МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. Шухова»**

**(БГТУ им. В. Г. Шухова)**

Кафедра программного обеспечения вычислительной

техники и автоматизированных систем

**Лабораторная работа №19**

по дисциплине: «Основы программирования»

по теме: «Оценка сложности алгоритмов сортировки по времени»

Выполнил/a: ст. группы ВТ-231

Петрухин Яков Ильич

Проверили:

Черников Сергей Викторович

Новожен Никита Викторович

Белгород, 2023 г.

**Цель работы:** решение задач на строки в стиле С.

**Содержание работы:**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Тип сортировки | 10000 | 20000 | 30000 | 40000 | 50000 | 60000 | 70000 | 80000 | 90000 | 100000 | 𝑇(𝑁) |
| Пузырьком | 0.108 | 0.558 | 1.221 | 1.9 | 3.17 | 4.477 | 5.868 | 7.651 | 9.61 | 12.492 | 𝑇(𝑁) = 1.21934117546141e-9\*x^2 |
| Выбором | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 𝑇(𝑁) = 0\*x^2 |
| Вставками | 0.091 | 0.425 | 0.688 | 1.182 | 1.948 | 3.3 | 4.014 | 5.018 | 6.617 | 8.297 | 𝑇(𝑁) = 8.20238424197068e-10\*x^2 |

Упорядоченные массивы:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Тип сортировки | 500000 | 1000000 | 1500000 | 2000000 | 2500000 | 3000000 | 3500000 | 4000000 | 4500000 | 5000000 | 𝑇(𝑁) |
| Расческой | 0.235 | 0.503 | 0.84 | 1.086 | 1.383 | 1.779 | 2.097 | 2.569 | 2.818 | 3.071 | 𝑇(𝑁) =  1.98062448219547e-13\*x^2 |
| Шелла | 0.019 | 0.041 | 0.063 | 0.085 | 0.113 | 0.15 | 0.159 | 0.189 | 0.234 | 0.26 | 𝑇(𝑁) =  5.34424242424243e-08 \* x + 0.0156666666666667 |
| Цифровая | 0.019 | 0.038 | 0.06 | 0.075 | 0.102 | 0.131 | 0.141 | 0.167 | 0.195 | 0.206 | 𝑇(𝑁) =  4.29575757575757e-08 \* x + 0.00473333333333333 |

Упорядоченные в обратном порядке массивы:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Тип сортировки | 10000 | 20000 | 30000 | 40000 | 50000 | 60000 | 70000 | 80000 | 90000 | 100000 | 𝑇(𝑁) |
| Пузырьком | 0.109 | 0.383 | 1.061 | 1.776 | 2.47 | 3.696 | 5.261 | 6.985 | 8.542 | 11.043 | 𝑇(𝑁) = 1.07979433943205e-9\*x^2 |
| Выбором | 0.119 | 0.469 | 1.051 | 1.704 | 2.65 | 3.921 | 5.349 | 7.456 | 8.823 | 11.895 | 𝑇(𝑁) = 1.14063553483411e-9\*x^2 |
| Вставками | 0.102 | 0.41 | 0.699 | 1.25 | 1.984 | 3.078 | 4.387 | 5.387 | 6.887 | 8.876 | 𝑇(𝑁) = 8.65913630420913e-10\*x^2 |
| Расческой | 0.01 | 0.008 | 0.011 | 0.015 | 0.019 | 0.028 | 0.028 | 0.029 | 0.039 | 0.044 | 𝑇(𝑁) = 4.99388150119243e-12\*x^2 |
| Шелла | 0.001 | 0.002 | 0.003 | 0.003 | 0.004 | 0.004 | 0.006 | 0.006 | 0.006 | 0.009 | 𝑇(𝑁) = 7.51515151515151e-08 \* x  - 0.000266666666666669 |
| Цифровая | 0.001 | 0.001 | 0.001 | 0.002 | 0.003 | 0.003 | 0.003 | 0.004 | 0.005 | 0.006 | 𝑇(𝑁) = 5.51515151515151E-08 \* x  + 0.000133333333333332 |

Неупорядоченные массивы:

