

Федеральное государственное автономное образовательное учреждение высшего образования  
«Санкт-Петербургский политехнический университет Петра Великого»  
Институт компьютерных наук и технологий  
Высшая школа киберфизических систем и управления

УДК 004.421

УТВЕРЖДАЮ

\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_  
« \_\_\_\_ » \_\_\_\_\_ г.

ОТЧЕТ  
по дисциплине «Теория и технологии программирования»

## Лабораторная работа №6

**Выполнил:**

студент гр. 3532703/00001

\_\_\_\_\_ Д.П. Кимельфельд  
подпись, дата

Санкт-Петербург 2021 г.

## ***Оглавление***

Введение.....	3
1 Текст программы.....	4
2 Пример работы.....	8
Вывод.....	10

## ***Введение***

### **1. Цель работы.**

Цель задания - изучить методы сортировки данных.

### **2. Задание.**

Составить программу для сортировки массива данных методами: пузырьковой, отбора, вставки, Шелла и быстрой сортировки. Вывести на экран неупорядоченную (один раз) и упорядоченные (для каждого из методов) массивы данных. Составить сравнительную таблицу эффективности методов, в которой необходимо указать число сравнений и перестановок переменных в каждом методе сортировки.

Неупорядоченная матрица задается один раз случайным образом, далее она используется для каждого из методов сортировки.

Вариант	Дана матрица $A [M, N]$
13	Упорядочить каждый столбец матрицы по возрастанию абсолютных величин.

Выходные данные должны быть сформатированы в виде таблицы, в которой необходимо отобразить количество сравнений и перестановок для каждой из сортировок.

## 1 Текст программы

```
#include <iostream>
#include<cmath>
#include <algorithm>
#include <string>
#include <ctime>
#include<iomanip>
using namespace std;

void inputMatrix(int A, int N, int **matrix) {
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < A; j++) {
            matrix[i][j] = rand() % 20 - 10; // rand() % (B-A) - A -> Необходимый
диапазон рандомных чисел
            //cin >> matrix[i][j];
            cout << internal << setw(2) << matrix[i][j] << "\t";
        }
        cout << endl;
    }
}

void copyMatrix(int **m0, int **m, int A, int N) {
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < A; j++) {
            m[i][j] = m0[i][j];
        }
    }
}

void selectSort(int j, int N, int **matrix, unsigned int &c, unsigned int &m) { // при 4
элементах не дб 6 сравнений // Сортировка простым выбором
    int ci = 0;
    int mi = 0;
    int smallestIndex;
    for (int i = 0; i < N - 1; i++) {
        smallestIndex = i;
        for (int k = i + 1; k < N; k++) {
            if (abs(matrix[k][j]) < abs(matrix[smallestIndex][j])) {
                smallestIndex = k;
                ci++;
            }
            else ci++;
        }
        if (smallestIndex == i) continue;
        swap(matrix[i][j], matrix[smallestIndex][j]);
        mi++;
    }
    c = c + ci;
    m = m + mi;
}

void bubbleSort(int j, int N, int **matrix, unsigned int &c, unsigned int &m) {
    bool optimum;
    for (int i = 0; i < N - 1; i++) {
        optimum = false;
        for (int k = 0; k < N - i - 1; k++) {
            c++; // сравнения
            if (abs(matrix[k][j]) > abs(matrix[k+1][j])) {
                swap(matrix[k][j], matrix[k+1][j]); // перестановка
                optimum = true;
                m++;
            }
        }
    }
}
```

```

    }
        if (optimum == false) break;
    }
}

void insertSort(int j, int N, int **matrix, unsigned int &c, unsigned int &m) { // Сортировка
    ВСТАВКАМИ
    int ci = 0;
    int mi = 0;
    int save = 1;
    for (int i = 1; i < N ; i++) { //ST
        save = i;
        bool check = false;
        for (int k = i - 1; k >= 0 ; k--) {
            if (abs(matrix[k][j]) <= abs(matrix[save][j])) { //ST
                ci++;
                break;
            }
            swap(matrix[save][j], matrix[k][j]);
            save--;
            check = true;
            ci++;
        }
        if (check == true) mi++;
    }

    c = c + ci;
    m = m + mi;
}

void outputSortMatrix(int A, int N, int **matrix) {
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < A; j++) {
            cout << internal << setw (2) << matrix[i][j] << "\t";
        }
        cout << endl;
    }
}

void quickSort(int **matrix, int j, int firstIndex, int lastIndex, unsigned int &c, unsigned int
&m) {
    if (firstIndex < lastIndex) {
        int ci = 0;
        int mi = 0;
        int left = firstIndex;
        int right = lastIndex;
        int pivotIndex = (left + right + 1) / 2;
        int pivot = abs(matrix[pivotIndex][j]);
        c = c + (lastIndex - firstIndex);
        do {
            mi = 0;
            bool rightSortHand = false;
            bool leftSortHand = false;
            bool ptpLeft = false;
            bool ptpRight = false;
            while (abs(matrix[left][j]) < pivot) left++;
            while (abs(matrix[right][j]) > pivot) right--;
            if (right - left > 1 || left - right > 1) {
                if (left == pivotIndex) leftSortHand = true;
                if (right == pivotIndex) rightSortHand = true;
            }
            if (left == pivotIndex) ptpLeft = true;
            if (right == pivotIndex) ptpRight = true;
        }
    }
}

