МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«МИРЭА – Российский технологический университет»

РТУ МИРЭА

Отчет по выполнению практического задания №1.1

Тема: ЭМПИРИЧЕСКИЙ АНАЛИЗ СЛОЖНОСТИ АЛГОРИТМОВ

Дисциплина: Структуры и алгоритмы обработки данных

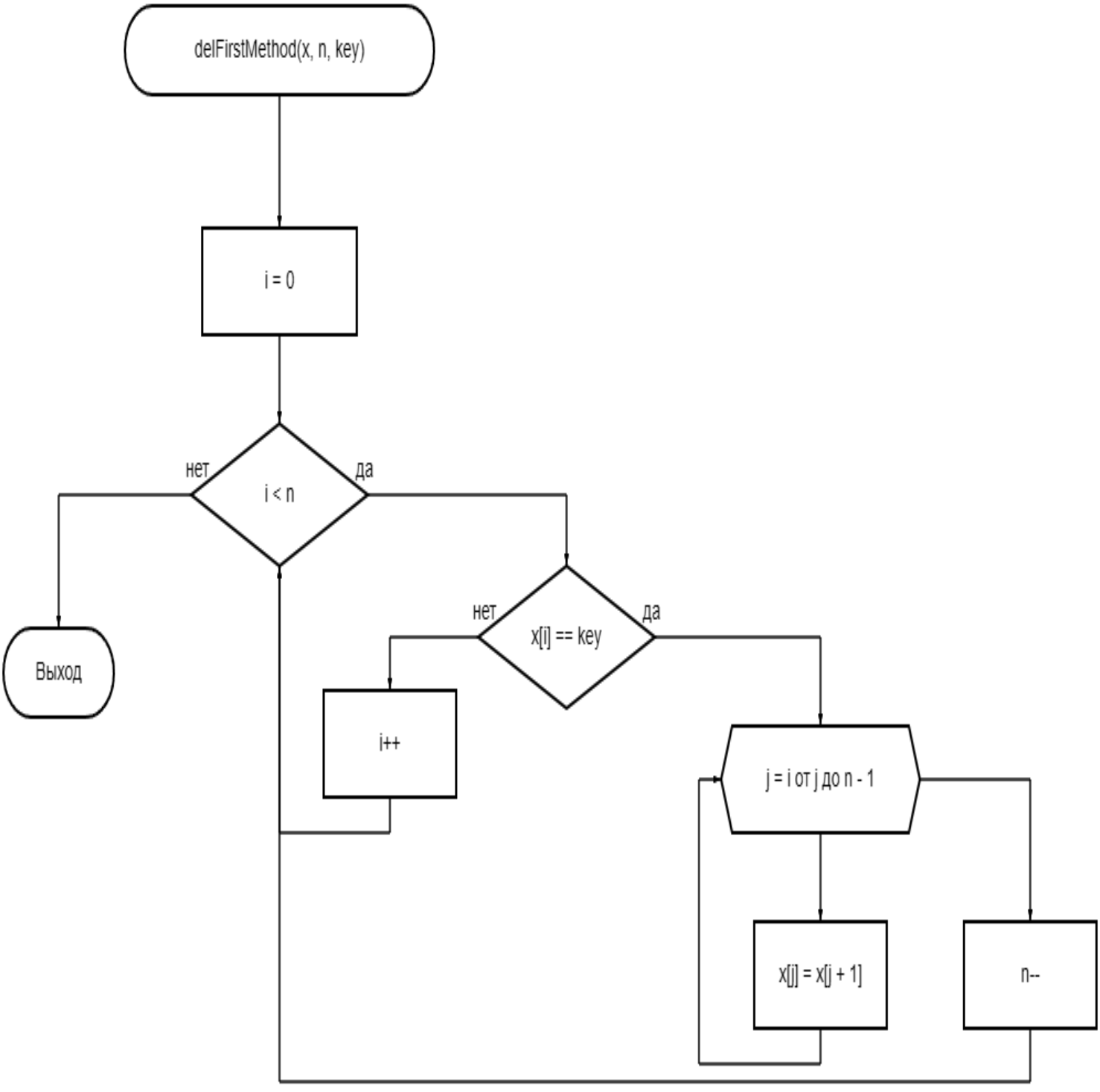
Выполнил студент student
группа group

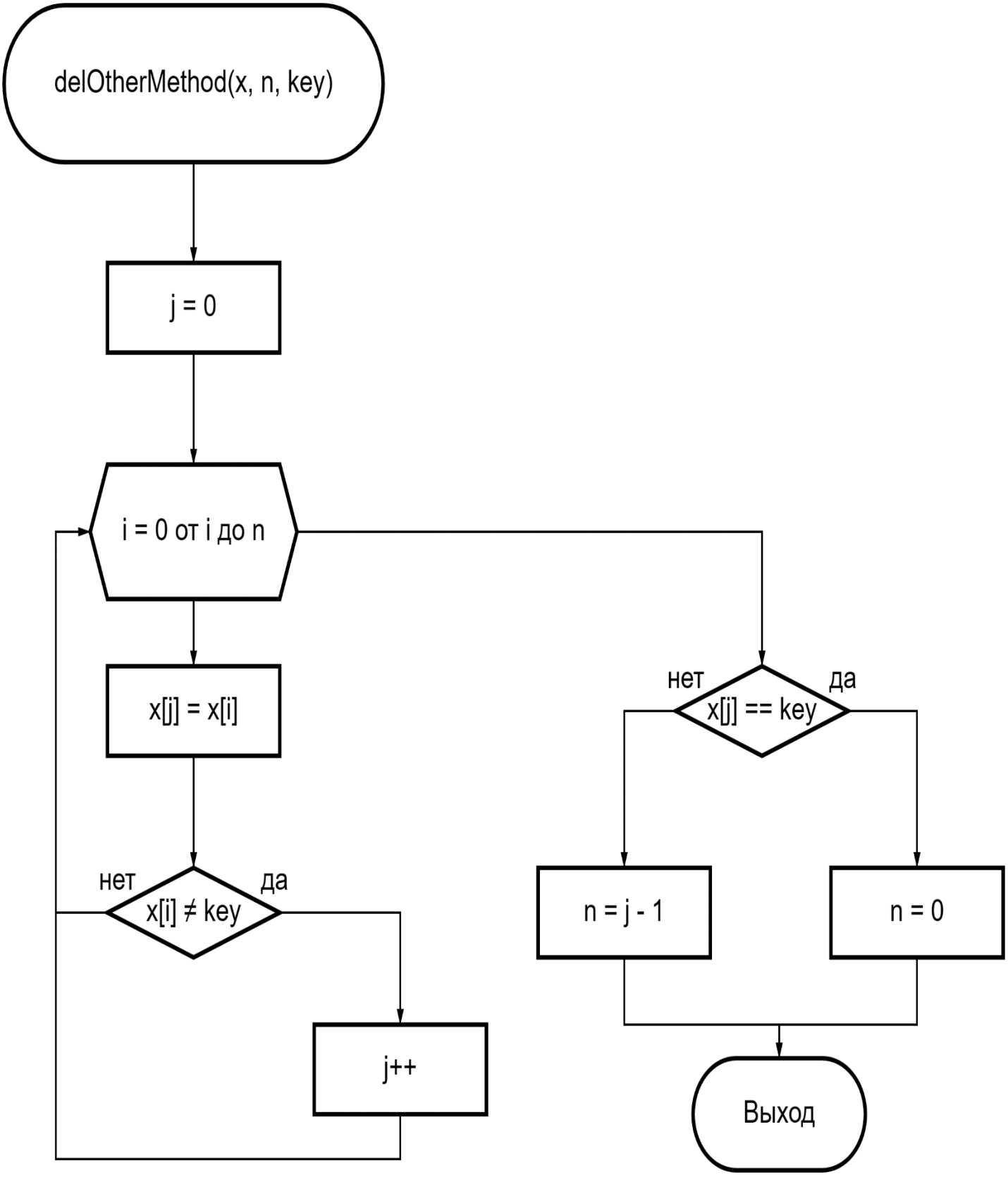
Москва 2025

Цель работы: актуализация знаний и приобретение практических умений и навыков по определению вычислительной сложности алгоритмов (эмпирический подход).

Задание 1

1.1 Блок-схемы алгоритмов




delOtherMethod(x, n, key)
j = 0
i = 0 от i до n
x[j] = x[i]
нет
x[i] ≠ key
да
j++
нет
x[j] == key
да
n = j - 1
n = 0
Выход

1.2 Реализация алгоритмов на C++

Код реализации первого алгоритма

```
void delFirstMethod(char x[], int n, char key)
{
    int i = 0;
    while (i < n)
    {
        if (x[i] == key)
        {
            for (int j = i; j < n - 1; j++)
            {
                x[j] = x[j + 1];
            }
            n--;
        }
        else
        {
            i++;
        }
    }
}
```

Код реализации второго алгоритма

```
void delOtherMethod(char x[], int n, char key)
{
    int j = 0;
    for (int i = 0; i < n; i++)
    {
        x[j] = x[i];
        if (x[i] != key)
        {
            j++;
        }
    }

    if (x[1] == key) n = 0;
    else n = j - 1;
}
```