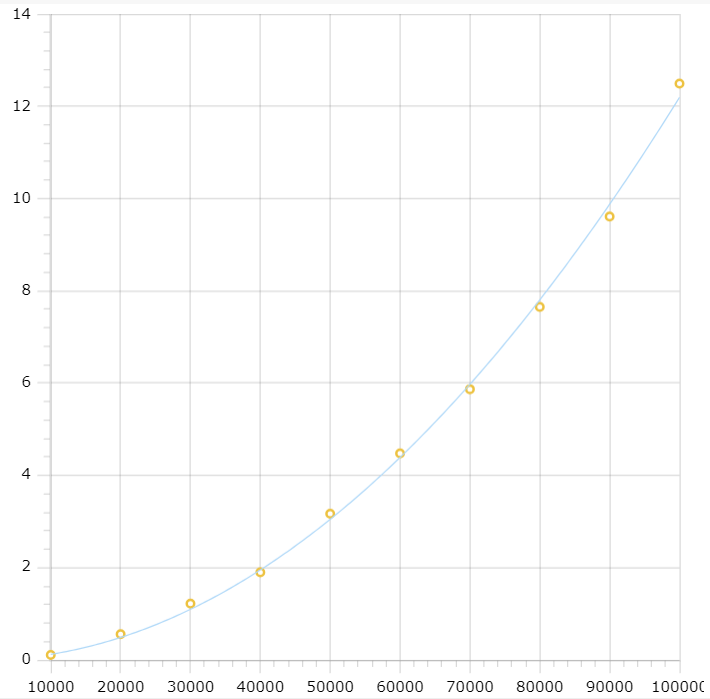
|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Тип сортировки | 10000 | 20000 | 30000 | 40000 | 50000 | 60000 | 70000 | 80000 | 90000 | 100000 | 𝑇(𝑁) |
| Пузырьком | 0.108 | 0.616 | 1.123 | 1.579 | 3.219 | 3.689 | 5.484 | 7.281 | 8.62 | 11.433 | 𝑇(𝑁) = 1.11611021200745e-9\*x^2 |
| Выбором | 0.066 | 0.208 | 0.568 | 0.863 | 1.251 | 1.881 | 2.784 | 3.492 | 5.26 | 6.318 | 𝑇(𝑁) = 6.06549954706081e-10\*x^2 |
| Вставками | 0.101 | 0.353 | 0.701 | 1.187 | 1.982 | 2.708 | 3.912 | 5.429 | 6.585 | 8.554 | 𝑇(𝑁) = 8.29661311298475e-10\*x^2 |
| Расческой | 0.005 | 0.011 | 0.013 | 0.017 | 0.022 | 0.029 | 0.038 | 0.038 | 0.052 | 0.053 | 𝑇(𝑁) = 6.25192436305112e-12\*x^2 |
| Шелла | 0.004 | 0.006 | 0.006 | 0.01 | 0.012 | 0.014 | 0.018 | 0.02 | 0.024 | 0.027 | 𝑇(𝑁) = 0.00000026 \* x + 0.0002 |
| Цифровая | 0.001 | 0.001 | 0.002 | 0.002 | 0.003 | 0.003 | 0.004 | 0.004 | 0.005 | 0.005 | 𝑇(𝑁) = 4.84848484848485E-08 \* x  - 0.000333333333333333 |

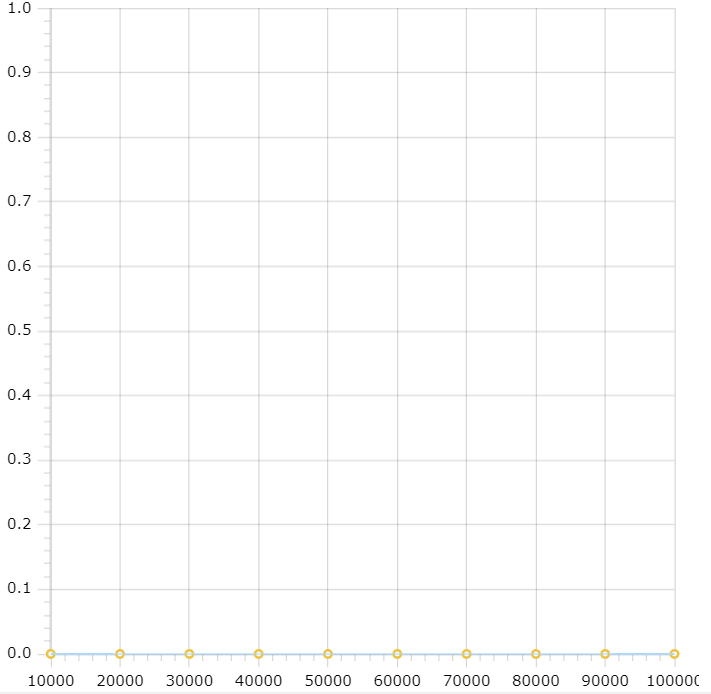
Таблицa зависимостей сортировок:

