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

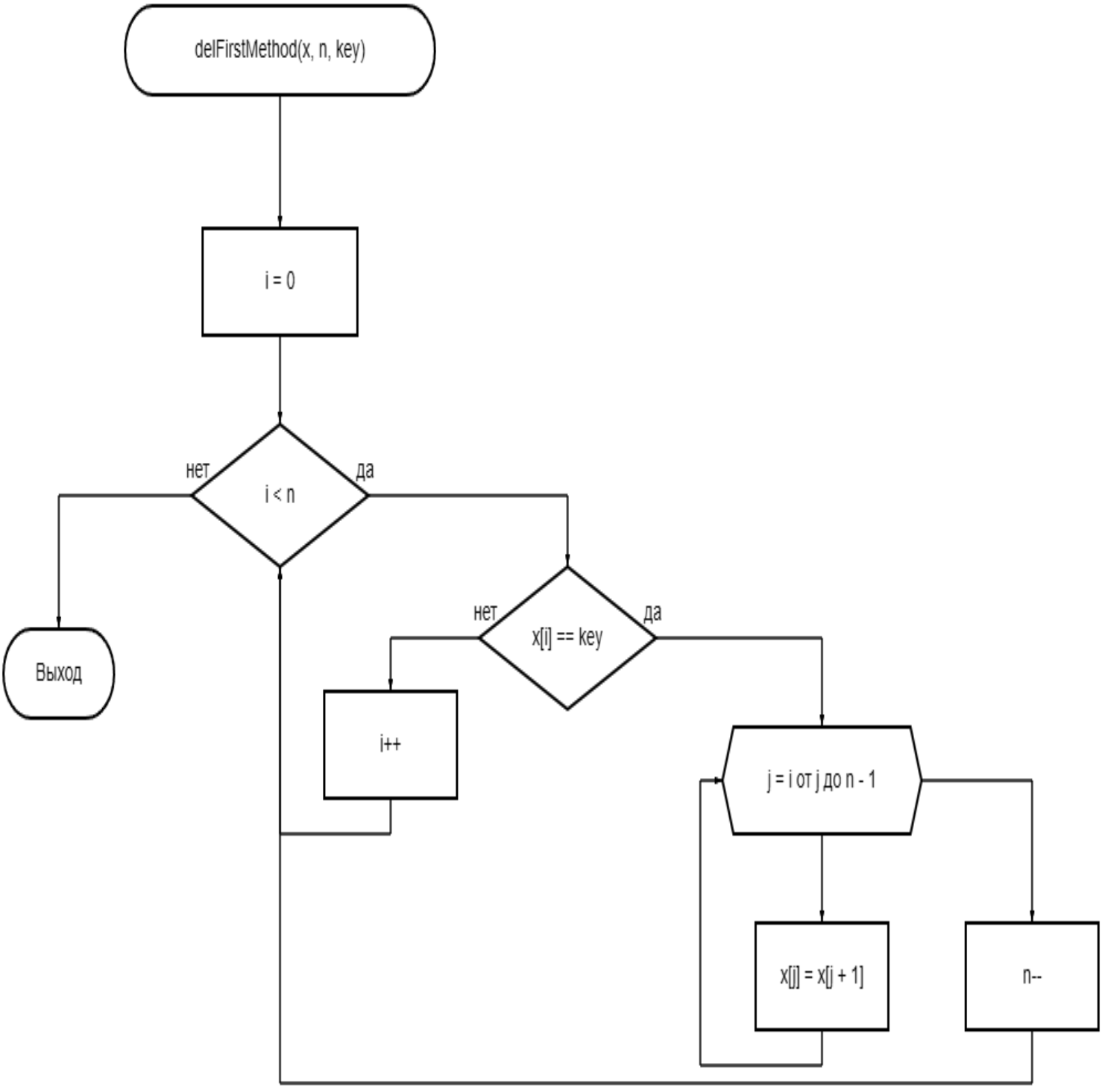
Выполнил студент student
группа group

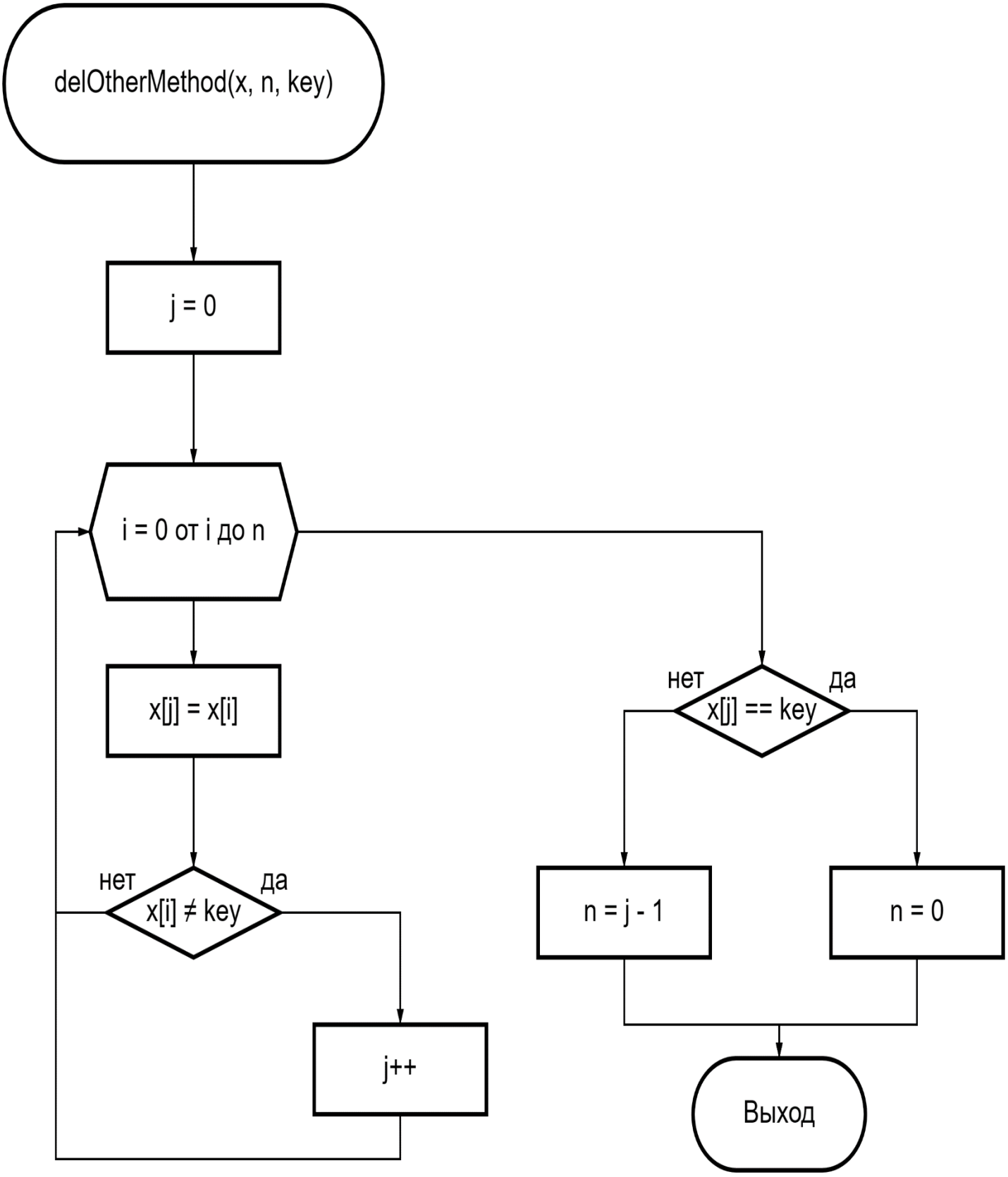
Москва 2025

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

Задание 1

1.1




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

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

```cpp
#include <iostream>
Ссылок: 0
#include <vector>
Ссылок: 0
#include <random>
Ссылок: 27
#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::milliseconds>(end - start);

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

