Лабораторная работа 1

ИЗУЧЕНИЕ МЕТОДОВ ОЦЕНКИ АЛГОРИТМОВ

Цель работы: Изучение методов оценки алгоритмов и программ и определение временной и емкостной сложности типовых алгоритмов и программ.

Вариант: 8.

Условие: Составить программу, которая формирует матрицу из n*nслучайных чисел. Определить сумму чисел, лежащих выше главной диагонали матрицы.

Асимтотическая сложность алгоритма

$1 + (1 + (n+1)*3 + 3*n) + [k = 0, k = n-1]\Sigma((2 + 2 + 1 + 1 + 1)*k)$

$= 2 + 3*n + 3 + 3*n + 7*[k = 0, k = n-1]\Sigma k$

$= 6*n + 5 + 7*(0 + 1 + 2 + \ldots + (n-1))$

$= 6*n + 5 + 7*(1 + 2 + \ldots (n-1))$

$= [Sn = 0.5 * (a1 + a2) * n] = 6*n + 5 + 7 * 0.5 * (1 + n – 1) * (n - 1)$

$= 6*n + 5 + 3.5*n*(n-1)$

$= 3.5 * n^2 – 3.5*n + 6*n + 5$

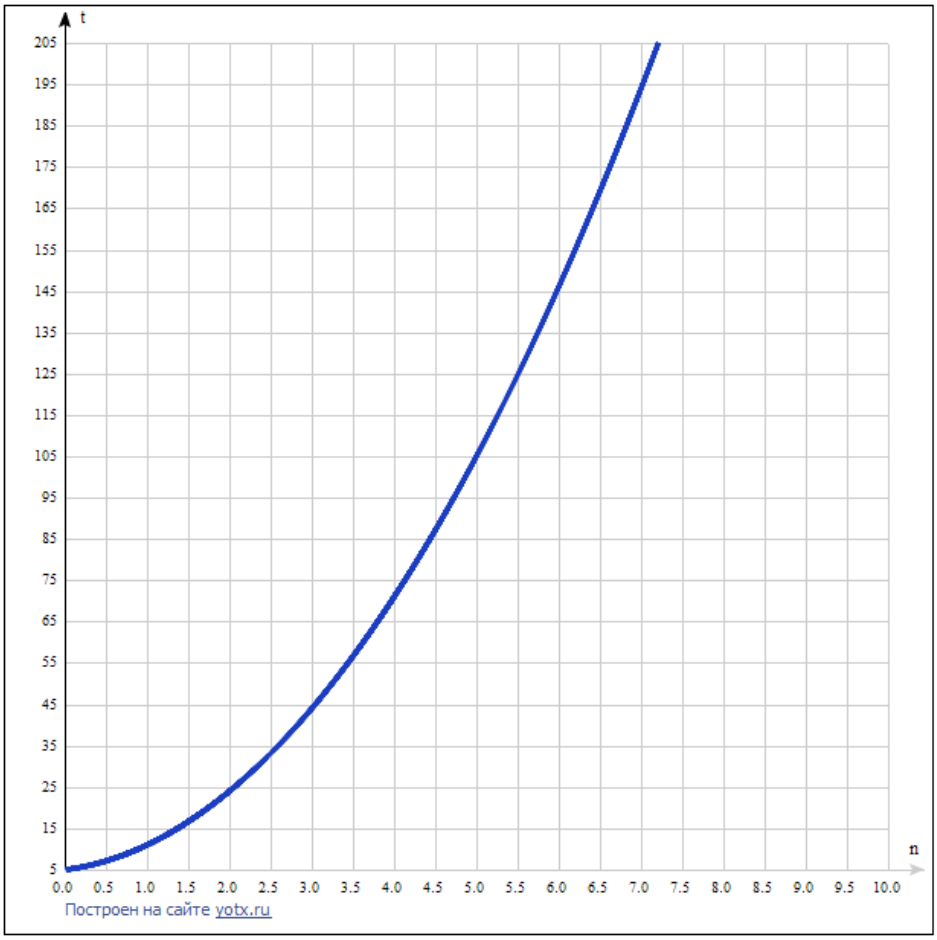
$= 3.5 * n^2 + 2.5*n + 5 = O(n^2)$

Таблица зависимости времени выполнения алгоритма от объема исходных данных

Время выполнения(в миллисекундах)	Объем исходных данных
232.653	Матрица 10000x10000
128.519	Матрица 7500x7500
56.3799	Матрица 5000x5000
57.6007	Матрица 2500x2500

2.1961	Матрица 1000x1000

График зависимости времени выполнения алгоритма от объема исходных данных



Код алгоритма

```
313     //*****************************************************************************
314     template<class ElementType>
315     ElementType Matrix<ElementType>::findSumOfElementsUnderMainDiagonal() const
316     {
317         ElementType sumOfElementsUnderMainDiagonal = 0;
318         for (int i = 0; i < this->amountOfRows; i++)
319         {
320             for (int j = i + 1; j < this->amountOfColumns; j++)
321             {
322                 sumOfElementsUnderMainDiagonal += *(*(this->elements + i) + j);
323             }
324         }
325         return sumOfElementsUnderMainDiagonal;
326     }
327     //*****************************************************************************
```