|  |  |  |  |
| --- | --- | --- | --- |
| Тип Сортировки | Упорядоченный (1) | В обратном порядке (2) | Неупорядоченный (3) |
| Пузырьком | 𝑇(𝑁) = 1.21934117546141e-9\*x^2 | 𝑇(𝑁) = 1.07979433943205e-9\*x^2 | 𝑇(𝑁) =  1.11611021200745e-9\*x^2 |
| Выбором | 𝑇(𝑁) = 0\*x^2 | 𝑇(𝑁) = 1.14063553483411e-9\*x^2 | 𝑇(𝑁) =  6.06549954706081e-10\*x^2 |
| Вставками | 𝑇(𝑁) = 8.20238424197068e-10\*x^2 | 𝑇(𝑁) = 8.65913630420913e-10\*x^2 | 𝑇(𝑁) = 8.29661311298475e-10\*x^2 |
| Расческой | 𝑇(𝑁) = 1.98062448219547e-13\*x^2 | 𝑇(𝑁) = 4.99388150119243e-12\*x^2 | 𝑇(𝑁) = 6.25192436305112e-12\*x^2 |
| Шелла | 𝑇(𝑁) = 5.34424242424243E-08 \* x + 0.0156666666666667 | 𝑇(𝑁) = 7.51515151515151e-08 \* x - 0.000266666666666669 | 𝑇(𝑁) = 0.00000026 \* x + 0.0002 |
| Цифровая | 𝑇(𝑁) = 4.29575757575757E-08 \* x + 0.00473333333333333 | 𝑇(𝑁) = 5.51515151515151E-08 \* x + 0.000133333333333332 | 𝑇(𝑁) =  4.8484848485e-08 \* x - 0.0003333333 |

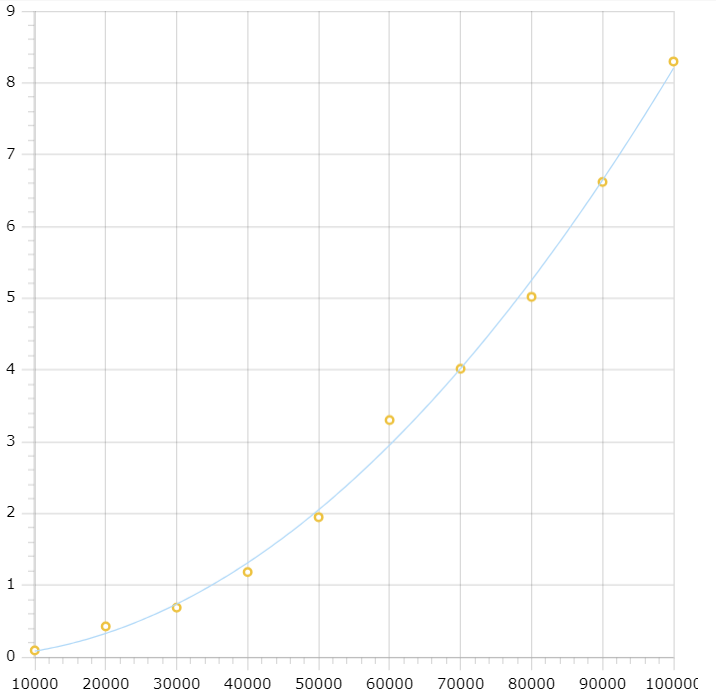
Графики зависимостей:

Пузырьковая сортировка:

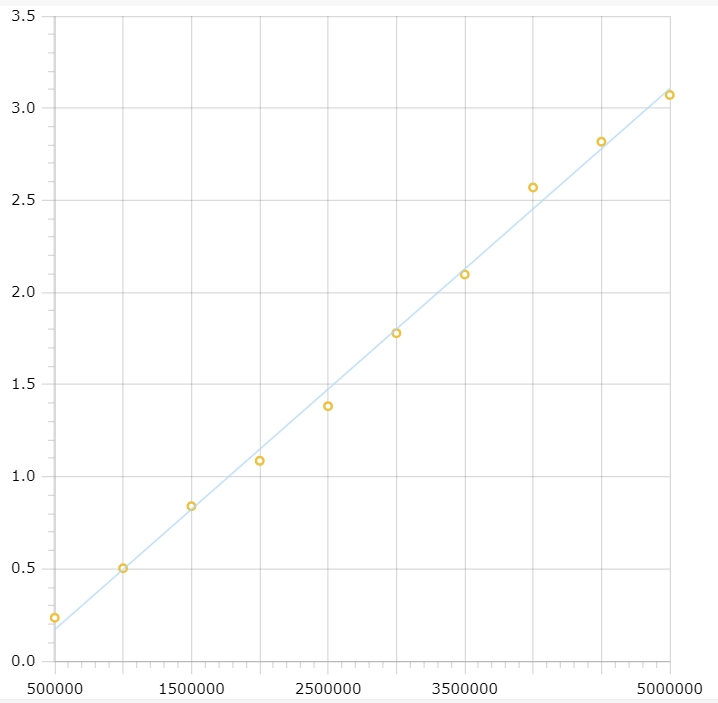


Сортировка выбором:  