```
90      std::cout << std::endl;
91
92      std::cout << "Time elapsed: " << deltaTime.count() << std::endl;
93      std::cout << "M: " << counterM << std::endl
94          << "C: " << counterC << std::endl
95          << "T: " << counterM + counterC << std::endl;
96  }
97
98  void delOtherMethod(char x[], int n, char key)
99  {
100     unsigned long long counterC = 0;
101     unsigned long long counterM = 0;
102
103
104     auto start = std::chrono::high_resolution_clock::now();
105
106     int j = 0;
107     for (int i = 0; i < n; i++)
108     {
109         x[j] = x[i];
110         counterM++;
111         if (x[i] != key)
112         {
113             j++;
114         }
115         counterC++;
116     }
117
118     if (x[1] == key) n = 0;
119     else n = j - 1;
120     counterC++;
121
122     auto end = std::chrono::high_resolution_clock::now();
123     auto deltaTime = std::chrono::duration_cast<std::chrono::milliseconds>(end - start);
124
125     std::cout << "Sorted array: ";
126     printArray(x, 10);
127     std::cout << std::endl;
128
129     std::cout << std::endl << "Time elapsed: " << deltaTime.count() << std::endl;
130     std::cout << "M: " << counterM << std::endl <<
131         "C: " << counterC << std::endl <<
132         "T: " << counterM + counterC << std::endl;
133 }
```

