МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА – Российский технологический университет»
РТУ МИРЭА

Институт Информационных технологий

Кафедра Математического обеспечения и стандартизации информационных технологий

ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ № 8

по дисциплине
«Структуры и алгоритмы обработки данных»

Тема: «Усовершенствованные сортировки»

Выполнил студент группы ИКБО-18-22 Ракитин В.А.

Принял преподаватель Филатов А.С.

Лабораторная работа выполнена «__»_____202__ г. *(подпись студента)*

«Зачтено» «__»_____202__ г. *(подпись руководителя)*

Москва 2023

1. Цель работы

Получение навыков по анализу вычислительной сложности нескольких алгоритмов сортировки и определение наиболее эффективного алгоритма.

2. Постановка задачи

Разработать три алгоритма сортировки, определенные вариантом. Провести анализ вычислительной и емкостной сложности алгоритма на массивах, заполненных случайно. Определить наиболее эффективный алгоритм.

1. Разработать алгоритм простой сортировки, определенной вариантом. Определить емкостную и временную сложность алгоритма.
2. Разработать алгоритм усовершенствованной сортировки, определенной вариантом. Определить емкостную и временную сложность алгоритма.
3. Разработать алгоритм сортировки методом простого слияния. Определить емкостную и временную сложность алгоритма.
4. Провести контрольные прогоны функций на уже отсортированных массивах, отсортированных в обратном порядке и сгенерированных с помощью генератора псевдослучайных чисел различных размеров (N>999). Рабочие прогоны функций должны проводиться на одинаковых массивах. Во время сортировки вычислять время её выполнения t. Провести эмпирическую (практическую) оценку вычислительной сложности алгоритмов для трех случаев, для чего предусмотреть в программе подсчет фактического количества операций сравнения C и количества операций перемещения M. Полученные результаты свести в сводные таблицы.
5. Представить графики зависимости C+M от N и t от N для анализируемых алгоритмов в трех случаях.
6. Провести анализ зависимости алгоритмов сортировок от размера и исходной упорядоченности массива по составленным таблицам. Определить эффективный алгоритм для каждого случая.

Вариант №5. Условие задания:

Упражнение	1. Алгоритм простой сортировки простой вставки 2. Алгоритм усовершенствованной сортировки Шелла со смещениями Д. Кнута 3. Алгоритм сортировки простым слиянием

3. Решение

Для решения первого упражнения была написана функция insertion_sort, реализующая сортировку простой вставкой. Сортировка простой вставки – это простой алгоритм сортировки. Суть его заключается в том, что на каждом шаге алгоритма мы берем один из элементов массива, находим позицию для вставки и вставляем. Функция на вход получает на вход четыре значения – массив, размер массива, число перемещений элементов массива и число операций сравнения. Далее программа сортирует массив методом простой вставки.

```
void insertion_sort(int* arr, long long int N, long long int& C, long long
int& M) { //Сортировка простой вставки
    int key = 0, j = 0;
    for (int i = 1; i < N; i++)
    {
        key = arr[i];
        int l = 0, r = i - 1;
        while (l <= r) {
            int m = (l + r) / 2;
            if (arr[m] < key) {
                l = m + 1;
                C++;
            }
            else {
                r = m - 1;
                C++;
            }
        }
        for (j = i - 1; j >= l; j--) {
            arr[j + 1] = arr[j];
            M++;
        }
        arr[l] = key;
        M++;
    }
}
```

Для решения второго упражнения была написана функция shellSort, реализующая сортировку Шелла со смещениями Д. Кнута. Сортировка Шелла со смещением Дональда Кнута – это усовершенствованная версия сортировки

вставками. Суть метода заключается в том, чтобы разбить массив на подмассивы и отсортировать каждый подмассив по отдельности методом вставок. Однако вместо того, чтобы делить массив на равные части, как в сортировке слиянием, сортировка Шелла использует последовательность шагов, которые в последствии уменьшаются до 1. Это позволяет ускорить сортировку, так как элементы, находящиеся на большом расстоянии друг от друга, сравниваются и перемещаются раньше, чем в случае сортировки вставками. Функция на вход получает на вход четыре значения – массив, размер массива, число перемещений элементов массива и число операций сравнения. Далее программа сортирует массив сортировкой Шелла со смещениями Д. Кнута.

```
void shellSort(int* arr, long long int N, long long int& C, long long int&
M) { //Сортировка Шелла со смещением Д. Кнута
    // Вычисляем начальное значение смещения
    int gap = 1;
    while (gap < N / 3) {
        gap = gap * 3 + 1;
    }
    // Сортируем массив для каждого значения смещения
    while (gap > 0) {
        // Применяем сортировку вставками для каждого подмассива
        for (int i = gap; i < N; i++) {
            int temp = arr[i];
            int j = i;
            C++;
            while (j >= gap && arr[j - gap] > temp) {
                arr[j] = arr[j - gap];
                j -= gap;
                M++;
                C++;
            }
            arr[j] = temp;
            M++;
        }
        // Уменьшаем смещение
        gap = (gap - 1) / 3;
    }
}
```

Для решения третьего упражнения была написана функция merge, реализующая сортировку простым слиянием. Сортировка простым слиянием – это алгоритм сортировки, который разделяет массив на две части, сортирует каждую из них отдельно, а затем объединяет их в один отсортированный список. Функция на вход получает на вход четыре значения – массив, размер

массива, число перемещений элементов массива и число операций сравнения.