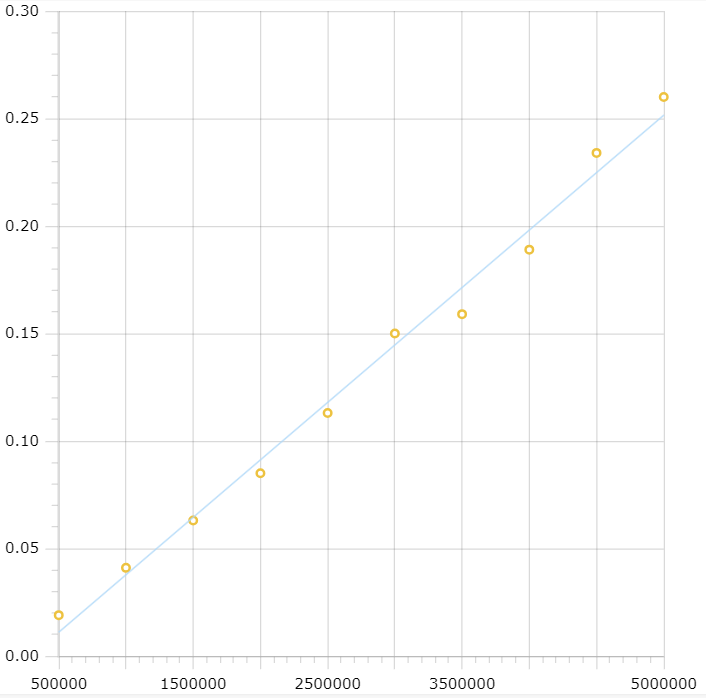
Сортировка вставкой:



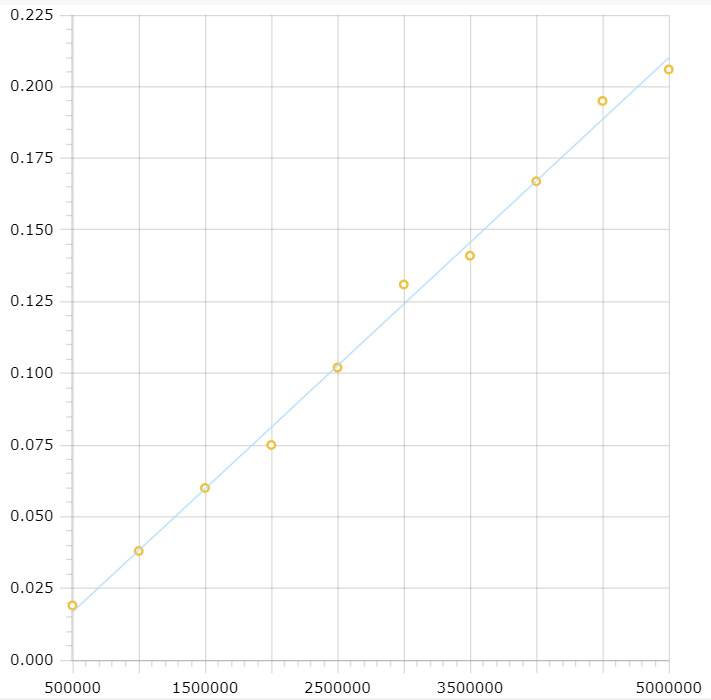
Сортировка расческой:



Сортировка Шелла:



Цифровая сортировка:



Исходный код:

Main.c:

#include <stdlib.h>

#include <stdio.h>

#include <stdbool.h>

#include <time.h>

#include <stdint.h>

#include <string.h>

#define MAX\_BUFFER\_SIZE    5000001

#define TIME\_TEST(testCode, time) \

{ \

    clock\_t start\_time = clock(); \

    testCode \

    clock\_t end\_time = clock();\

    clock\_t sort\_time = end\_time - start\_time; \

    time = (double) sort\_time / CLOCKS\_PER\_SEC; \

}

#define ARRAY\_SIZE(a) sizeof(a) / sizeof(a[0])

typedef struct SortFunc

{

    void    (\*sort)(int \*a, size\_t n);  // указатель на функцию сортировки

    char    name[64];                   // имя сортировки, используемое при выводе

} SortFunc;

typedef struct SortFuncNComp

{

    long long    (\*sort)(int \*a, size\_t n);  // указатель на функцию сортировки

    char    name[64];                   // имя сортировки, используемое при выводе

} SortFuncNComp;

typedef struct GeneratingFunc

{

    void (\*generate)(int \*a, size\_t n); // указатель на функцию генерации последоват.