```

```

        if (left <= right) {
            swap(matrix[left][j], matrix[right][j]);
            if (ptpLeft && abs(matrix[left][j]) != abs(matrix[right][j]))
                privotIndex = right;
            if (ptpRight && abs(matrix[left][j]) != abs(matrix[right][j]))
                privotIndex = left;
            if ((left != right) && abs(matrix[left][j]) != abs(matrix[right][j]))
                mi++;
            left++;
            right--;
            if (!(ptpRight) && rightSortHand)
                right++;
            if (!(ptpLeft) && leftSortHand)
                left--;
        }
        m = m + mi;
    } while (left <= right);
    if (left == privotIndex) left++;
    if (right == privotIndex) right--;
    quickSort(matrix, j, firstIndex, right, c, m);
    quickSort(matrix, j, left, lastIndex, c, m);
}

void shellsSort(int j, int N, int **matrix, unsigned int &c, unsigned int &m){
    int buff;
    int s;
    for (int step = N / 2; step > 0; step = step / 2) {
        for (int rInsert = step; rInsert < N; rInsert++) {
            buff = matrix[rInsert][j];
            bool check = false;
            for (s = rInsert; s >= step; s -= step) {
                s++;
                if (abs(buff) < abs(matrix[s - step][j])) {
                    matrix[s][j] = matrix[s - step][j];
                    check = true;
                }
                else {
                    break;
                }
            }
            if (check == true) m++;
            matrix[s][j] = buff;
        }
    }
}

int main() {
    setlocale(LC_ALL, "Russian");
    int A, N;
    srand(time(NULL));
    cout << "\\ \\ \\ Сортировка элементов по модулю в столбцах матрицы в порядке неубывания\\ \\ \\ "
    << endl;
    cout << "Введите кол-во строк в матрице: ";
    cin >> N;
    cout << endl << "Введите кол-во столбцов в матрице: ";
    cin >> A;
    //cout << endl << endl << "Изначальный массив" << endl;
    // Формирование массивов в памяти
    int **matrix = new int*[N];
    int **matrixCopy = new int*[N];
    for (int i = 0; i < N; i++) {
        matrix[i] = new int[A];
    }
}

```

```

        matrixCopy[i] = new int[A];
    }
    // Создание массивов счетчиков
    unsigned int C[5];
    unsigned int M[5];
    for (int i = 0; i < 5; i++) {
        C[i] = 0;
        M[i] = 0;
    }
    // Заполнение одного из массивов, а затем копирование
    string s[] = { "Сортировка 'пузырьком'", "Сортировка простым выбором", "Сортировка
вставками", "Быстрая сортировка", "Сортировка Шелла" };
    inputMatrix(A, N, matrix);
    copyMatrix(matrix, matrixCopy, A, N);
    // Сортировка
    for (int sortingType = 0; sortingType < 5; sortingType++) {
        if (N == 1) continue;
        for (int j = 0; j < A && sortingType == 0; j++) bubbleSort(j, N,
matrix, C[sortingType], M[sortingType]);
        for (int j = 0; j < A && sortingType == 1; j++) selectSort(j, N,
matrix, C[sortingType], M[sortingType]);
        for (int j = 0; j < A && sortingType == 2; j++) insertSort(j, N,
matrix, C[sortingType], M[sortingType]);
        for (int j = 0; j < A && sortingType == 3; j++) quickSort(matrix, j, 0, N-
1, C[sortingType], M[sortingType]);
        for (int j = 0; j < A && sortingType == 4; j++) shellsSort(j, N, matrix,
C[sortingType], M[sortingType]);
        cout << s[sortingType] << endl;
        outputSortMatrix(A, N, matrix);
        copyMatrix(matrixCopy, matrix, A, N);
    }
    // Табличка
    for (int i = -1; i < 5; i++, cout << endl) {
        if (i == -1) {
            cout << endl << " | _____ | _____ | _____ | "
<< endl << "#      Тип сортировки      # Сравнения  # Перестановки # " << endl <<
"# _____ # _____ # _____ # ";
        }
        else
        {
            cout << " | " << setw(26) << s[i] << " | " << setw(14) << C[i] << " | " <<
setw(14) << M[i] << " | ";
        }
        cout << " # _____ # _____ # _____ # ";

    }

    // Чистим память =)
    for (int i = 0; i < N ; i++) {
        delete[] matrix[i];
        delete[] matrixCopy[i];
    }
    delete[] matrix;
    delete[] matrixCopy;
    //
    system("pause");
    return 0;
}