1.3-1.4 Реализация дополнительных возможностей

```cpp
#include <iostream>
#include <vector>
#include <random>
#include <chrono>

enum class Case {
    WORST,
    AVERAGE,
    BEST
};

int n[11] { 100, 200, 500, 1000, 2000, 5000, 10000, 100000, 200000, 500000, 1000000 };

void copyArray(char* originArray, char* copyTo, int size)
{
    for (int i = 0; i < size; i++)
    {
        copyTo[i] = originArray[i];
    }
}

void fillArray(int size, Case _case, char* arr, char key)
{
    char letters[10] = {'A', 'B', 'C', 'D', 'E', 'F', 'D', 'H', 'I', 'G' };

    for (int i = 0; i < size; i++)
    {
        if (_case == Case::WORST)
        {
            arr[i] = key;
        }
        else if (_case == Case::AVERAGE)
        {
            if (rand() % 2 == 0)
            {
                arr[i] = key;
            }
            else
            {
                arr[i] = letters[rand() % 10];
            }
        }
        else if (_case == Case::BEST)
        {
            arr[i] = letters[rand() % 9];
        }
    }
}

void printArray(char* arr, int size)
{
    for (int i = 0; i < size; i++)
    {
        std::cout << arr[i] << " ";
    }
    std::cout << std::endl;
}

void delFirstMethod(char x[], int n, char key)
{
    unsigned long long counterC = 0;
    unsigned long long counterM = 0;

    auto start = std::chrono::high_resolution_clock::now();

    int i = 0;
    while (i < n)
    {
        if (x[i] == key)
        {
            for (int j = i; j < n - 1; j++)
            {
                x[j] = x[j + 1];
                counterM++;
            }
            n--;
        }
        else
        {
            i++;
        }
        counterC++;
    }

    auto end = std::chrono::high_resolution_clock::now();
    auto deltaTime = std::chrono::duration_cast<std::chrono::nanoseconds>(end - start);

    std::cout << "Sorted array: ";
    printArray(x, 10);
```

```
    std::cout << std::endl;

    std::cout << "Time elapsed: " << deltaTime.count() / 1'000'000.0 << " ms" << std::endl;
    std::cout << "M: " << counterM << std::endl
        << "C: " << counterC << std::endl
        << "T: " << counterM + counterC << std::endl;
}

void delOtherMethod(char x[], int n, char key)
{
    unsigned long long counterC = 0;
    unsigned long long counterM = 0;


    auto start = std::chrono::high_resolution_clock::now();

    int j = 0;
    for (int i = 0; i < n; i++)
    {
        x[j] = x[i];
        counterM++;
        if (x[i] != key)
        {
            j++;
        }
        counterC++;
    }

    if (x[1] == key) n = 0;
    else n = j - 1;
    counterC++;

    auto end = std::chrono::high_resolution_clock::now();
    auto deltaTime = std::chrono::duration_cast<std::chrono::nanoseconds>(end - start);

    std::cout << "Sorted array: ";
    printArray(x, 10);
    std::cout << std::endl;

    std::cout << std::endl << "Time elapsed: " << deltaTime.count() / 1'000'000.0 << " ms" << std::endl;
    std::cout << "M: " << counterM << std::endl <<
        "C: " << counterC << std::endl <<
        "T: " << counterM + counterC << std::endl;
}
int main()
{
    srand(time(NULL));

    char key;
    std::cin >> key;

    for (int size : n)
    {
        char* arr = new char[size];
        char* arr2 = new char[size];

        fillArray(size, Case::WORST, arr, key);
        copyArray(arr, arr2, size);

        std::cout << "--------------------" << std::endl;
        std::cout << "n = " << size << std::endl;

        std::cout << "First algorithm" << std::endl;
        std::cout << "Origin array: ";
        printArray(arr, 10);
        delFirstMethod(arr, size, key);

        std::cout << std::endl;

        std::cout << "Second algorithm" << std::endl;
        std::cout << "Origin array: ";
        printArray(arr2, 10);
        delOtherMethod(arr2, size, key);

        delete[] arr;
        delete[] arr2;
    }

    return 0;
}
```

1.5 Тест алгоритма при n = 100, 200 … 10000 в трёх случаях

а) все элементы должны быть удалены

Худший случай (1 алгоритм)				
n	Время, мс	Мп	Сп	Тп
100	0.0066	4950	100	5050
200	0.0251	19900	200	20100
500	0.183	124750	500	125250
1000	0.5937	499500	1000	500500
2000	2.6119	1999000	2000	2001000
5000	15.3418	12497500	5000	12502500
10000	59.8348	49995000	10000	50005000

Худший случай (2 алгоритм)				
n	Время, мс	Мп	Сп	Тп
100	0.0003	100	100	200
200	0.0004	200	200	400
500	0.0009	500	500	1000
1000	0.0018	1000	1000	2000
2000	0.0043	2000	2000	4000
5000	0.0086	5000	5000	10000
10000	0.0206	10000	10000	20000

б) случайное заполнение

Средний случай (1 алгоритм)				
n	Время, мс	Мп	Сп	Тп
100	0.0032	2737	100	2837
200	0.0117	10105	200	10305
500	0.0929	59573	500	60073
1000	0.3446	256291	1000	257291
2000	1.2518	990935	2000	992935
5000	7.2221	6256506	5000	6261506
10000	29.6154	25280580	10000	25290580

Средний случай (2 алгоритм)				
n	Время, мс	Мп	Сп	Тп
100	0.0006	100	100	200
200	0.0014	200	200	400
500	0.0031	500	500	1000
1000	0.0065	1000	1000	2000
2000	0.0123	2000	2000	4000
5000	0.0307	5000	5000	10000
10000	0.0617	10000	10000	20000

в) ни один элемент не удаляется

Лучший случай (1 алгоритм)				
n	Время, мс	Мп	Сп	Тп
100	0.0003	0	100	100
200	0.0003	0	200	200
500	0.0008	0	500	500
1000	0.0012	0	1000	1000
2000	0.0027	0	2000	2000
5000	0.0053	0	5000	5000
10000	0.0104	0	10000	10000

Лучший случай (2 алгоритм)				
n	Время, мс	Мп	Сп	Тп
100	0.0003	100	100	200
200	0.0005	200	200	400
500	0.0011	500	500	1000
1000	0.0021	1000	1000	2000
2000	0.0049	2000	2000	4000
5000	0.0124	5000	5000	10000
10000	0.0254	10000	10000	20000

1.6 Результаты эмпирического исследования

Худший случай:

Худший случай (1 алгоритм)				
n	Время, мс	Мп	Сп	Тп
100	0.0066	4950	100	5050
200	0.0251	19900	200	20100
500	0.183	124750	500	125250
1000	0.5937	499500	1000	500500
2000	2.6119	1999000	2000	2001000
5000	15.3418	12497500	5000	12502500
10000	59.8348	49995000	10000	50005000
100000	5920.43	4999950000	100000	5000050000
200000	23556.8	1999990000	200000	2000190000
500000	148025	124999750000	500000	125000250000
1000000	611377	499999500000	1000000	500000500000

Худший случай (2 алгоритм)				
n	Время, мс	Мп	Сп	Тп
100	0.0003	100	100	200
200	0.0004	200	200	400
500	0.0009	500	500	1000
1000	0.0018	1000	1000	2000
2000	0.0043	2000	2000	4000
5000	0.0086	5000	5000	10000
10000	0.0206	10000	10000	20000
100000	0.1795	100000	100000	200000
200000	0.3374	200000	200000	400000
500000	0.8265	500000	500000	1000000
1000000	1.7343	1000000	1000000	2000000