```
135 int main()
136 {
137     srand(time(NULL));
138
139     char key;
140     std::cin >> key;
141
142     for (int size : n)
143     {
144         char* arr = new char[size];
145         char* arr2 = new char[size];
146
147         fillArray(size, Case::BEST, arr, key);
148         copyArray(arr, arr2, size);
149
150         std::cout << "--------------------" << std::endl;
151         std::cout << "n = " << size << std::endl;
152
153         std::cout << "First algorithm" << std::endl;
154         std::cout << "Origin array: ";
155         printArray(arr, 10);
156         delFirstMethod(arr, size, key);
157
158         std::cout << std::endl;
159
160         std::cout << "Second algorithm" << std::endl;
161         std::cout << "Origin array: ";
162         printArray(arr2, 10);
163         delOtherMethod(arr2, size, key);
164
165         delete[] arr;
166         delete[] arr2;
167     }
168
169     return 0;
170 }
```

1.5

а) все элементы должны быть удалены

Худший случай (1 алгоритм)				
n	Время, мс	Мп	Сп	Тп
100	0	4950	100	5050
200	0	19900	200	20100
500	0	124750	500	125250
1000	0	499500	1000	500500
2000	2	1999000	2000	2001000
5000	14	12497500	5000	12502500
10000	58	49995000	10000	50005000

Худший случай (2 алгоритм)				
n	Время, мс	Мп	Сп	Тп
100	0	100	101	201
200	0	200	201	401
500	0	500	501	1001
1000	0	1000	1001	2001
2000	0	2000	2001	4001
5000	0	5000	5001	10001
10000	0	10000	10001	20001

б) случайное заполнение

Средний случай (1 алгоритм)				
n	Время, мс	Мп	Сп	Тп
100	0	2737	100	2837
200	0	10105	200	10305
500	0	59573	500	60073
1000	0	256291	1000	257291
2000	1	990935	2000	992935