```

## 2 Пример работы программы

```

C:\Users\HUAWEI\source\repos\Project11\Debug\ТПП_6.exe
\\Сортировка элементов по модулю в столбцах матрицы в порядке неубывания\\
Введите кол-во строк в матрице: 5

Введите кол-во столбцов в матрице: 1
-8
-4
-6
3
-2
Сортировка 'пузырьком'
-2
3
-4
-6
-8
Сортировка простым выбором
-2
3
-4
-6
-8
Сортировка вставками
-2
3
-4
-6
-8
Быстрая сортировка
-2
3
-4
-6
-8
Сортировка Шелла
-2
3
-4
-6
-8

```

Тип сортировки	# Сравнения	# Перестановки
Сортировка 'пузырьком'	10	9
Сортировка простым выбором	10	3
Сортировка вставками	10	4
Быстрая сортировка	7	3
Сортировка Шелла	9	4

```

# Для продолжения нажмите любую клавишу . . .

```

Рисунок 1 – пример №1 работы программы



```

C:\Users\HUAWEI\source\repos\Project11\Debug\TTP_6.exe
\\Сортировка элементов по модулю в столбцах матрицы в порядке неубывания\\
Введите кол-во строк в матрице: 5

Введите кол-во столбцов в матрице: 10
-4 1 2 1 -5 2 -5 -1 -2 9
7 -10 8 4 2 4 7 -10 -6 2
-8 2 -8 4 4 -1 -10 0 -2 -4
-6 9 6 -3 9 2 -9 6 0 -5
6 -5 3 3 3 0 -3 3 -9 -7
Сортировка 'пузырьком'
-4 1 2 1 2 0 -3 0 0 2
-6 2 3 -3 3 -1 -5 -1 -2 -4
6 -5 6 3 4 2 7 3 -2 -5
7 9 8 4 -5 2 -9 6 -6 -7
-8 -10 -8 4 9 4 -10 -10 -9 9
Сортировка простым выбором
-4 1 2 1 2 0 -3 0 0 2
-6 2 3 -3 3 -1 -5 -1 -2 -4
6 -5 6 3 4 2 7 3 -2 -5
7 9 8 4 -5 2 -9 6 -6 -7
-8 -10 8 4 9 4 -10 -10 -9 9
Сортировка вставками
-4 1 2 1 2 0 -3 0 0 2
-6 2 3 -3 3 -1 -5 -1 -2 -4
6 -5 6 3 4 2 7 3 -2 -5
7 9 8 4 -5 2 -9 6 -6 -7
-8 -10 -8 4 9 4 -10 -10 -9 9
Быстрая сортировка
-4 1 2 1 2 0 -3 0 0 2
-6 2 3 -3 3 -1 -5 -1 -2 -4
6 -5 6 3 4 2 7 3 -2 -5
7 9 8 4 -5 2 -9 6 -6 -7
-8 -10 8 4 9 4 -10 -10 -9 9
Сортировка Шелла
-4 1 2 1 2 0 -3 0 0 2
-6 2 3 -3 3 -1 -5 -1 -2 -4
6 -5 6 3 4 2 7 3 -2 -5
7 9 8 4 -5 2 -9 6 -6 -7
-8 -10 -8 4 9 4 -10 -10 -9 9

# Тип сортировки # Сравнения # Перестановки #
# # # #
Сортировка 'пузырьком' 93 47
Сортировка простым выбором 100 28
Сортировка вставками 80 26
Быстрая сортировка 71 27
Сортировка Шелла 89 31
# # # #
#Для продолжения нажмите любую клавишу . . .

```

Рисунок 2 – пример №2 работы программы

```

C:\Users\HUAWEI\source\repos\Project11\Debug\TTP_6.exe
\\Сортировка элементов по модулю в столбцах матрицы в порядке неубывания\\
Введите кол-во строк в матрице: 1

Введите кол-во столбцов в матрице: 10
8 -4 2 7 9 -3 6 -10 4 -3

# Тип сортировки # Сравнения # Перестановки #
# # # #
Сортировка 'пузырьком' 0 0
Сортировка простым выбором 0 0
Сортировка вставками 0 0
Быстрая сортировка 0 0
Сортировка Шелла 0 0
# # # #
#Для продолжения нажмите любую клавишу . . .

```

Рисунок 3 – пример №3 работы программы

## ***Вывод***

Лабораторная работа по методам сортировок познакомила меня с важными алгоритмами в упорядочивании чисел. В ходе данной работы я закрепил свои знания в работе с двумерными динамическими массивами, повторил такое понятие, как указатель, узнал, что Сортировка Шелла является оптимизированной сортировкой вставками, научился вручную сосчитывать количество сравнений и перестановок для различных маленьких массивов чисел, что несомненно позволило закрепить знания в алгоритмах сортировок.

Так же во время тестирования программ было очевидно, что такие сортировки, как сортировка Шелла и Быстрая, рассчитаны на массивы больших размеров. Этот факт я собираюсь проанализировать в своей курсовой работе по дисциплине “Теория и технологии программирования”.