Формулировка первого задания:

Задание 1:

1.Вычислить порядок сложности программы (О-символику).

2.Оценить время выполнения программы и кода, выполняющего перемножение матриц, используя функции библиотеки time.h для матриц размерами от 100, 200, 400, 1000, 2000, 4000, 10000.

3.Построить график зависимости времени выполнения программы от размера матриц и сравнить полученный результат с теоретической оценкой.

Для выполнения задания необходимо использовать следующие команды (функции):

- Функция clock() - она позволяет узнать время прошедшее с запуска программы, при этом время будет дано в тактах.

- Константа CLOCKS\_PER\_SEC — определяющая количество тактов, производящихся процессором, в секундах.

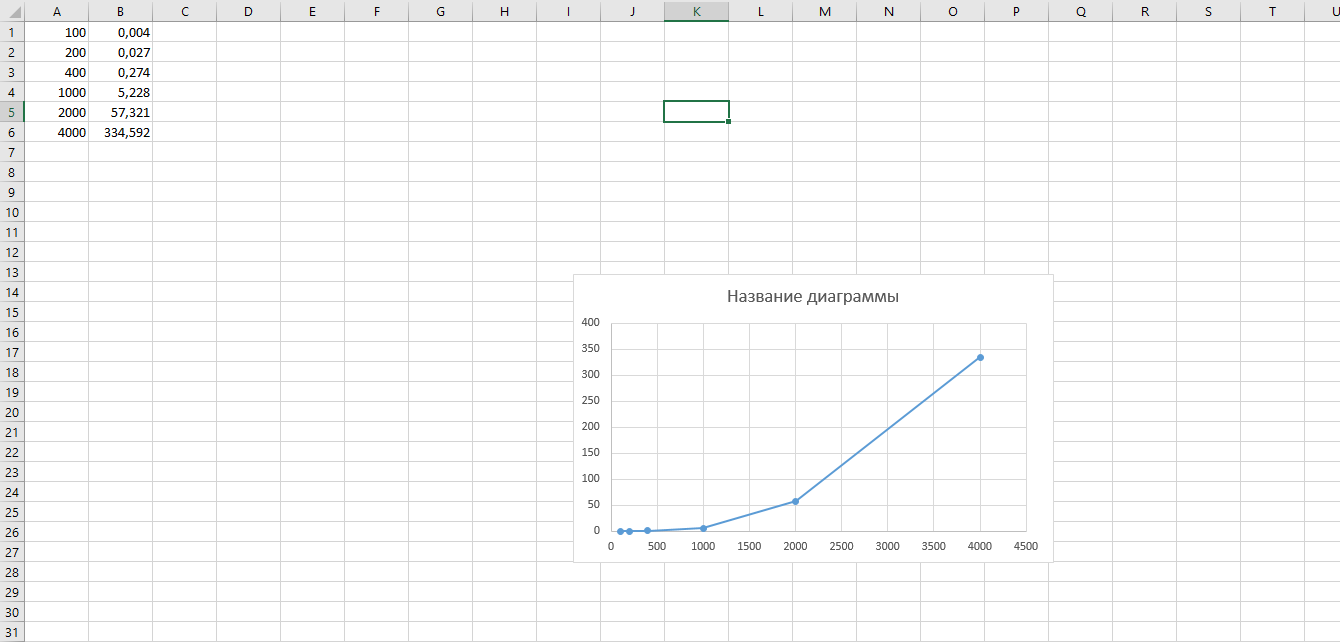
При этом если найти разницу между двумя значениями, данными функцией clock(), то можно узнать сколько времени прошло в секундах между двумя этими замерами. То есть в начале замера запомнить время в переменную, например, starttime, с помощью функции clock(). А в конце замера взять разницу и разделить её на константу CLOCKS\_PER\_SEC, запомнив при этом в какую нибудь переменную, например diff, тогда в этой переменной будет находится время, прошедшее с первого замера до второго, в секундах. В общем это действие будет выглядеть следующим образом:

float diff = ((clock()) - starttime)/float(CLOCKS\_PER\_SEC);

Теперь по заданию, нужно определить, за какое время будет выполнен алгоритм, производящий умножение матриц, с разным объёмом данных. Для этого из задания был скопирован весь код этого алгоритма, но были убраны константно-заданные размеры матриц — двумерных массивов. Вместо этого в цикле for перебираются значения из заранее созданного массива. Так же каждый раз при увеличении размера матрицы, для трёх массивов выделяется некоторое количество памяти, в зависимости от объёма.

В качестве начальной точки замера времени алгоритма был выбран цикл, отвечающий именно на перемножение матриц. Это важно, так как в алгоритме так же присутствует цикл, производящий заполнение массива случайными числами.

По полученным данным был построен график:



Задание 2.

Формулировка задания:

1.Оценить время работы каждого из реализованных алгоритмов на случайном наборе значений массива.

2.Оценить время работы каждого из реализованных алгоритмов на массиве, представляющем собой возрастающую последовательность чисел.

3.Оценить время работы каждого из реализованных алгоритмов на массиве, представляющем собой убывающую последовательность чисел.

4.Оценить время работы каждого из реализованных алгоритмов на массиве, одна половина которого представляет собой возрастающую последовательность чисел, а вторая, – убывающую.

5.Оценить время работы стандартной функции qsort, реализующей алгоритм быстрой сортировки на выше указанных наборах данных.

Для выполнения задания, из приложения к заданию, были скопированы функции shell() и qs(), производящие сортировку Шелла и быструю сортировку соответственно. Для измерения времени были выбраны те же самые функции — clock() и константа CLOCKS\_PER\_SEC.

В качестве тестовых данных — массив из 10000 элементов. Определены так же несколько функций, которые заполняют данный массив последовательностью из случайных чисел, возрастающей и убывающей последовательность, а так же последовательностью чисел возрастающей до середины и убывающей после середины. Последовательно были запущены функции сортировки Шелла, быстрой сортировки и быстрой сортировки, встроенной в одну из используемых библиотек и вызывающуюся с помощью функции qsort(). При это функция принимает в качестве входных параметров: указатель на массив, количество элементов, размер одного элемента и функцию, которая определяет отношение между двумя числами в отсортированном массиве.