5000	7	6256506	5000	6261506
10000	29	25280580	10000	25290580

Средний случай (2 алгоритм)				
n	Время, мс	Мп	Сп	Тп
100	0	100	101	201
200	0	200	201	401
500	0	500	501	1001
1000	0	1000	1001	2001
2000	0	2000	2001	4001
5000	0	5000	5001	10001
10000	0	10000	10001	20001

в) ни один элемент не удаляется

Лучший случай (1 алгоритм)				
n	Время, мс	Мп	Сп	Тп
100	0	0	100	100
200	0	0	200	200
500	0	0	500	500
1000	0	0	1000	1000
2000	0	0	2000	2000
5000	0	0	5000	5000
10000	0	0	10000	10000

Лучший случай (2 алгоритм)				
n	Время, мс	Мп	Сп	Тп
100	0	100	101	201
200	0	200	201	401
500	0	500	501	1001
1000	0	1000	1001	2001
2000	0	2000	2001	4001
5000	0	5000	5001	10001
10000	0	10000	10001	20001

1.6

Худший случай:

Худший случай (1 алгоритм)				
n	Время, мс	Мп	Сп	Тп
100	0	4950	100	5050
200	0	19900	200	20100
500	0	124750	500	125250
1000	0	499500	1000	500500
2000	2	1999000	2000	2001000
5000	14	12497500	5000	12502500
10000	58	49995000	10000	50005000
100000	5820	4999950000	100000	5000050000
200000	23379	1999990000	200000	2000190000
500000	146448	124999750000	500000	125000250000
1000000	587099	499999500000	1000000	500000500000

Худший случай (2 алгоритм)				
n	Время, мс	Мп	Сп	Тп
100	0	100	101	201
200	0	200	201	401
500	0	500	501	1001
1000	0	1000	1001	2001
2000	0	2000	2001	4001
5000	0	5000	5001	10001
10000	0	10000	10001	20001
100000	0	100000	100001	200001
200000	0	200000	200001	400001
500000	0	500000	500001	1000001
1000000	1	1000000	1000001	2000001

Случайное заполнение:

Средний случай (1 алгоритм)				
n	Время, мс	Мп	Сп	Тп
100	0	2737	100	2837
200	0	10105	200	10305
500	0	59573	500	60073
1000	0	256291	1000	257291