Случайное заполнение:

Средний случай (1 алгоритм)				
n	Время, мс	Мп	Сп	Тп
100	0.0032	2737	100	2837
200	0.0117	10105	200	10305
500	0.0929	59573	500	60073
1000	0.3446	256291	1000	257291
2000	1.2518	990935	2000	992935
5000	7.2221	6256506	5000	6261506
10000	29.6154	25280580	10000	25290580
100000	2987.61	2495461439	100000	2495561439
200000	11766	9980770488	200000	9980970488
500000	72992.9	62463298269	500000	62463798269
1000000	295419	249779924572	1000000	249780924572

Средний случай (2 алгоритм)				
n	Время, мс	Мп	Сп	Тп
100	0.0006	100	100	200
200	0.0014	200	200	400
500	0.0031	500	500	1000
1000	0.0065	1000	1000	2000
2000	0.0123	2000	2000	4000
5000	0.0307	5000	5000	10000
10000	0.0617	10000	10000	20000
100000	0.5596	100000	100000	200000
200000	1.0631	200000	200000	400000
500000	2.5149	500000	500000	1000000
1000000	5.4253	1000000	1000000	2000000

Лучший случай:

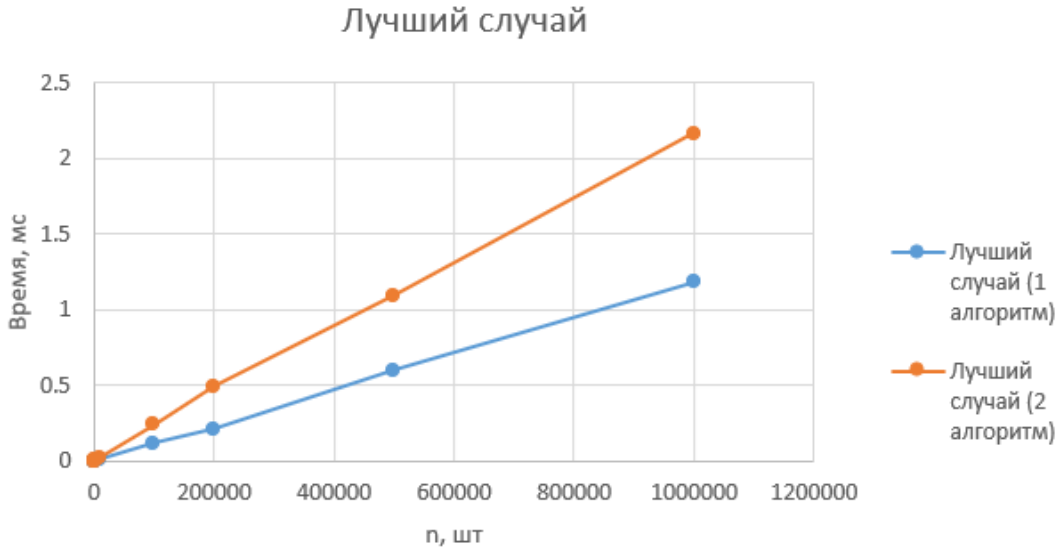
Лучший случай (1 алгоритм)				
n	Время, мс	Мп	Сп	Тп
100	0.0003	0	100	100
200	0.0003	0	200	200
500	0.0008	0	500	500
1000	0.0012	0	1000	1000
2000	0.0027	0	2000	2000
5000	0.0053	0	5000	5000
10000	0.0104	0	10000	10000
100000	0.1217	0	100000	100000
200000	0.2111	0	200000	200000
500000	0.5996	0	500000	500000
1000000	1.1819	0	1000000	1000000

Лучший случай (2 алгоритм)				
n	Время, мс	Мп	Сп	Тп
100	0.0003	100	100	200
200	0.0005	200	200	400
500	0.0011	500	500	1000
1000	0.0021	1000	1000	2000
2000	0.0049	2000	2000	4000
5000	0.0124	5000	5000	10000
10000	0.0254	10000	10000	20000
100000	0.2408	100000	100000	200000
200000	0.4947	200000	200000	400000
500000	1.0955	500000	500000	1000000
1000000	2.1666	1000000	1000000	2000000