Далее программа сортирует массив сортировкой простым слиянием.

```
void merge(int* arr, long long int l, long long int m, long long int r, long
long int& C, long long int& M) { //Сортировка простым слиянием
    long long int i, j, k, n1 = m - l + 1, n2 = r - m;
    int* L = new int[n1];
    int* R = new int[n2];

    for (i = 0; i < n1; i++) {
        L[i] = arr[l + i];
        M++;
    }
    for (j = 0; j < n2; j++) {
        R[j] = arr[m + 1 + j];
        M++;
    }
    i = 0;
    j = 0;
    k = l;
    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) {
            arr[k] = L[i];
            i++;
            M++;
        }
        else {
            arr[k] = R[j];
            j++;
            M++;
        }
        k++;
        C++;
    }
    while (i < n1) {
        arr[k] = L[i];
        i++;
        k++;
        M++;
    }
    while (j < n2) {
        arr[k] = R[j];
        j++;
        k++;
        M++;
    }
    delete[] R;
    delete[] L;
}
void mergeSort(int* arr, long long int l, long long int r, long long int& C,
long long int& M) {
    if (l < r) {
        long long int m = l + (r - l) / 2;
        mergeSort(arr, l, m, C, M);
        mergeSort(arr, m + 1, r, C, M);
        merge(arr, l, m, r, C, M);
    }
}
```

Также для решения третьего упражнения была написана функция mergeSort, которая рекурсивно вызывает саму себя и вызывает функцию сортировки. Функция на вход получает на вход пять значений – массив, число 0, размер массива, число перемещений элементов массива и число операций сравнения.

```
void mergeSort(int* arr, long long int l, long long int r, long long int& C,
long long int& M) {
    if (l < r) {
        long long int m = l + (r - l) / 2;
        mergeSort(arr, l, m, C, M);
        mergeSort(arr, m + 1, r, C, M);
        merge(arr, l, m, r, C, M);
    }
}
```

Для решения упражнений была написана функция fillarr, которая получает на вход два значения: массив и его размер. Программа заполняет массив случайными числами.

```
void fillarr(int* arr, long long int N) { //Функция заполнения массива
      for (long long int i = 0; i < N; i++) {
            arr[i] = rand() % 100000000; //Заполняем массив случайными
числами
      }
}
```

При запуске программы пользователь видит пользовательское меню, где пользователю надо ввести размер массива.

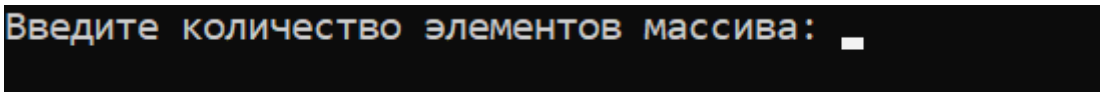

Рисунок 1. Интерфейс программы

4. Тестирование

Протестируем программой выполнение упражнения. Составим таблицы, в которых будут данные, зависящие от количества элементов массива.

Таблица 1. Таблица результатов тестирования сортировки простой вставкой в простом случае

N	t, c.	$T_a(n)=N^2$	$T_э=C+M$	$T_э/T_a$
100000	7.0057	10000000000	2507819689	0.25078

150000	14.0731	22500000000	5643376195	0.25082
200000	22.794	40000000000	10029024200	0.25073
250000	34.1855	62500000000	15640991584	0.25026
300000	48.4912	90000000000	22501613352	0.25002

Таблица 2. Таблица результатов тестирования сортировки простой вставкой в лучшем случае

N	**t, c.**	**$T_a(n)=N$**	**$T_э=C+M$**	**$T_э/T_a$**
100000	0.0075635	100000	1800834	18.00834
150000	0.0089938	150000	2893726	19.29151
200000	0.0133996	200000	4088853	20.44427
250000	0.0156176	250000	5375373	21.50149
300000	0.0241948	300000	6750649	22.50216

Таблица 3. Таблица результатов тестирования сортировки простой вставкой в худшем случае

N	**t, c.**	**$T_a(n)=N^2$**	**$T_э=C+M$**	**$T_э/T_a$**
100000	10.3009	10000000000	5001518945	0.50015
150000	22.9421	22500000000	11252362874	0.50011
200000	40.866	40000000000	20003237874	0.50008
250000	63.7299	62500000000	31254112874	0.50007
300000	92.0093	90000000000	45005025731	0.50006

Таблица 4. Таблица результатов тестирования сортировкой Шелла со смещениями Д. Кнута в простом случае

N	**t, c.**	**$T_a(n)=N^{(5/4)}$**	**$T_э=C+M$**	**$T_э/T_a$**
10000000	4.62266	562341325	1889109836	3.35937
11000000	6.78409	633491594	2098481358	3.31256
12000000	7.68211	706279430	2408349750	3.40991
13000000	8.03153	780601157	2571269304	3.29396
14000000	8.48236	856367239	2764524492	3.22820

Таблица 5. Таблица результатов тестирования сортировкой Шелла со смещениями Д. Кнута в лучшем случае

N	**t, c.**	**$T_a(n)=N*\log^2 N$**	**$T_э=C+M$**	**$T_э/T_a$**
10000000	0.545216	5407020090	278476656	0.05150
11000000	0.821184	6018527690	308476656	0.05125
12000000	0.682121	6636591470	338476656	0.05100
13000000	0.723027	7260128510	368476656	0.05075
14000000	0.785522	7889561690	398476656	0.05051