2000	1	990935	2000	992935
5000	7	6256506	5000	6261506
10000	29	25280580	10000	25290580
100000	2948	2495461439	100000	2495561439
200000	11679	9980770488	200000	9980970488
500000	76197	62463298269	500000	62463798269
1000000	304685	249779924572	1000000	249780924572

Средний случай (2 алгоритм)				
n	Время, мс	Мп	Сп	Тп
100	0	100	101	201
200	0	200	201	401
500	0	500	501	1001
1000	0	1000	1001	2001
2000	0	2000	2001	4001
5000	0	5000	5001	10001
10000	0	10000	10001	20001
100000	0	100000	100001	200001
200000	0	200000	200001	400001
500000	2	500000	500001	1000001
1000000	5	1000000	1000001	2000001

Лучший случай:

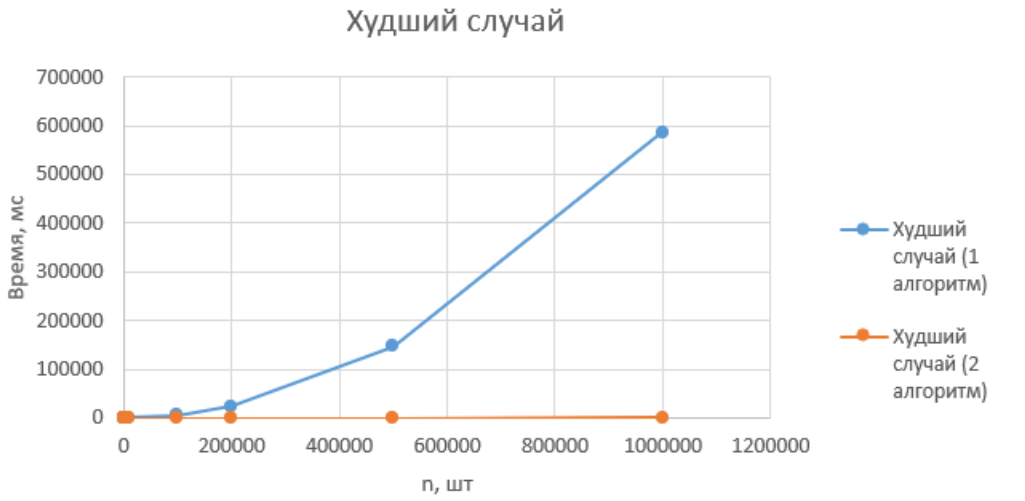
Лучший случай (1 алгоритм)				
n	Время, мс	Мп	Сп	Тп
100	0	0	100	100
200	0	0	200	200
500	0	0	500	500
1000	0	0	1000	1000
2000	0	0	2000	2000
5000	0	0	5000	5000
10000	0	0	10000	10000
100000	0	0	100000	100000
200000	0	0	200000	200000
500000	0	0	500000	500000
1000000	0	0	1000000	1000000

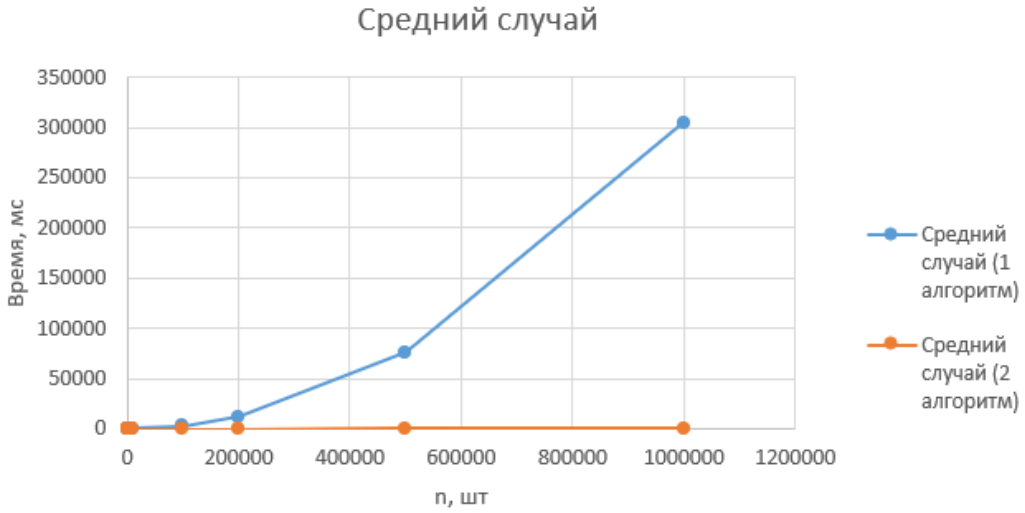
Лучший случай (2 алгоритм)				
n	Время, мс	Мп	Сп	Тп
100	0	100	101	201
200	0	200	201	401
500	0	500	501	1001
1000	0	1000	1001	2001
2000	0	2000	2001	4001
5000	0	5000	5001	10001
10000	0	10000	10001	20001
100000	0	100000	100001	200001
200000	0	200000	200001	400001
500000	1	500000	500001	1000001
1000000	2	1000000	1000001	2000001

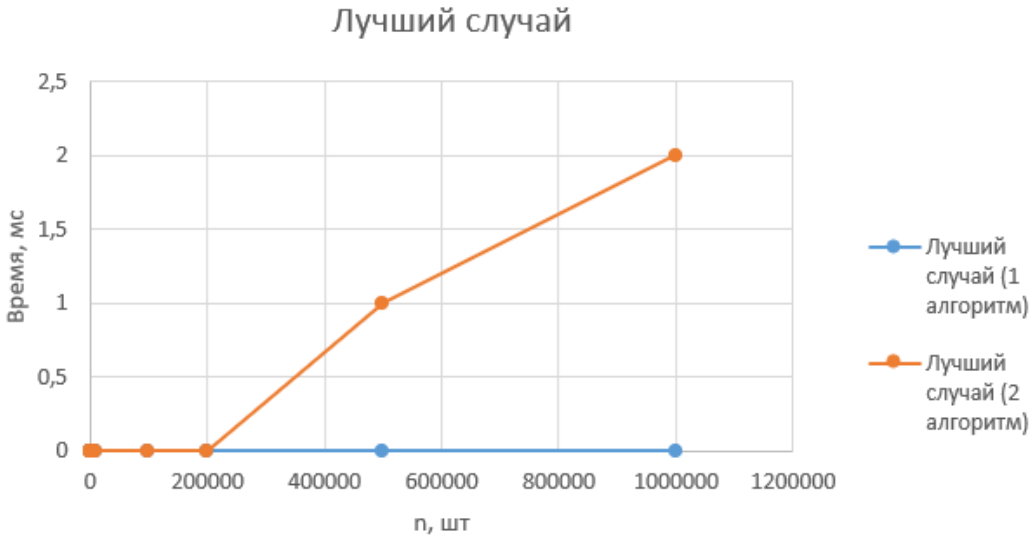
1.7

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

1.8

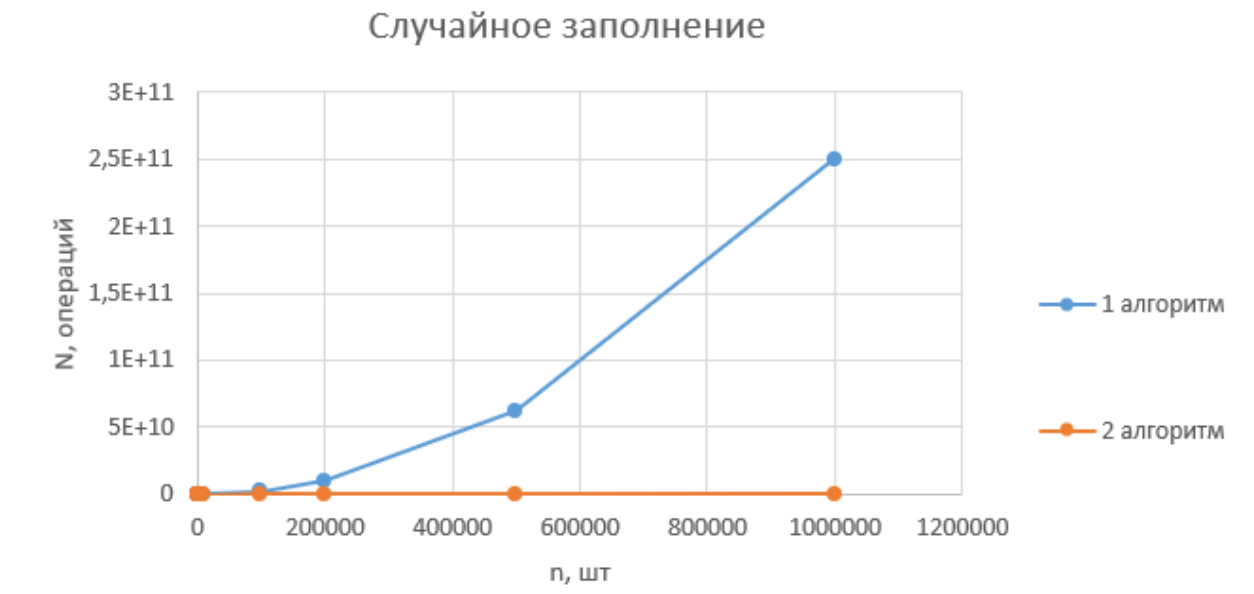




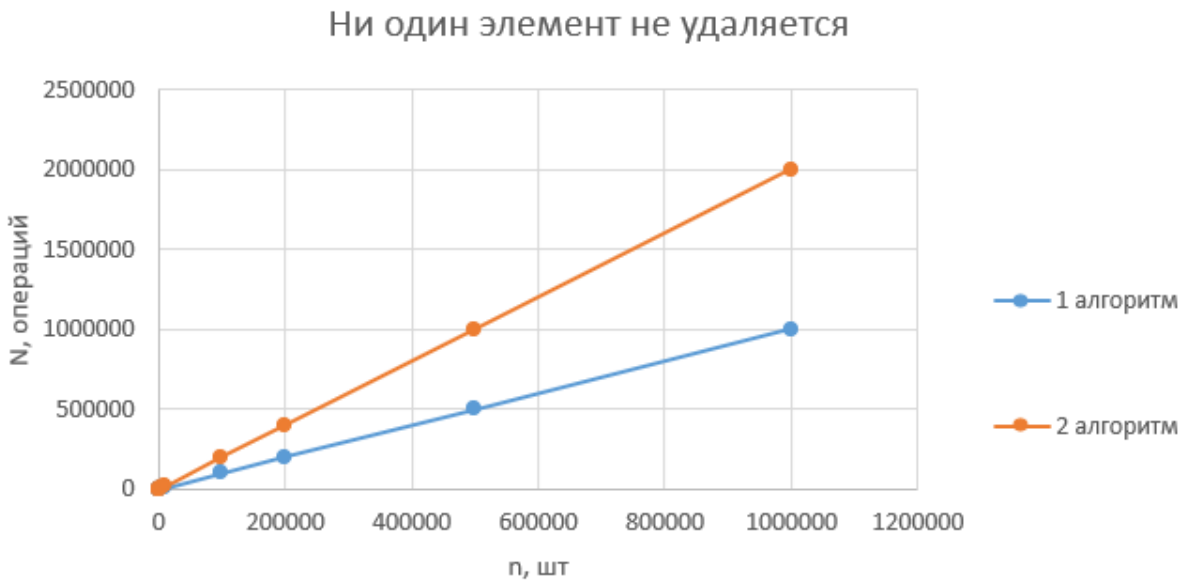


2-3

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.

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

Задание 2

2.1