1.7 Ёмкостная сложность алгоритмов

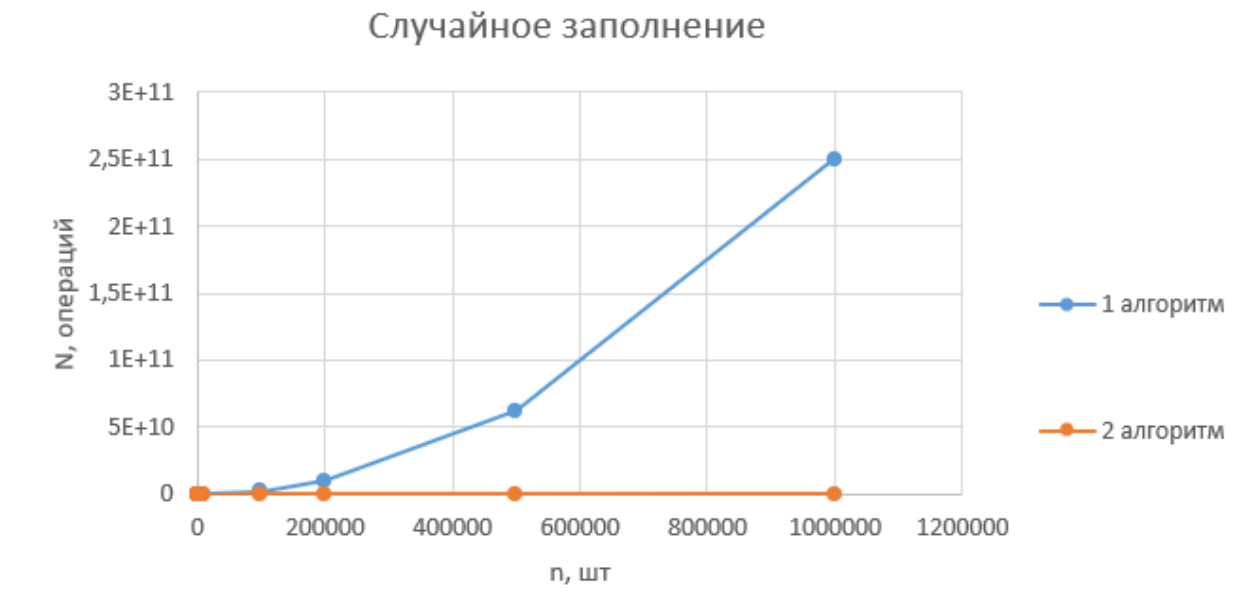
У обоих алгоритмов сложность по памяти O(1), т.к количество выделяемой доп. памяти не зависит от входных данных.

1.8 Графики зависимости по времени в трёх случаях

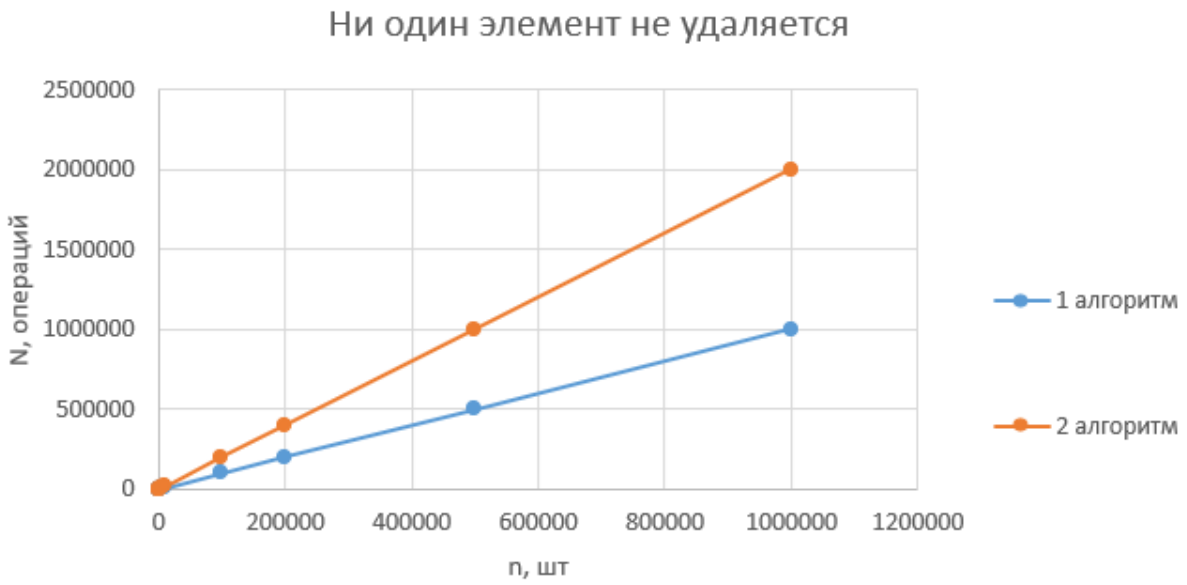


2-3 Зависимость по количеству операций в трёх случаях

Второй алгоритм совершает значительно меньше операций при одинаковых входных данных.



Второй алгоритм совершает значительно меньше операций при одинаковых входных данных.



В данном случае 1 алгоритм отрабатывает быстрее, но практической пользы это не имеет.

4 Предложение способа приведения длины массива к эффективной

Можно создать новый массив нужного размера, скопировать в него данные и удалить старый массив.

5 Вывод

Вывод: второй алгоритм гораздо эффективней при сложности O(n), в то время, как первый имеет сложность $O(n^2)$.

Задание 2

2.1 Код функции простой сортировки и проверка на массиве из 10 знач.

```
void ExchangeSort(int* arr, int size)
{
    for (int i = 0; i < size - 1; i++)
    {
        for (int j = 0; j < size - i - 1; j++)
        {
            if (arr[j] > arr[j + 1])
            {
                int temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }
}
```

```
Origin array: 10 3 8 4 6 5 1 9 2 7
Sorted array: 1 2 3 4 5 6 7 8 9 10
```