Таблица 6. Таблица результатов тестирования сортировкой Шелла со смещениями Д. Кнута в худшем случае

N	**t, c.**	**$T_a(n)=N^{(5/4)}$**	**$T_э=C+M$**	**$T_э/T_a$**
10000000	0.735702	562341325	389131064	0.69198
11000000	0.77879	633491594	413035384	0.65200
12000000	0.931775	706279430	494349786	0.69994

13000000	0.977936	780601157	516083514	0.66114
14000000	1.08139	856367239	568520538	0.66387

Таблица 7. Таблица результатов тестирования сортировкой слиянием в простом случае

N	**t, c.**	**$T_a(n)=N*log(N)$**	**$T_э=C+M$**	**$T_э/T_a$**
10000000	6.79937	232530000	686543646	2.95249
11000000	7.29177	257301000	760093999	2.95410
12000000	7.52417	282204000	833728481	2.95435
13000000	8.00403	307216000	907465913	2.95384
14000000	8.44187	332346000	981337473	2.95276

Таблица 8. Таблица результатов тестирования сортировкой слиянием в лучшем случае

N	**t, c.**	**$T_a(n)=N*log(N)$**	**$T_э=C+M$**	**$T_э/T_a$**
10000000	5.15453	232530000	585233728	2.51681
11000000	5.68835	257301000	648198848	2.51922
12000000	6.15083	282204000	710778624	2.51867
13000000	6.77726	307216000	773211840	2.51683
14000000	7.27996	332346000	835842624	2.51498

Таблица 9. Таблица результатов тестирования сортировкой слиянием в худшем случае

N	**t, c.**	**$T_a(n)=N*log(N)$**	**$T_э=C+M$**	**$T_э/T_a$**

10000000	6.02485	232530000	686546127	2.95251
11000000	6.57983	257301000	760094073	2.95410
12000000	7.19075	282204000	833729209	2.95435
13000000	7.86888	307216000	907469642	2.95385
14000000	8.44966	332346000	981339367	2.95276

На основе данных таблиц построим графики зависимости C+M от N и t от N.