    char name[64];                      // имя генератора, используемое при выводе

} GeneratingFunc;

uint8\_t digit(int a, uint8\_t n)

{

    uint8\_t shift = n \* 8;

    return (uint8\_t)(a >> shift);

}

double getTime()

{

    clock\_t start\_time = clock();

        // фрагмент кода

        // время которого измеряется

    clock\_t end\_time = clock();

    clock\_t sort\_time = end\_time - start\_time;

    return (double) sort\_time / CLOCKS\_PER\_SEC;

}

bool isOrdered(int \*a, size\_t n)

{

    bool answ = true;

    int prev = a[0];

    for(size\_t i = 1; i < n; i++)

    {

        if(prev > a[i])

        {

            answ = false;

            break;

        }

        prev = a[i];

    }

    return answ;

}

void outputArray\_(int \*a, size\_t n)

{

    for(size\_t i = 0; i < n; i++)

    {

        printf("%d ", a[i]);

    }

    printf("\n");

}

void outputArrayHex(int \*a, size\_t n)

{

    for(size\_t i = 0; i < n; i++)

    {

        printf("%08X ", a[i]);

    }

    printf("\n");

}

void generateOrderedArray (int \*a, size\_t n)

{

    for(size\_t i = 0; i < n; i++)

    {

        \*a++ = i;

    }

}

void generateOrderedBackwards (int \*a, size\_t n)

{

    for(size\_t i = n; i > 0; i--)

    {

        \*a++ = i;

    }

}

void generateRandomArray (int \*a, size\_t n)

{

    srand(0);

    for(size\_t i = 0; i < n; i++)

    {

        \*a++ = rand();

    }

}

void bubbleSort (int \*a, size\_t n)

{

    for (size\_t i = 0; i < n; i++)

    {

        int min = a[i];

        for (size\_t j = i + 1; j < n; j++)

        {

            int \*a\_j = &a[j];

            if (min < \*a\_j)

            {

                min = \*a\_j;

            }

            int temp = min;

            min = \*a\_j;

            \*a\_j = temp;

        }

    }

}

void mySort (int \*a, size\_t n)

{

    int \*start = &a[0];

    int \*end = &a[n];

    int \*min;

    int \*start1;

    while (start < end)

    {

        // find min value in range start1 - end

        min = start;

        int \*start1 = start;

        while (start1 < end)

        {

            if (\*min > \*start1)

            {

                min = start1;

            }

            start1++;

        }

        // swap min to start

        if (min > start)

        {

            int temp = \*min;

            \*min = \*start;

            \*start = temp;

        }

        start++;

    }

}

void selectionSort (int \*a, size\_t n)

{

    for (int i = 1; i < n; i++)

    {

        int newElement = a[i];

        int location = i - 1;

        while(location >= 0 && a[location] > newElement)

        {

            a[location + 1] = a[location];

            location = location - 1;

        }

        a[location + 1] = newElement;

    }

}

void insertionSort (int \*a, size\_t n)

{

    for (int i = 0; i < n; i++)

    {

        int minPosition = i;

        for (int j = i + 1; j < n; j++)

        {

            if (a[minPosition] > a[j])

            {

                minPosition = j;

            }

        }

        int \*a\_min = &a[minPosition];

        int \*a\_cur = &a[i];

        int tmp = \*a\_min;

        \*a\_min = \*a\_cur;

        \*a\_cur = tmp;

    }

}

void brushSort (int \*a, size\_t n)

{

    float factor = 1.05;

    size\_t step = n - 1;

    while (step >= 1)

    {

        for (size\_t i = 0; i + step < n; i++)

        {

            int \*a\_i = &a[i];

            int \*a\_i\_step = &a[i + step];

            if (\*a\_i > \*a\_i\_step)

            {

                int temp = \*a\_i;

                \*a\_i = \*a\_i\_step;

                \*a\_i\_step = temp;

            }

        }

        step = (size\_t)((float)step/factor);

        //step--;

    }

}

void shellSort (int \*a, size\_t n)

{

    for (int s = n / 2; s > 0; s /= 2)

    {

        for (int i = s; i < n; ++i)

        {

            for (int j = i - s; j >= 0 && a[j] > a[j + s]; j -= s)

            {

                int temp = a[j];

                a[j] = a[j + s];

                a[j + s] = temp;

