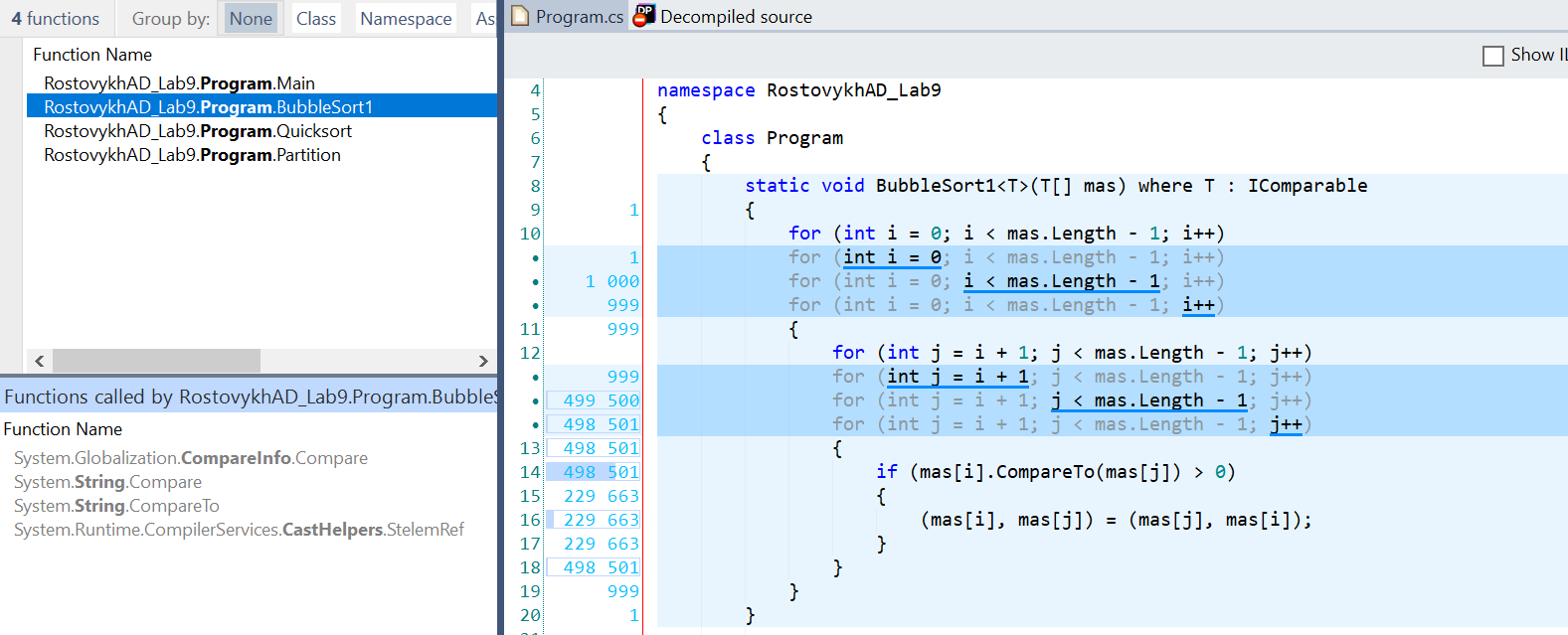
Функции для быстрой сортировки:



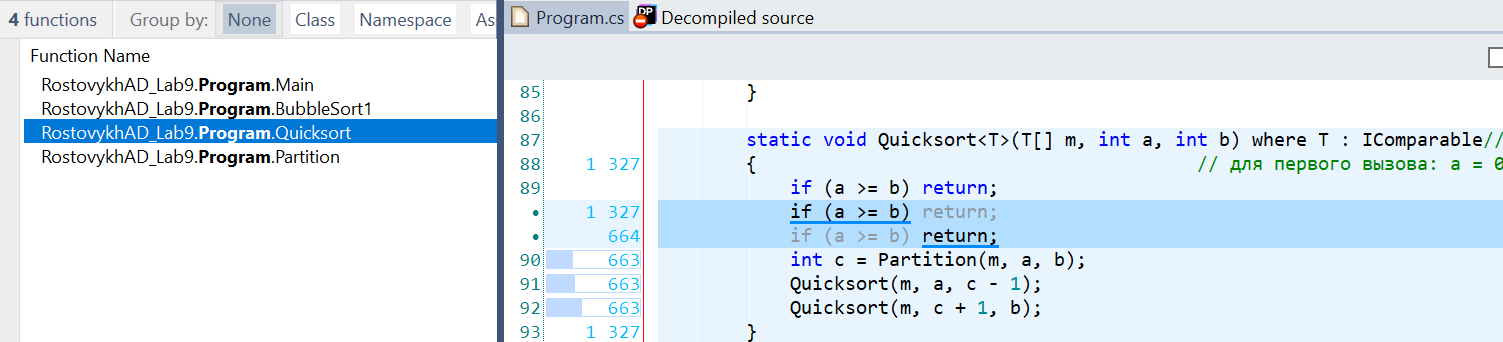


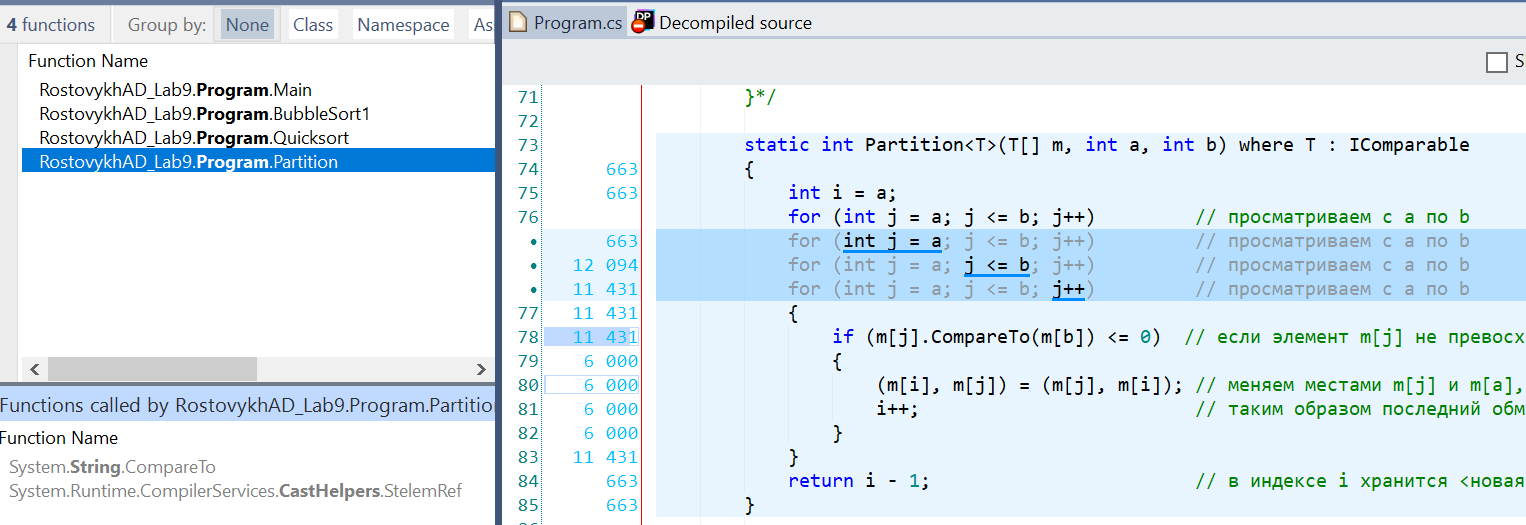
*Файл с 1000 строковых данных:*

Сортировка пузырьком:



Функции для быстрой сортировки:





**5. Используя полученную информацию улучшить реализацию и проверить это с помощью повтора шагов 2-4**

Попробуем изменить наши алгоритмы сортировки

Теперь в быстрой сортировке попробуем взять за опорный элемент середину массива

Общий алгоритм будет такой:

1. вводятся указатели *first* и *last* для обозначения начального и конечного элементов последовательности, а также опорный элемент *mid*;
2. вычисляется значение опорного элемента (*first*+*last*)/2, и заноситься в переменную *mid*;
3. указатель *first* смещается с шагом в 1 элемент к концу массива до тех пор, пока *Mas*[*first*]>*mid*. А указатель *last* смещается от конца массива к его началу, пока *Mas*[*last*]<*mid*;
4. каждые два найденных элемента меняются местами;
5. пункты 3 и 4 выполняются до тех пор, пока first<last.

После разбиения последовательности следует проверить условие на необходимость дальнейшего продолжения сортировки его частей.

static void Quicksort<T>(T[] mas, int a, int b) where T : IComparable

{

if (a>= b)

return;

int mid = Partition(mas, a, b);

Quicksort<T>(mas, a, mid);

Quicksort<T>(mas, mid + 1, b);

}

static int Partition<T>(T[] mas, int a, int b) where T : IComparable

{

T mid = mas[(a + b) / 2];

int i = a;

int j = b;

while (i <= j)

{

while (mas[i].CompareTo(mid) < 0) i++;

while (mas[j].CompareTo(mid) > 0) j--;

if (i >= j) break;

(mas[i], mas[j]) = (mas[j], mas[i]);

i++; j--;

}

return j;

}

И для сортировки пузырьком:

static void BubbleSort2<T>(T[] mas) where T : IComparable

{

int k = 0;

bool flag = false;

while (!flag)