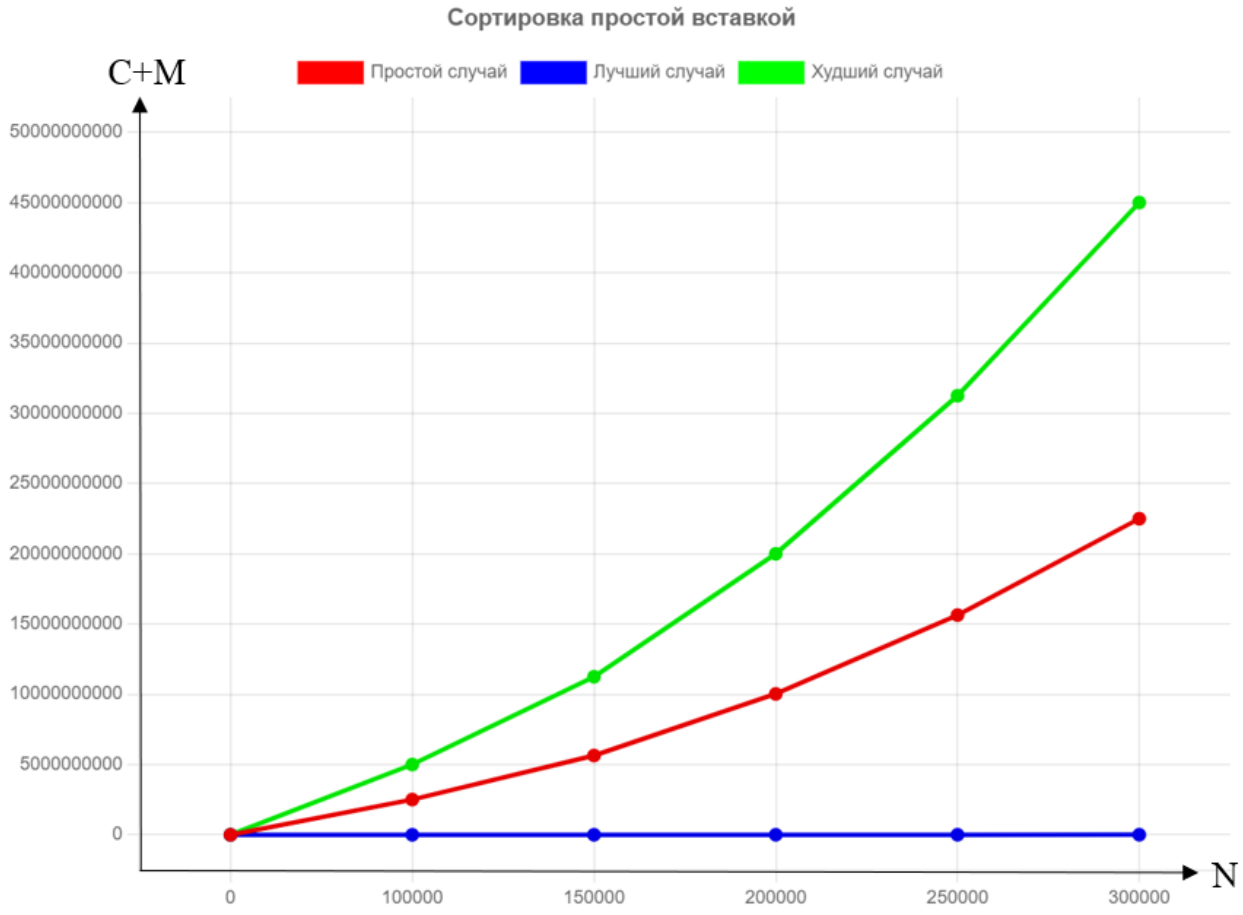

Рисунок 2. График зависимости C+M от N сортировки простой вставки

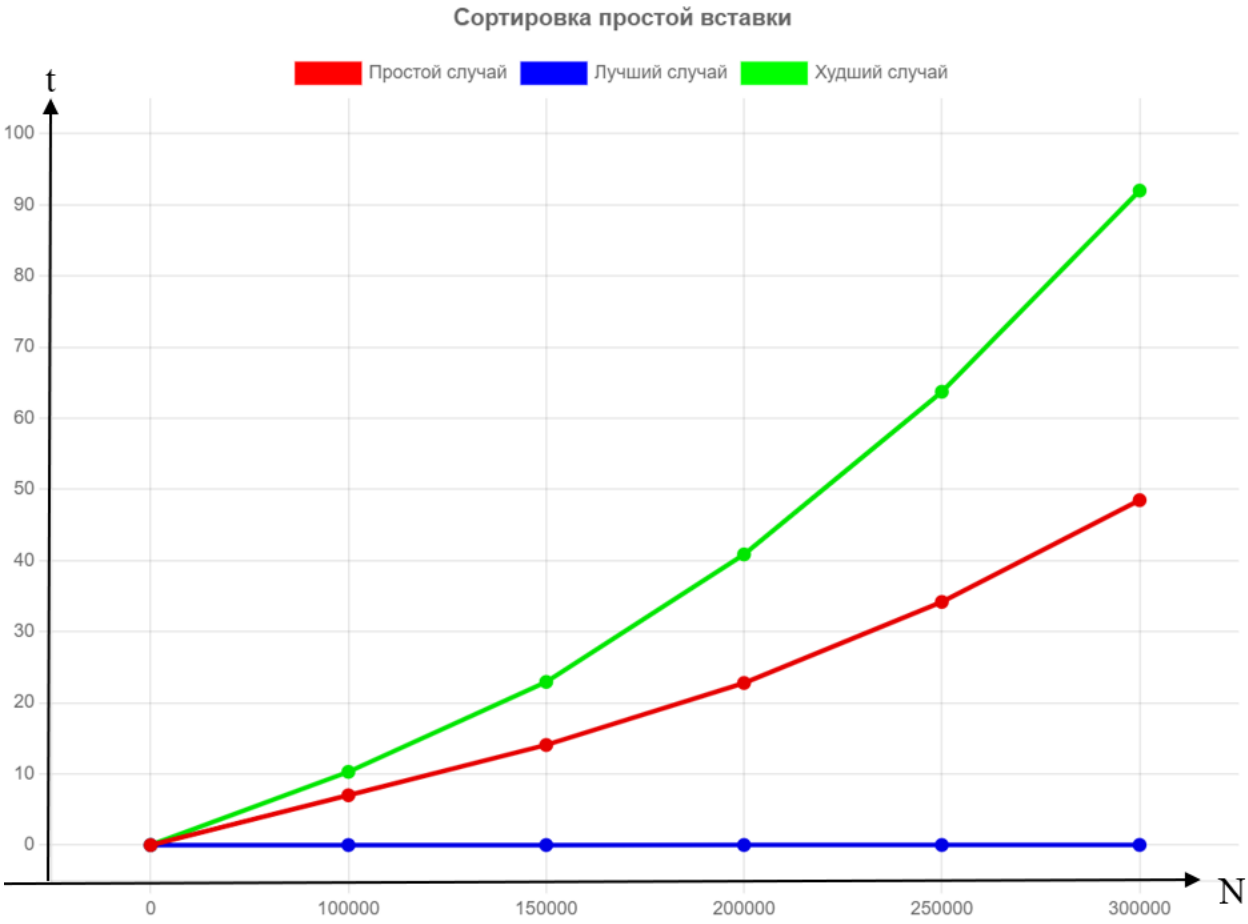

Рисунок 3. График зависимости t от N сортировки простой вставки

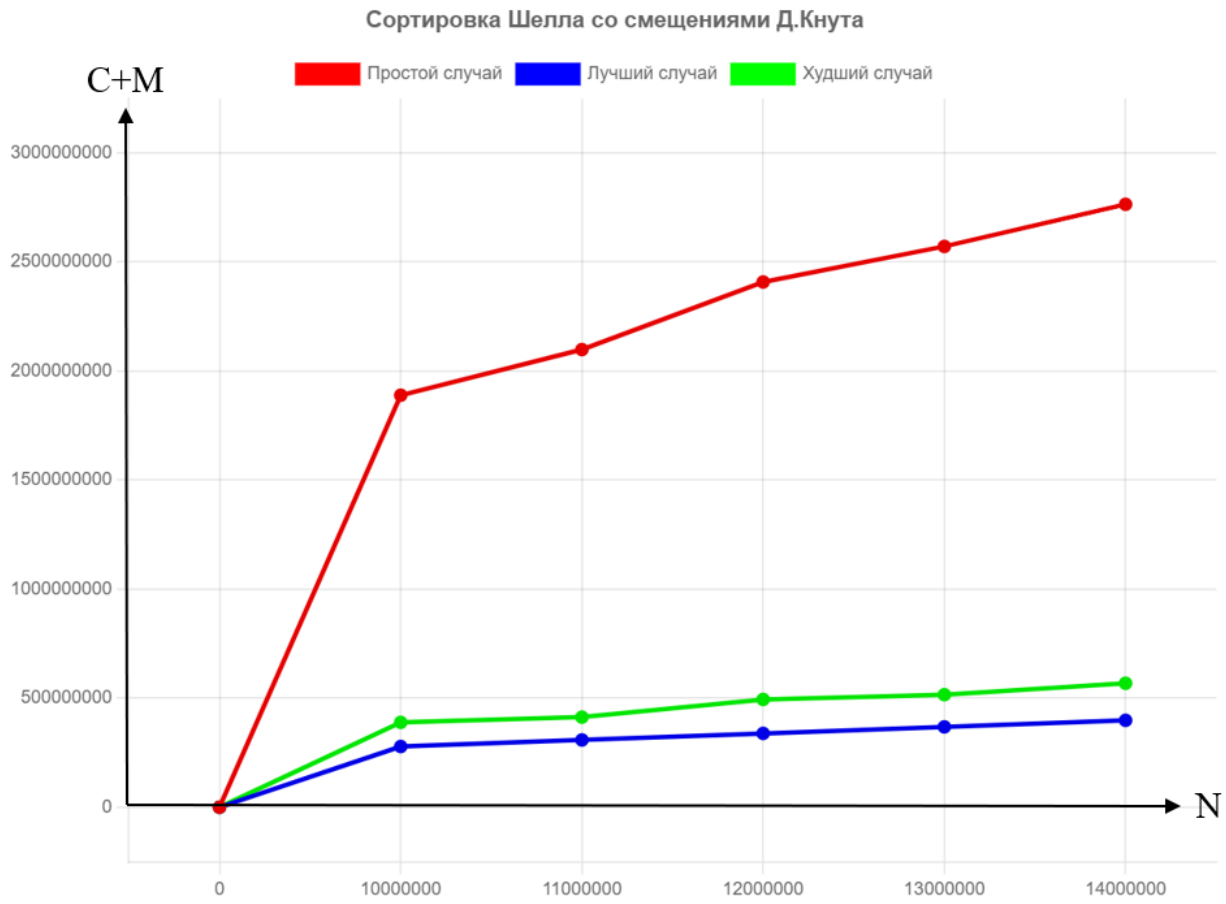