            }

        }

    }

}

void radixSort (int \*a, size\_t n)

{

    const size\_t len = sizeof(int);     // длина числа

    const size\_t radix = 256;           // количество чисел в разряде 2^8 // основание системы счисления

    size\_t basket\_len = n;

    uint8\_t d;

    if (basket\_len > 1500000)

    {

        basket\_len = 1500000;

    }

    int \*b = malloc(sizeof(int) \* radix),       // число чисел в корзине

        \*c = malloc(sizeof(int) \* radix \* basket\_len);   // корзины для сортировки radix

    for (size\_t i = 0; i < len; i++)  // перебираем все разряды, начиная с нулевого

    {

        memset (b, 0, sizeof(int) \* radix); // пустой массив корзин b = 0

        for (size\_t j = 0; j < n; j++)

        {

            int val = a[i];

            d = digit(val, i);              // получаем цифру, стоящую на текущем разряде в каждом числе массива

            int index = (d \* basket\_len) + b[d];     // индекс записи

            c[index] = val;                 // отправляем число в промежуточный массив в корзину, которая совпадает со значением этой цифры

            b[d]++;                         // добавляем число чисел в корзине

        }

        size\_t count = 0;

        // сложим обратно числа из корзин в массив a

        for (size\_t j = 0; j < radix; j++)

        {

            int k = 0;

            int bj = b[j];

            while (k < bj)

            {

                a[count++] = c[(j \* basket\_len) + k];

                k++;

            }

        }

    }

    free(b);

    free(c);

}

long long getBubbleSortNComp (int \*a, size\_t n)

{

    long long nComps = 0;

    for (size\_t i = 0; ++nComps && i < n; i++)

    {

        int min = a[i];

        for (size\_t j = i + 1; ++nComps && j < n; j++)

        {

            if (++nComps && min < a[j])

            {

                min = a[j];

            }

            int temp = min;

            min = a[j];

            a[j] = temp;

        }

    }

    return nComps;

}

long long getSelectionSortNComp (int \*a, size\_t n)

{

    long long nComps = 0;

    for (int i = 1; ++nComps && i < n; i++)

    {

        int newElement = a[i];

        int location = i - 1;

        while(++nComps && location >= 0 && a[location] > newElement)

        {

            a[location + 1] = a[location];

            location = location - 1;

        }

        a[location + 1] = newElement;

    }

    return nComps;

}

long long getInsertionSortNComp (int \*a, size\_t n)

{

    long long nComps = 0;

    for (int i = 0; ++nComps && i < n; i++)

    {

        int minPosition = i;

        for (int j = i + 1; ++nComps && j < n; j++)

        {

            if (++nComps && a[minPosition] > a[j])

            {

                minPosition = j;

            }

        }

        int \*a\_min = &a[minPosition];

        int \*a\_cur = &a[i];

        int tmp = \*a\_min;

        \*a\_min = \*a\_cur;

        \*a\_cur = tmp;

    }

    return nComps;

}

long long getBrushSortNComp (int \*a, size\_t n)

{

    float factor = 1.05;

    size\_t step = n - 1;

    long long nComps = 0;

    while (++nComps && step >= 1)

    {

        for (size\_t i = 0; ++nComps && i + step < n; i++)

        {

            int \*a\_i = &a[i];

            int \*a\_i\_step = &a[i + step];

            if (++nComps && \*a\_i > \*a\_i\_step)

            {

                int temp = \*a\_i;

                \*a\_i = \*a\_i\_step;

                \*a\_i\_step = temp;

            }

        }

        step = (size\_t)((float)step/factor);

        //step--;

    }

    return nComps;

}

long long getShellSortNComp (int \*a, size\_t n)

{

    long long nComps = 0;

    for (int s = n / 2; ++nComps && s > 0; s /= 2)

    {

        for (int i = s; ++nComps && i < n; ++i)

        {

            for (int j = i - s; ++nComps && j >= 0 && a[j] > a[j + s]; j -= s)

            {

                int temp = a[j];

                a[j] = a[j + s];

                a[j + s] = temp;

            }

        }

    }

    return nComps;

}

long long getRadixSortNComp (int \*a, size\_t n)

{

    const size\_t len = sizeof(int);     // длина числа

    const size\_t radix = 256;           // количество чисел в разряде 2^8 // основание системы счисления

    long long nComps = 0;

    size\_t basket\_len = n;

    uint8\_t d;

    if (basket\_len > 1500000)

    {

        basket\_len = 1500000;