Полный код реализации алгоритмов для заданий 2-4

```cpp
#include <iostream>

#include <chrono>

int n[11] { 100, 200, 500, 1000, 2000, 5000, 10000, 100000, 200000, 500000, 1000000};

enum class Case {
    WORST, AVERAGE, BEST
};

void fillArray(int* arr, int size, Case _case)
{
    for (int i = 0; i < size; i++)
    {
        if (_case == Case::WORST)
        {
            arr[i] = size - i;
        }
        else if (_case == Case::AVERAGE)
        {
            arr[i] = rand() % 1000;
        }
        else if (_case == Case::BEST)
        {
            arr[i] = i;
        }
    }
}

void copyArray(int* originArray, int* copyToArray, int size)
{
    for (int i = 0; i < size; i++)
    {
        copyToArray[i] = originArray[i];
    }
}

void printArray(int* arr, int size)
{
    for (int i = 0; i < size; i++)
    {
        std::cout << arr[i] << " ";
    }
    std::cout << std::endl;
}
```

```cpp
void ExchangeSort(int* arr, int size)
{
    unsigned long long counterM = 0, counterC = 0;

    auto start = std::chrono::high_resolution_clock::now();

    for (int i = 0; i < size - 1; i++)
    {
        for (int j = 0; j < size - i - 1; j++)
        {
            if (arr[j] > arr[j + 1])
            {
                int temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;

                counterM += 3;
            }
            counterC++;
        }
    }

    auto end = std::chrono::high_resolution_clock::now();
    auto deltaTime = std::chrono::duration_cast<std::chrono::nanoseconds>(end - start);

    std::cout << "Sorted array: ";
    printArray(arr, 10);
    std::cout << std::endl;

    std::cout << "Time elapsed: " << deltaTime.count() / 1'000'000.0 << " ms" << std::endl;
    std::cout << "M: " << counterM << std::endl <<
        "C: " << counterC << std::endl <<
        "T: " << counterM + counterC << std::endl;
}

void InsertionSort(int* arr, int size)
{
    unsigned long long counterM = 0, counterC = 0;

    auto start = std::chrono::high_resolution_clock::now();

    for (int i = 0; i < size; i++)
    {
        int key = arr[i];
        int j = i - 1;
```

```
92          while (j >= 0 && arr[j] > key)
93          {
94              arr[j + 1] = arr[j];
95              j--;
96
97              counterC++;
98              counterM++;
99          }
100
101         if (j >= 0) counterC++;
102
103         arr[j + 1] = key;
104         counterM++;
105     }
106
107     auto end = std::chrono::high_resolution_clock::now();
108     auto deltaTime = std::chrono::duration_cast<std::chrono::nanoseconds>(end - start);
109
110     std::cout << "Sorted array: ";
111     printArray(arr, 10);
112     std::cout << std::endl;
113
114     std::cout << "Time elapsed: " << deltaTime.count() / 1'000'000.0 << " ms" << std::endl;
115     std::cout << "M: " << counterM << std::endl <<
116         "C: " << counterC << std::endl <<
117         "T: " << counterM + counterC << std::endl;
118 }
119
120 int main()
121 {
122     for (int size : n)
123     {
124         int* arr = new int[size];
125         int* arr2 = new int[size];
126
127         fillArray(arr, size, Case::WORST);
128         copyArray(arr, arr2, size);
129
130         std::cout << "--------------------" << std::endl;
131         std::cout << "n = " << size << std::endl;
132
133         std::cout << std::endl << "- ExchangeSort -" << std::endl;
134         std::cout << "Origin array: ";
135         printArray(arr, 10);
```

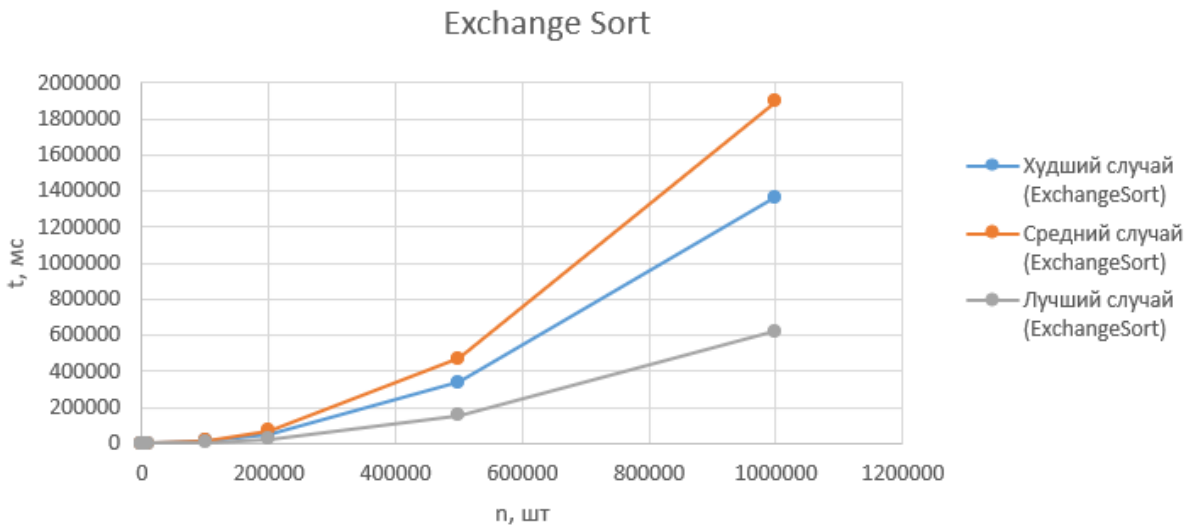
```
136
137             ExchangeSort(arr, size);
138             std::cout << std::endl;
139
140             std::cout << "- InsertionSort -" << std::endl;
141             std::cout << "Origin array: ";
142             printArray(arr2, 10);
143
144             InsertionSort(arr2, size);
145
146             delete[] arr;
147             delete[] arr2;
148         }
149
150         return 0;
151     }
```

2.2-2.3 Контрольные прогоны при n = 100, 200 …, 1000000 (сред. случай)

Средний случай (ExchangeSort)				
n	Время, мс	Мп	Сп	Тп
100	0.0198	7584	4950	12534
200	0.0531	30432	19900	50332
500	0.2828	176385	124750	301135
1000	1.127	756159	499500	1255659
2000	4.2116	2965992	1999000	4964992
5000	25.2344	18392379	12497500	30889879
10000	103.363	76072482	49995000	126067482
100000	17067.1	7508771682	4999950000	12508721682
200000	71820	29922293010	19999900000	49922193010
500000	465406	187120572372	124999750000	312120322372
1000000	1855530	748510547154	1248510047154	1997020594308

Вычислительная сложность алгоритма $O(n^2)$.