Рисунок 4. График зависимости C+M от N сортировки Шелла со смещениями Д.Кнута

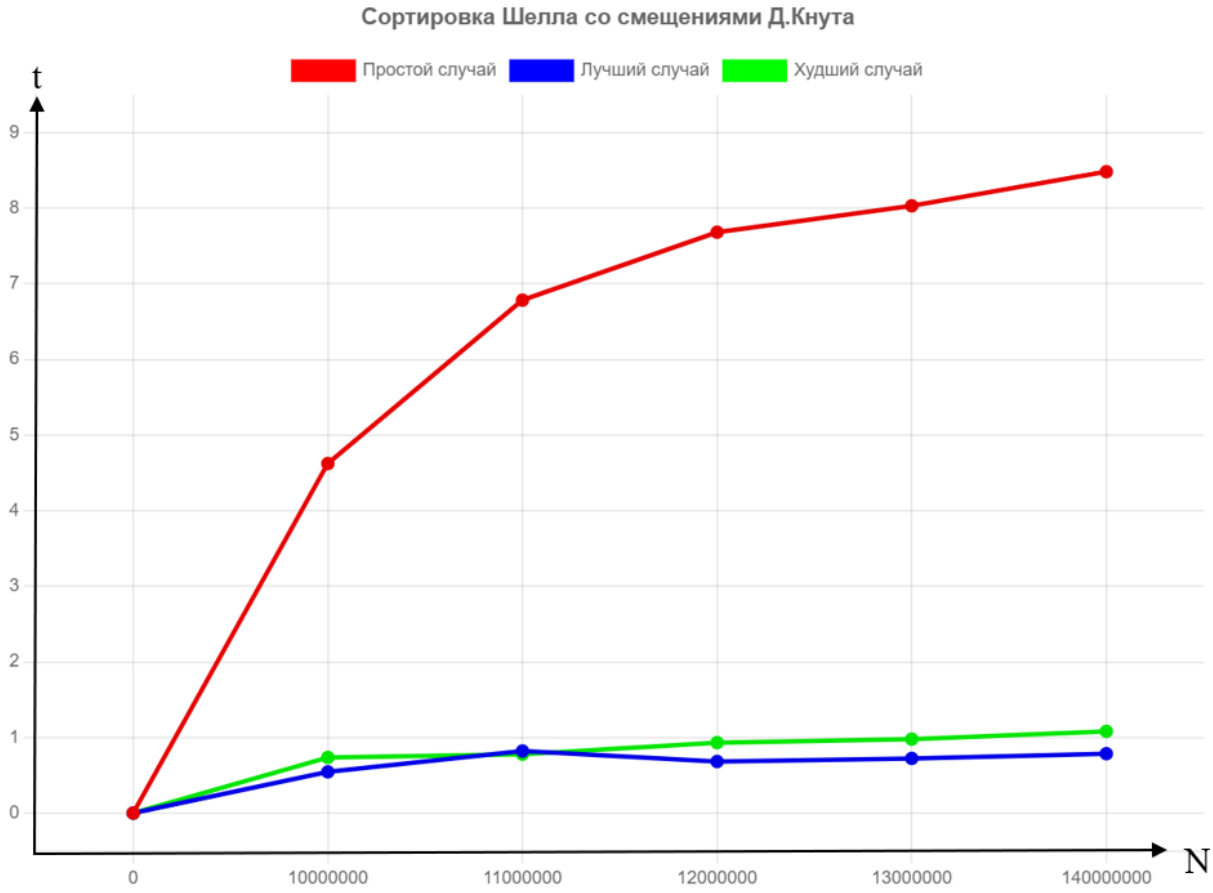

Рисунок 5. График зависимости t от N сортировки Шелла со смещениями Д.Кнута

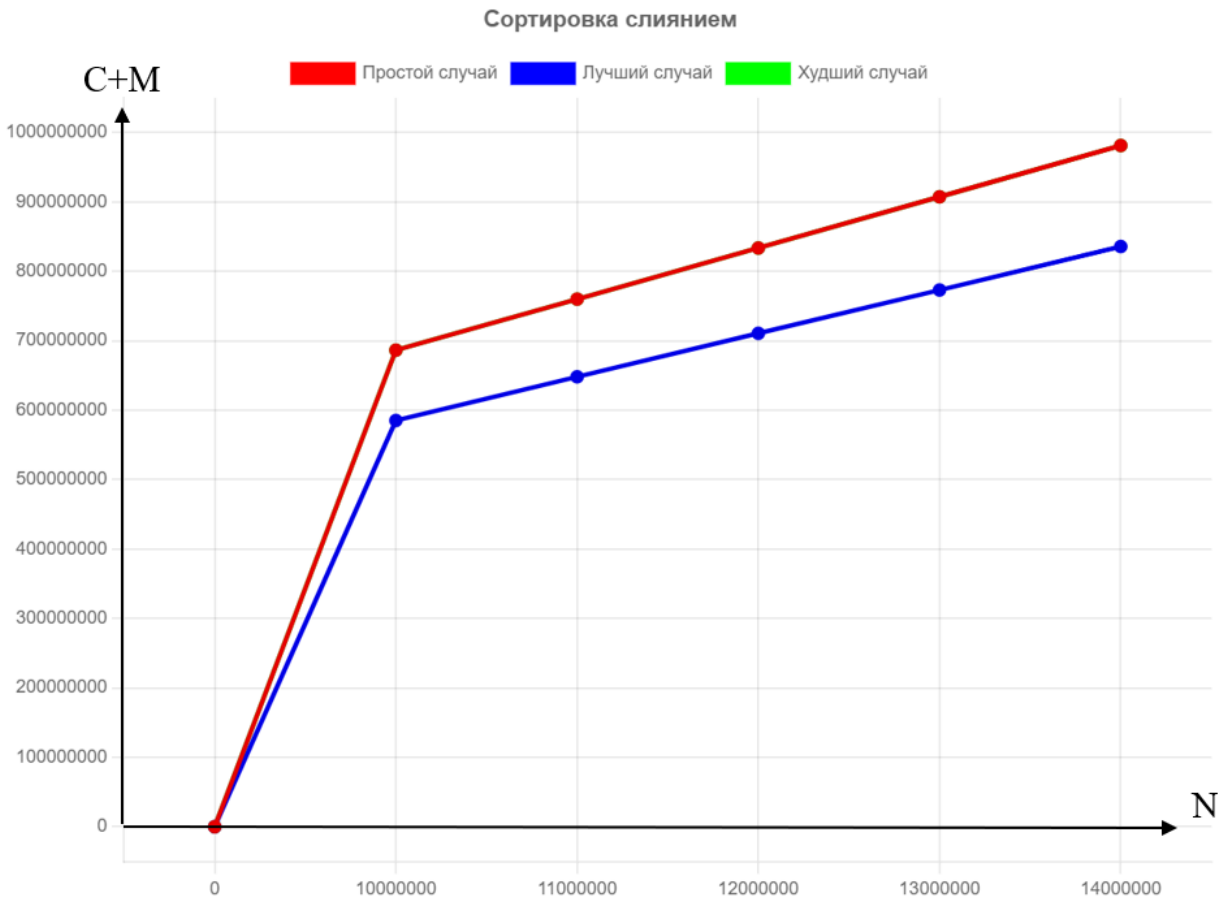