Исходный код программы эксперементальной оценки временной сложности алгоритма для матрицы размерности 10000x10000

```
    const double minimalBorderOfGeneration = -10;
    const double maximalBorderOfGeneration = 10;
    const ElementsGenerator<double> * const elementsGenerator
        = new ElementsGenerator<double>(minimalBorderOfGeneration, maximalBorderOfGeneration);

    //For matrix 10000x10000
    int amountOfRowsOfMatrix = 10000;
    int amountOfColumnsOfMatrix = 10000;
    int amountOfElementsOfMatrix = amountOfRowsOfMatrix * amountOfColumnsOfMatrix;

    double *valuesOfElementsOfMatrix = elementsGenerator->generateElements(amountOfElementsOfMatrix);

    const Matrix<double> *matrix = new Matrix<double>(valuesOfElementsOfMatrix, amountOfRowsOfMatrix,
        amountOfColumnsOfMatrix);
    delete[] valuesOfElementsOfMatrix;

    cout << "For matrix 10000x10000:\n";
    chrono::steady_clock::time_point startOfExecution = chrono::high_resolution_clock::now();
    matrix->findSumOfElementsUnderMainDiagonal();
    chrono::steady_clock::time_point endOfExecution = chrono::high_resolution_clock::now();
    delete matrix;
    chrono::duration<double> durationOfExecution = endOfExecution - startOfExecution;
    cout << "Duration of execution in seconds : " << durationOfExecution.count() << "\n";
```

Исходный код программы эксперементальной оценки временной сложности алгоритма для матрицы размерности 7500x7500

```
    //For matrix 7500x7500
    amountOfRowsOfMatrix = 7500;
    amountOfColumnsOfMatrix = 7500;
    amountOfElementsOfMatrix = amountOfRowsOfMatrix * amountOfColumnsOfMatrix;

    valuesOfElementsOfMatrix = elementsGenerator->generateElements(amountOfElementsOfMatrix);
    matrix = new Matrix<double>(valuesOfElementsOfMatrix, amountOfRowsOfMatrix,
        amountOfColumnsOfMatrix);
    delete[] valuesOfElementsOfMatrix;

    cout << "For matrix 7500x7500:\n";
    startOfExecution = chrono::high_resolution_clock::now();
    matrix->findSumOfElementsUnderMainDiagonal();
    endOfExecution = chrono::high_resolution_clock::now();
    delete matrix;
    durationOfExecution = endOfExecution - startOfExecution;
    cout << "Duration of execution in seconds : " << durationOfExecution.count() << "\n";
```

Исходный код программы эксперементальной оценки временной сложности алгоритма для матрицы размерности 5000x5000

```
62      //For matrix 5000x5000
63      amountOfRowsOfMatrix = 5000;
64      amountOfColumnsOfMatrix = 5000;
65      amountOfElementsOfMatrix = amountOfRowsOfMatrix * amountOfColumnsOfMatrix;
66
67      valuesOfElementsOfMatrix = elementsGenerator->generateElements(amountOfElementsOfMatrix);
68      matrix = new Matrix<double>(valuesOfElementsOfMatrix, amountOfRowsOfMatrix,
69          amountOfColumnsOfMatrix);
70      delete[] valuesOfElementsOfMatrix;
71
72      cout << "For matrix 5000x5000:\n";
73      startOfExecution = chrono::high_resolution_clock::now();
74      matrix->findSumOfElementsUnderMainDiagonal();
75      endOfExecution = chrono::high_resolution_clock::now();
76      delete matrix;
77      durationOfExecution = endOfExecution - startOfExecution;
78      cout << "Duration of execution in seconds : " << durationOfExecution.count() << "\n";
```

Исходный код программы эксперементальной оценки временной сложности алгоритма для матрицы размерности 2500x2500

```
80      //For matrix 2500x2500
81      amountOfRowsOfMatrix = 5000;
82      amountOfColumnsOfMatrix = 5000;
83      amountOfElementsOfMatrix = amountOfRowsOfMatrix * amountOfColumnsOfMatrix;
84
85      valuesOfElementsOfMatrix = elementsGenerator->generateElements(amountOfElementsOfMatrix);
86      matrix = new Matrix<double>(valuesOfElementsOfMatrix, amountOfRowsOfMatrix,
87          amountOfColumnsOfMatrix);
88      delete[] valuesOfElementsOfMatrix;
89
90      cout << "For matrix 2500x2500:\n";
91      startOfExecution = chrono::high_resolution_clock::now();
92      matrix->findSumOfElementsUnderMainDiagonal();
93      endOfExecution = chrono::high_resolution_clock::now();
94      delete matrix;
95      durationOfExecution = endOfExecution - startOfExecution;
96      cout << "Duration of execution in seconds : " << durationOfExecution.count() << "\n";
```

Исходный код программы эксперементальной оценки временной сложности алгоритма для матрицы размерности 1000x1000

```
98      //For matrix 1000x1000
99      amountOfRowsOfMatrix = 1000;
100     amountOfColumnsOfMatrix = 1000;
101     amountOfElementsOfMatrix = amountOfRowsOfMatrix * amountOfColumnsOfMatrix;
102
103     valuesOfElementsOfMatrix = elementsGenerator->generateElements(amountOfElementsOfMatrix);
104     matrix = new Matrix<double>(valuesOfElementsOfMatrix, amountOfRowsOfMatrix,
105         amountOfColumnsOfMatrix);
106     delete[] valuesOfElementsOfMatrix;
107
108     cout << "For matrix 1000x1000:\n";
109     startOfExecution = chrono::high_resolution_clock::now();
110     matrix->findSumOfElementsUnderMainDiagonal();
111     endOfExecution = chrono::high_resolution_clock::now();
112     delete matrix;
113     durationOfExecution = endOfExecution - startOfExecution;
114     cout << "Duration of execution in seconds : " << durationOfExecution.count() << "\n";
```

Полный проект: https://github.com/vladzuev00/matrix-project-for-bsu-cpp