    }

    int \*b = malloc(sizeof(int) \* radix),       // число чисел в корзине

        \*c = malloc(sizeof(int) \* radix \* basket\_len);   // корзины для сортировки radix

    for (size\_t i = 0; ++nComps && i < len; i++)  // перебираем все разряды, начиная с нулевого

    {

        memset (b, 0, sizeof(int) \* radix); // пустой массив корзин b = 0

        for (size\_t j = 0; ++nComps && j < n; j++)

        {

            int val = a[i];

            d = digit(val, i);              // получаем цифру, стоящую на текущем разряде в каждом числе массива

            int index = (d \* basket\_len) + b[d];     // индекс записи

            c[index] = val;                 // отправляем число в промежуточный массив в корзину, которая совпадает со значением этой цифры

            b[d]++;                         // добавляем число чисел в корзине

        }

        size\_t count = 0;

        // сложим обратно числа из корзин в массив a

        for (size\_t j = 0; ++nComps && j < radix; j++)

        {

            int k = 0;

            int bj = b[j];

            while (++nComps && k < bj)

            {

                a[count++] = c[(j \* basket\_len) + k];

                k++;

            }

        }

    }

    free(b);

    free(c);

    return nComps;

}

void checkTime (void(\*sortFunc)(int \*, size\_t),

                void (\*generateFunc)(int \*, size\_t),

                size\_t size,

                char \*experimentName)

{

    static size\_t runCounter = 1;

    // генерация последовательности

    static int innerBuffer[MAX\_BUFFER\_SIZE];

    generateFunc(innerBuffer, size);

    printf("Run #%zu| ", runCounter++);

    printf("Name: %s\n", experimentName);

    double time;

    TIME\_TEST(

    {

        sortFunc(innerBuffer, size);

    }, time);

    printf("Status: ");

    if (isOrdered(innerBuffer, size))

    {

        printf("OK! Time: %.3f s.\n", time);

        // запись в файл

        char filename[256];

        // sprintf(filename, "./data/%s.csv", experimentName);

        sprintf(filename, "%s.csv", experimentName);

        FILE \*f = fopen(filename, "a");

        if (f == NULL)

        {

            printf("FileOpenError %s", filename);

            exit(1);

        }

        fprintf(f, "%zu, %.3f\n", size, time);

        fclose(f);

    }

    else

    {

        printf("Wrong!\n");

        // вывод массива, который не смог быть отсортирован

        outputArrayHex(innerBuffer, size);

        exit(1);

    }

}

void checkTimeNComp (long long(\*sortFunc)(int \*, size\_t),

                    void (\*generateFunc)(int \*, size\_t),

                    size\_t size,

                    char \*experimentName)

{

    static size\_t runCounter = 1;

    // генерация последовательности

    static int innerBuffer[MAX\_BUFFER\_SIZE];

    generateFunc(innerBuffer, size);

    printf("Run #%zu| ", runCounter++);

    printf("Name: %s\n", experimentName);

    double time;

    long long nComp;

    TIME\_TEST(

    {

        nComp = sortFunc(innerBuffer, size);

    }, time);

    printf("Status: ");

    if (isOrdered(innerBuffer, size))

    {

        printf("OK! Time: %.3f s.\n", time);

        printf("Comps: %lld \n", nComp);

        // запись в файл

        char filename[256];

        // sprintf(filename, "./data/%s.csv", experimentName);

        sprintf(filename, "%s.csv", experimentName);

        FILE \*f = fopen(filename, "a");

        if (f == NULL)

        {

            printf("FileOpenError %s", filename);

            exit(1);

        }

        fprintf(f, "%zu, %.3f, %lld\n", size, time, nComp);

        fclose(f);

    }

    else

    {

        printf("Wrong!\n");

        // вывод массива, который не смог быть отсортирован

        outputArrayHex(innerBuffer, size);

        exit(1);

    }

}

void timeExperiment()

{

    // описание функций сортировки

    SortFunc sorts[] =

    {

        //{bubbleSort, "bubbleSort"},

        //{selectionSort, "selectionSort"},

        //{insertionSort, "insertionSort"},

        {brushSort, "brushSort"},

        {shellSort, "shellSort"},

        {radixSort, "radixSort"},

        //{mySort, "mySort"}

        // вы добавите свои сортировки

    };

    const unsigned FUNCS\_N = ARRAY\_SIZE(sorts);

    // описание функций генерации

    GeneratingFunc generatingFuncs[] =

    {

        // генерируется случайный массив

        {generateRandomArray, "random"},

        // генерируется массив 0, 1, 2, ..., n - 1

        {generateOrderedArray, "ordered"},

        // генерируется массив n - 1, n - 2, ..., 0

        {generateOrderedBackwards, "orderedBackwards"}

    };

    const unsigned CASES\_N = ARRAY\_SIZE(generatingFuncs);