Рисунок 6. График зависимости C+M от N сортировки слиянием

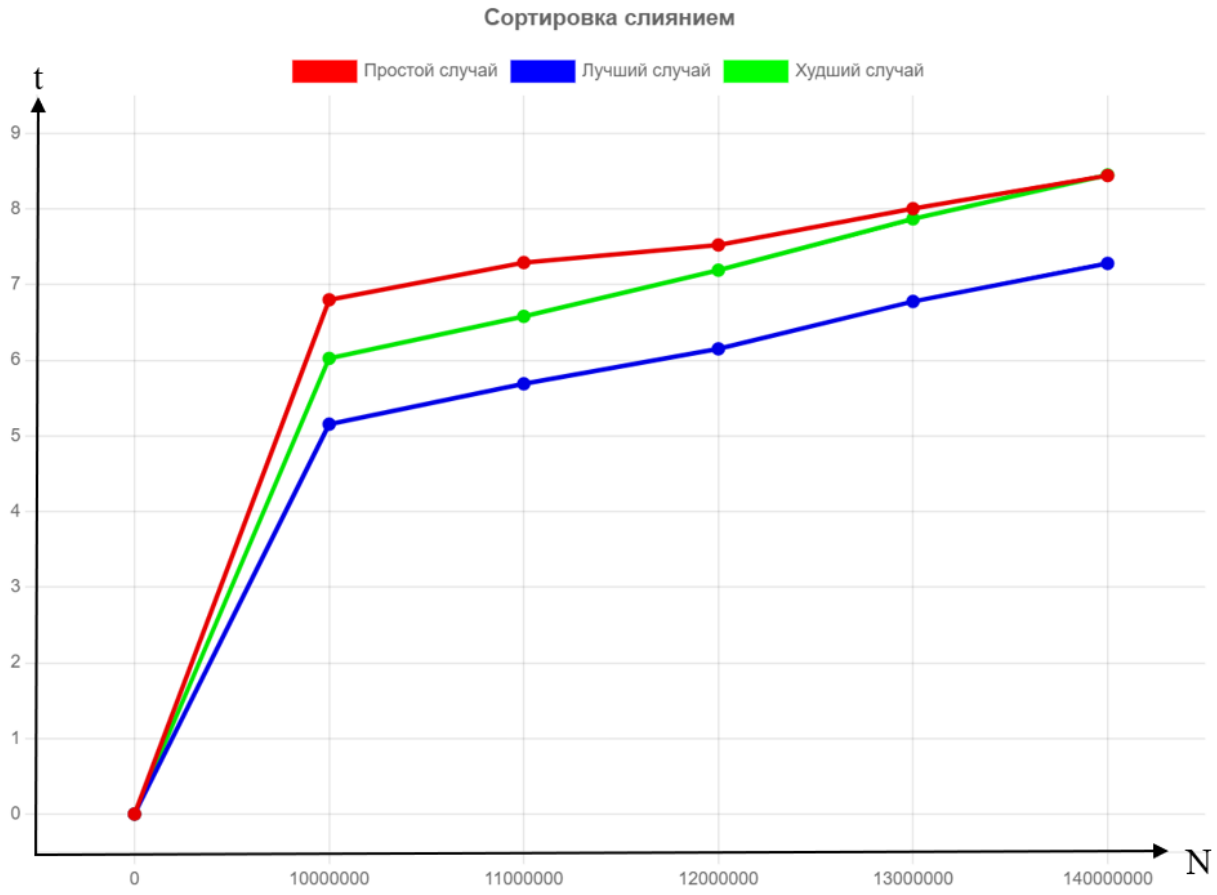

Рисунок 7. График зависимости t от N сортировки слиянием

На основе данных графиков и таблиц с результатами измерений можно сделать вывод, что алгоритм сортировки Шелла со смещениями Д.Кнута является наиболее эффективным алгоритмом.

5. Вывод

В результате работы я:

1. Получил навыки по анализу вычислительной сложности алгоритмов сортировки
2. Научился реализовывать алгоритмы сортировки на языке программирования C++