{

flag = true;

for (int i = 0; i < mas.Length - 1 - k; i++)

{

if (mas[i].CompareTo(mas[i + 1]) > 0)

{

flag = false;

(mas[i], mas[i + 1]) = (mas[i + 1], mas[i]);

}

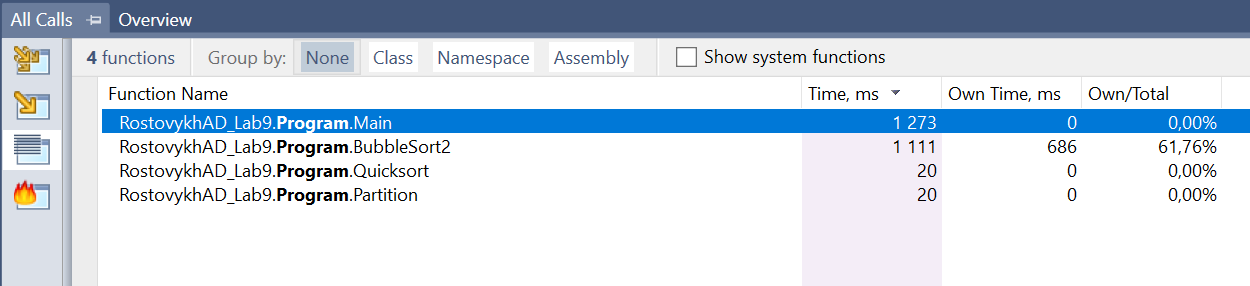
}

k++;

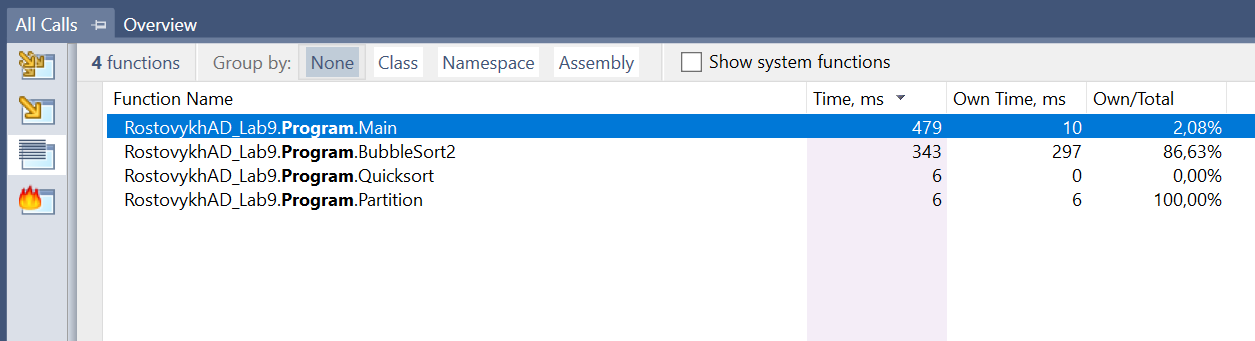
}

}

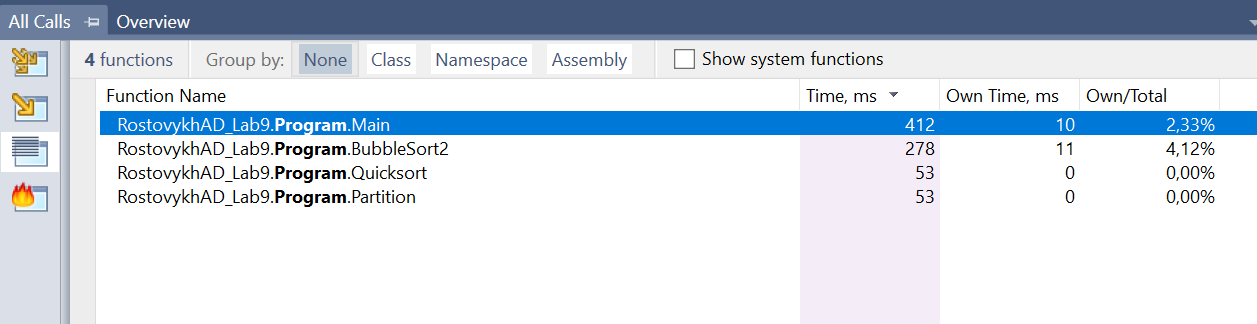
*Файл с 7000 числовых данных:*



*Файл с 3000 числовых данных:*



*Файл с 1000 строковых данных:*



Видим, что результаты не сильно изменились, возможно, даже стали немного хуже, так что можем сделать вывод, что первая реализация алгоритмов сортировки была достаточно успешной!