    // запись статистики в файл up to MAX\_BUFFER\_SIZE

    for (size\_t size = 500000; size <= 5000000; size += 500000)

    {

        printf("------------------------------\n");

        printf("Size: %d\n", size);

        for (int i = 0; i < FUNCS\_N; i++)

        {

            for (int j = 0; j < CASES\_N; j++)

            {

                // генерация имени файла

                static char filename[128];

                sprintf (filename, "%s\_%s\_time", sorts[i].name, generatingFuncs[j].name);

                checkTime(sorts[i].sort, generatingFuncs[j].generate, size, filename);

            }

        }

        printf("\n");

    }

}

void timeExperimentNComp()

{

    // описание функций сортировки

    SortFuncNComp sorts[] =

    {

        {getBubbleSortNComp, "getBubbleSortNComp"},

        {getSelectionSortNComp, "getSelectionSortNComp"},

        {getInsertionSortNComp, "getInsertionSortNComp"},

        {getBrushSortNComp, "getBrushSortNComp"},

        {getShellSortNComp, "getShellSortNComp"},

        {getRadixSortNComp, "getRadixSortNComp"}

        // вы добавите свои сортировки

    };

    const unsigned FUNCS\_N = ARRAY\_SIZE(sorts);

    // описание функций генерации

    GeneratingFunc generatingFuncs[] =

    {

        // генерируется случайный массив

        {generateRandomArray, "random"},

        // генерируется массив 0, 1, 2, ..., n - 1

        {generateOrderedArray, "ordered"},

        // генерируется массив n - 1, n - 2, ..., 0

        {generateOrderedBackwards, "orderedBackwards"}

    };

    const unsigned CASES\_N = ARRAY\_SIZE(generatingFuncs);

    // запись статистики в файл up to MAX\_BUFFER\_SIZE

    for (size\_t size = 500; size <= 5000; size += 500)

    {

        printf("------------------------------\n");

        printf("Size: %d\n", size);

        for (int i = 0; i < FUNCS\_N; i++)

        {

            for (int j = 0; j < CASES\_N; j++)

            {

                // генерация имени файла

                static char filename[128];

                sprintf (filename, "%s\_%s\_time", sorts[i].name, generatingFuncs[j].name);

                checkTimeNComp(sorts[i].sort, generatingFuncs[j].generate, size, filename);

            }

        }

        printf("\n");

    }

}

int main()

{

    timeExperiment();

    //timeExperimentNComp();

    return 0;

}

Ссылка на репозиторий GitHub:  
<https://github.com/Frozzie/UniLabWork19>



Вопросы:  
1. При измерении времени выполнения, конечно же, влияет не только количество сравнений, но и операции обмена. Если бы вам было необходимо сортировать обменной сортировкой массив из структур большого размера (скажем, 8096 байт) – как можно было бы избежать “просадок” в скорости сортировки, не меняя алгоритм сортировки? Помните, что нам важно получить правильный порядок элементов, но для этого они сами не обязаны располагаться в правильном порядке.

Один из возможных вариантов состоит в том чтобы создать массив указателей на структуры и сортировать элементы, переставляя указатели на структуры.

2. Возьмите произвольную сортировку. Оцените, сколько времени потребуется ей для упорядочения массива из 𝑁 (𝑁 ≥ 200000) элементов. Все вычисления приложите к отчёту. Выполните запуск выбранной сортировки на вашей ЭВМ.

Оцените погрешность измерений (укажите отклонение в процентах).

Сортировка Шелла. Аппроксимирующая формула по методу наименьших квадратов

𝑇(𝑁) = 2.98787878787878e-07 \* x + 0.00173333333333333

Для N = 500000 элементов, время выполнения программы будет равно:

T = 151 мс.  
Экспериментальное значение Т(э) = 140 мс.

Погрешность аппроксимации равна:

D = ((T – T(э)) / T(э)) \* 100% = 7.86%

3. Если бы мы использовали сортировку вставками для сортировки двусвязного

линейного списка, а не массива, то от какого недостатка этого алгоритма мы

бы избавились, по сравнению с сортировкой массива?

Вместо того, чтобы копировать данные мы копируем их указатели.

4. Подумайте, можно ли применить вашу реализацию цифровой сортировки к

дробным числам? Что нужно изменить, чтобы она работала? Не обязательно

реализовывать что-то, главное - описать идею. Ответить на эти вопросы поможет стандарт IEEE-754.

Для сортировки дробных чисел, представленных в формате IEEE-754 можно использовать функции получения двоичных или десятичных цифр числа для сравнения.