6. Исходный код программы

```
#include <iostream> //Библиотека для ввода/вывода в консоль
#include <random>
#include <chrono>
using namespace std; //Пространство имён std
```

```
void fillarr(int* arr, long long int N) { //Функция заполнения массива
      for (long long int i = 0; i < N; i++) {
            arr[i] = rand() % 100000000; //Заполняем массив случайными
числами
      }
}

void copyarr(int* arr, long long int N, int* arr2) { //Функция копирования
массива
    for (long long int i = 0; i < N; i++) {
        arr2[i] = arr[i]; //Заполняем другой массив такими же элементами
    }
}

void reversearr(int* arr, long long int N) { //Функция разворота масства
    for (long long int i = 0; i < N; i++) {
        for (long long int j = 0; j < N - 1 - i; j++) {
            swap(arr[j], arr[j + 1]);
        }
    }
}

void insertion_sort(int* arr, long long int N, long long int& C, long long
int& M) { //Сортировка простой вставки
    int key = 0, j = 0;
    for (int i = 1; i < N; i++)
    {
        key = arr[i];
        int l = 0, r = i - 1;
        while (l <= r) {
            int m = (l + r) / 2;
            if (arr[m] < key) {
                l = m + 1;
                C++;
            }
            else {
                r = m - 1;
                C++;
            }
        }
        for (j = i - 1; j >= l; j--) {
            arr[j + 1] = arr[j];
            M++;
        }
        arr[l] = key;
        M++;
    }
}

void merge(int* arr, long long int l, long long int m, long long int r, long
long int& C, long long int& M) { //Сортировка простым слиянием
    long long int i, j, k, n1 = m - l + 1, n2 = r - m;
    int* L = new int[n1];
    int* R = new int[n2];

    for (i = 0; i < n1; i++) {
        L[i] = arr[l + i];
        M++;
    }
    for (j = 0; j < n2; j++) {
        R[j] = arr[m + 1 + j];
        M++;
    }
```

```
    i = 0;
    j = 0;
    k = l;
    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) {
            arr[k] = L[i];
            i++;
            M++;
        }
        else {
            arr[k] = R[j];
            j++;
            M++;
        }
        k++;
        C++;
    }

    while (i < n1) {
        arr[k] = L[i];
        i++;
        k++;
        M++;
    }

    while (j < n2) {
        arr[k] = R[j];
        j++;
        k++;
        M++;
    }
    delete[] R;
    delete[] L;
}

void mergeSort(int* arr, long long int l, long long int r, long long int& C,
long long int& M) {
    if (l < r) {
        long long int m = l + (r - l) / 2;
        mergeSort(arr, l, m, C, M);
        mergeSort(arr, m + 1, r, C, M);
        merge(arr, l, m, r, C, M);
    }
}

void shellSort(int* arr, long long int N, long long int& C, long long int&
M) { //Сортировка Шелла со смещением Д. Кнута
    // Вычисляем начальное значение смещения
    int gap = 1;
    while (gap < N / 3) {
        gap = gap * 3 + 1;
    }

    // Сортируем массив для каждого значения смещения
    while (gap > 0) {
        // Применяем сортировку вставками для каждого подмассива
        for (int i = gap; i < N; i++) {
            int temp = arr[i];
            int j = i;
            C++;
            while (j >= gap && arr[j - gap] > temp) {
                arr[j] = arr[j - gap];
                j -= gap;
                M++;
```

```
                C++;
            }

            arr[j] = temp;
            M++;
        }

        // Уменьшаем смещение
        gap = (gap - 1) / 3;
    }
}

int main() {
    setlocale(LC_ALL, "Rus"); // Установка русского языка для вывода в
консоль
      long long int N;
      cout << "Введите количество элементов массива: "; cin >> N;
      int* arr = new int[N];
    int* arr2 = new int[N];
    int* arr3 = new int[N];
    fillarr(arr, N);
    copyarr(arr, N, arr2);
    copyarr(arr, N, arr3);

    long long int C1 = 0, M1 = 0, C2 = 0, M2 = 0, C3 = 0, M3 = 0;
    //avg
    auto start = std::chrono::steady_clock::now();
    insertion_sort(arr, N, C1, M1);
    auto end = std::chrono::steady_clock::now();
    std::chrono::duration<double> elapsed_seconds1 = end - start;

    start = std::chrono::steady_clock::now();
    shellSort(arr2, N, C2, M2);
    end = std::chrono::steady_clock::now();
    std::chrono::duration<double> elapsed_seconds2 = end - start;

    start = std::chrono::steady_clock::now();
    mergeSort(arr3, 0, N-1, C3, M3);
    end = std::chrono::steady_clock::now();
    std::chrono::duration<double> elapsed_seconds3 = end - start;

    cout << "Сорт. Вставка   (простой случай): C = " << C1 << "; M = " << M1
<< "; t = " << elapsed_seconds1.count() << endl;
    cout << "Сорт. Шелла     (простой случай): C = " << C2 << "; M = " << M2
<< "; t = " << elapsed_seconds2.count() << endl;
    cout << "Сорт. слиянием  (простой случай): C = " << C3 << "; M = " << M3
<< "; t = " << elapsed_seconds3.count() << endl << endl;

    C1 = C2 = C3 = M1 = M2 = M3 = 0;
    //BEST
    start = std::chrono::steady_clock::now();
    insertion_sort(arr, N, C1, M1);
    end = std::chrono::steady_clock::now();
    elapsed_seconds1 = end - start;

    start = std::chrono::steady_clock::now();
    shellSort(arr2, N, C2, M2);
    end = std::chrono::steady_clock::now();
    elapsed_seconds2 = end - start;

    start = std::chrono::steady_clock::now();
    mergeSort(arr3, 0, N-1, C3, M3);
    end = std::chrono::steady_clock::now();
    elapsed_seconds3 = end - start;
```

```
    cout << "Сорт. Вставка   (лучший случай): C = " << C1 << "; M = " << M1
<< "; t = " << elapsed_seconds1.count() << endl;
    cout << "Сорт. Шелла     (лучший случай): C = " << C2 << "; M = " << M2
<< "; t = " << elapsed_seconds2.count() << endl;
    cout << "Сорт. слиянием  (лучший случай): C = " << C3 << "; M = " << M3
<< "; t = " << elapsed_seconds3.count() << endl << endl;

    C1 = C2 = C3 = M1 = M2 = M3 = 0;

    reversearr(arr, N);
    copyarr(arr, N, arr2);
    copyarr(arr, N, arr3);
    //worst
    start = std::chrono::steady_clock::now();
    insertion_sort(arr, N, C1, M1);
    end = std::chrono::steady_clock::now();
    elapsed_seconds1 = end - start;

    start = std::chrono::steady_clock::now();
    shellSort(arr2, N, C2, M2);
    end = std::chrono::steady_clock::now();
    elapsed_seconds2 = end - start;

    start = std::chrono::steady_clock::now();
    mergeSort(arr3, 0, N-1, C3, M3);
    end = std::chrono::steady_clock::now();
    elapsed_seconds3 = end - start;

    cout << "Сорт. Вставка   (худший случай): C = " << C1 << "; M = " << M1
<< "; t = " << elapsed_seconds1.count() << endl;
    cout << "Сорт. Шелла     (худший случай): C = " << C2 << "; M = " << M2
<< "; t = " << elapsed_seconds2.count() << endl;
    cout << "Сорт. слиянием  (худший случай): C = " << C3 << "; M = " << M3
<< "; t = " << elapsed_seconds3.count() << endl << endl;

    delete[] arr;
    delete[] arr2;
    delete[] arr3;
    return 0;
}
```