2.4 График зависимости по времени



2.5 Ёмкостная сложность

Емкостная сложность алгоритма - O(1), т.к алгоритм не задействует дополнительную память в ходе выполнения.

2.6 Вывод

Вывод: эмпирически доказано, что сложность и количество выполненных операций растут квадратично, что подтверждает предположенную сложность алгоритма $O(n^2)$.

Задание 3

3.1 Дополнительные прогоны (худший и лучший случаи)

а) В строго убывающем порядке:

Худший случай (ExchangeSort)				
n	Время, мс	Мп	Сп	Тп
100	0.0142	14850	4950	19800
200	0.0547	59700	19900	79600
500	0.3445	374250	124750	499000
1000	1.3518	1498500	499500	1998000
2000	5.3806	5997000	1999000	7996000
5000	33.8683	37492500	12497500	49990000
10000	134.726	149985000	49995000	199980000
100000	13565.4	14999850000	4999950000	19999800000
200000	54104.1	59999700000	19999900000	79999600000
500000	339591	374999250000	124999750000	499999000000
1000000	1356227	1499998500000	499999500000	1999998000000

б) В строго возрастающем порядке:

Лучший случай (ExchangeSort)				
n	Время, мс	Мп	Сп	Тп
100	0.007	0	4950	4950
200	0.0307	0	19900	19900
500	0.163	0	124750	124750
1000	0.7786	0	499500	499500
2000	2.4899	0	1999000	1999000
5000	15.7767	0	12497500	12497500
10000	62.3558	0	49995000	49995000
100000	6164.77	0	4999950000	4999950000
200000	24872	0	19999900000	19999900000
500000	154627	0	124999750000	124999750000
1000000	623110	0	499999500000	499999500000

3.2 Вывод о зависимости

Вывод: в наихудшем случае количество перемещений максимально, что приводит к большим затратам по времени выполнения. В наилучшем случае перемещения не выполняются, что сказывается на выполнении алгоритма в лучшую сторону.

Задание 4

4.1 Код реализации сортировки вставками

Реализация алгоритма (сортировка вставками):

```cpp
void InsertionSort(int* arr, int size)
{
    unsigned long long counterM = 0, counterC = 0;

    auto start = std::chrono::high_resolution_clock::now();

    for (int i = 0; i < size; i++)
    {
        int key = arr[i];
        int j = i - 1;

        while (j >= 0 && arr[j] > key)
        {
            arr[j + 1] = arr[j];
            j--;

            counterC++;
            counterM++;
        }

        if (j >= 0) counterC++;

        arr[j + 1] = key;
        counterM++;
    }

    auto end = std::chrono::high_resolution_clock::now();
    auto deltaTime = std::chrono::duration_cast<std::chrono::milliseconds>(end - start);

    std::cout << "Sorted array: ";
    printArray(arr, 10);
    std::cout << std::endl;

    std::cout << "Time elapsed: " << deltaTime.count() << std::endl;
    std::cout << "M: " << counterM << std::endl <<
        "C: " << counterC << std::endl <<
        "T: " << counterM + counterC << std::endl;
}
```

4.2 Таблицы с результатами эмпирического исследования

Худший случай (InsertionSort)				
n	Время, мс	Мп	Сп	Тп
100	0.0089	5050	4950	10000
200	0.0327	20100	19900	40000
500	0.1972	125250	124750	250000
1000	0.8854	500500	499500	1000000
2000	3.4337	2001000	1999000	4000000
5000	20.8165	12502500	12497500	25000000
10000	82.2531	50005000	49995000	100000000
100000	7915.26	5000050000	4999950000	10000000000
200000	31637.4	20000100000	19999900000	40000000000
500000	197838	125000250000	124999750000	250000000000
1000000	785223	500000500000	499999500000	1000000000000

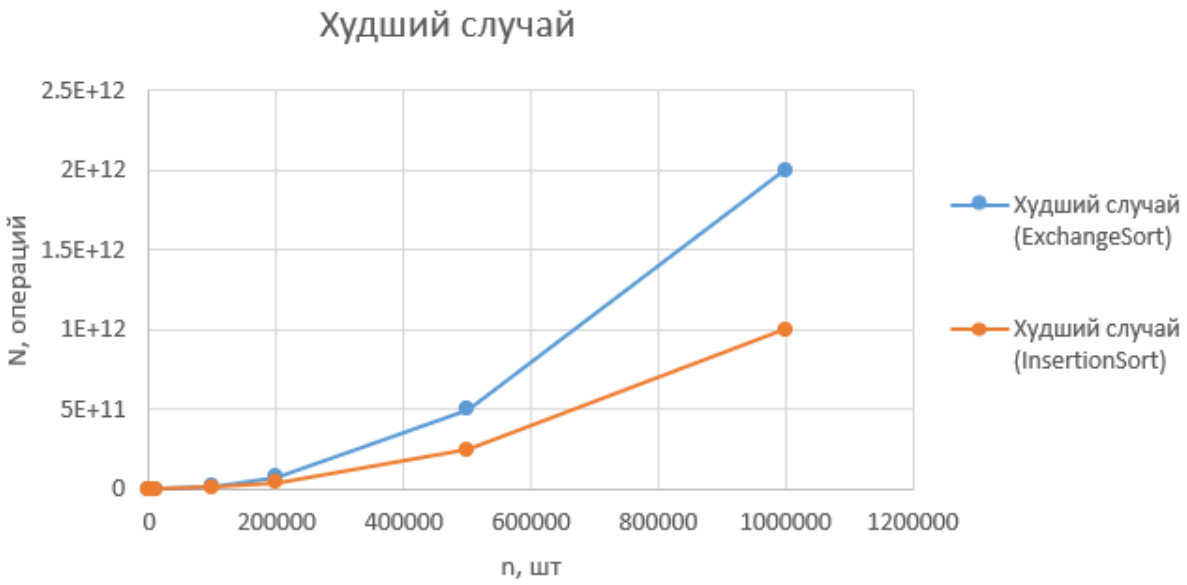
Средний случай (InsertionSort)				
n	Время, мс	Мп	Сп	Тп
100	0.0077	2628	2625	5253
200	0.0179	10344	10338	20682
500	0.1118	59295	59285	118580
1000	0.5079	253053	253048	506101
2000	1.5969	990664	990655	1981319
5000	9.6829	6135793	6135787	12271580
10000	40.1842	25367494	25367483	50734977
100000	3947.85	2503023894	2503023887	5006047781
200000	15808.6	9974297670	9974297659	19948595329
500000	99515.6	62374024124	62374024114	124748048238
1000000	393466	249504515718	249504515713	499009031431

Лучший случай (InsertionSort)				
n	Время, мс	Мп	Сп	Тп
100	0.0004	100	99	199
200	0.0007	200	199	399
500	0.0017	500	499	999
1000	0.003	1000	999	1999
2000	0.0059	2000	1999	3999
5000	0.0116	5000	4999	9999
10000	0.0287	10000	9999	19999
100000	0.2413	100000	99999	199999
200000	0.5391	200000	199999	399999
500000	1.3111	500000	499999	999999
1000000	2.7124	1000000	999999	1999999

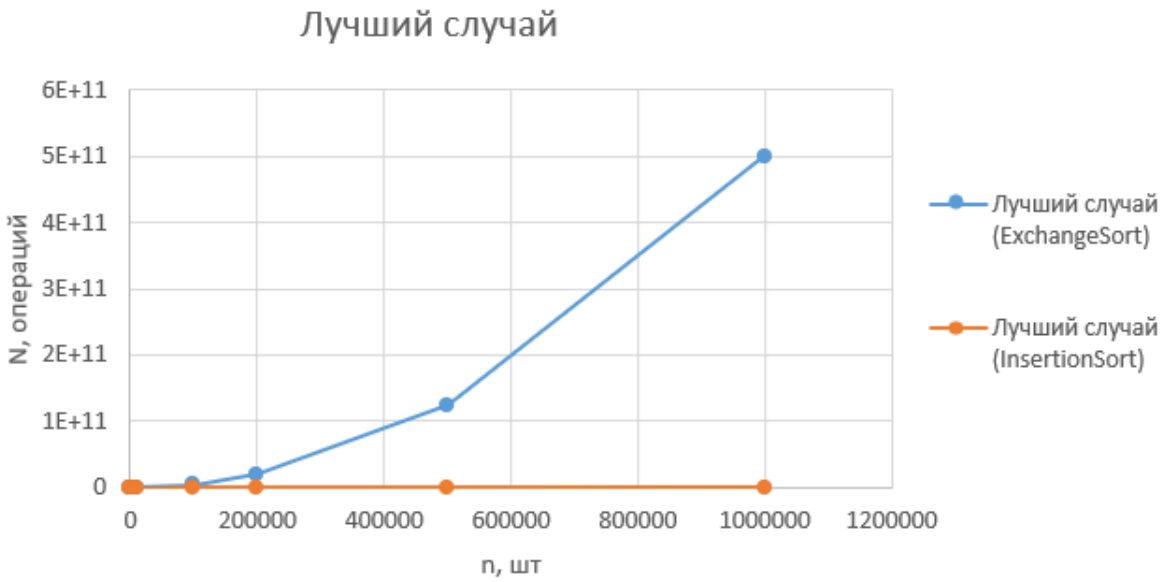
4.3 Ёмкостная сложность

В ходе выполнения алгоритма дополнительная память не выделяется, так что ёмкостная сложность алгоритма O(1).

4.4 График зависимости по количеству операций (худший случай)



4.5 График зависимости по количеству операций (лучший случай)



4.6 Вывод

Вывод: в худшем случае оба алгоритма имеют сложность $O(n^2)$. В лучшем случае сортировка вставками значительно эффективнее $O(n)$, против $O(n^2)$ у пузырьковой. В целом, сортировка вставками, показала себя более эффективной, чем сортировка пузырьком.

ЛИТЕРАТУРА:

1. Бхаргава А. Грокаем алгоритмы, 2-е изд. – СПб: Питер, 2024. – 352 с.

2. Вирт Н. Алгоритмы + структуры данных = программы. – М.: Мир, 1985. – 406 с. 3. Кнут Д.Э. Искусство программирования, том

3. Сортировка и поиск, 2-е изд. – М.: ООО «И.Д. Вильямс», 2018. – 832 с.

4. Седжвик Р. Фундаментальные алгоритмы на C++. Анализ/Структуры данных/Сортировка/Поиск. – К.: Издательство «Диасофт», 2001. – 688 с.

5. Алгоритмы – всё об алгоритмах / Хабр [Электронный ресурс]. URL: https://habr.com/ru/hub/algorithms/ (дата обращения 05.02.2025).