```
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::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;
}
```

```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;
}
```

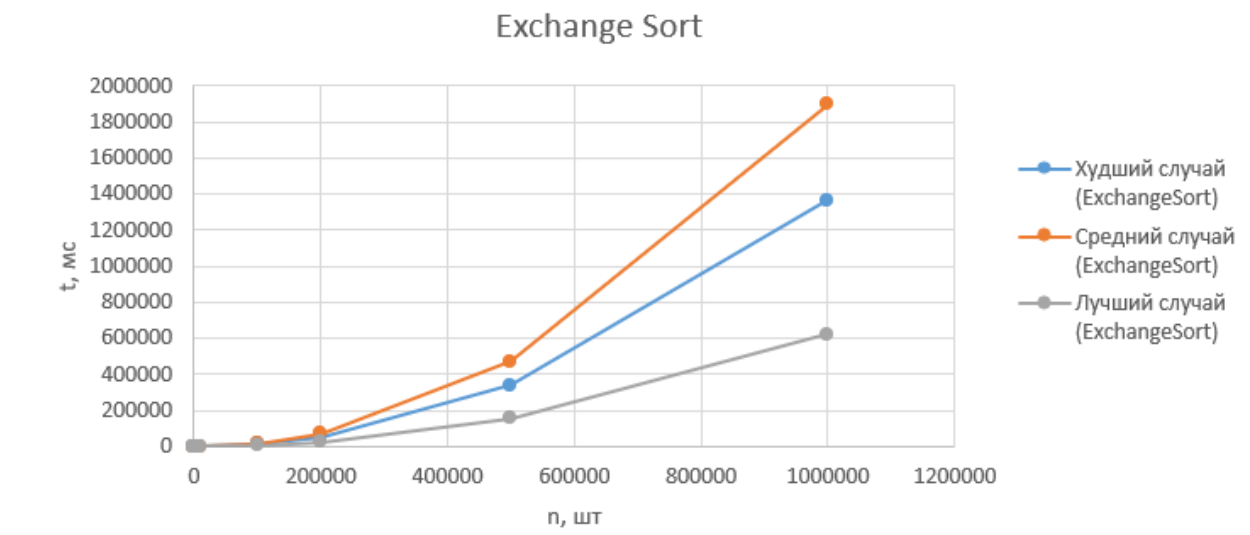
```cpp
int main()
{
    for (int size : n)
    {
        int* arr = new int[size];
        int* arr2 = new int[size];

        fillArray(arr, size, Case::AVERAGE);
        copyArray(arr, arr2, size);

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

        std::cout << std::endl << "- ExchangeSort -" << std::endl;
        std::cout << "Origin array: ";
        printArray(arr, 10);

        ExchangeSort(arr, size);
        std::cout << std::endl;

        std::cout << "- InsertionSort -" << std::endl;
        std::cout << "Origin array: ";
        printArray(arr2, 10);

        InsertionSort(arr2, size);

        delete[] arr;
        delete[] arr2;
    }

    return 0;
}
```

2.2-2.3

n	Время, мс	Мп	Сп	Тп
100	0	5056	4950	10006
200	0	20288	19900	40188
500	0	117590	124750	242340
1000	1	504106	499500	1003606
2000	4	1977328	1999000	3976328
5000	25	12261586	12497500	24759086
10000	111	50714988	49995000	100709988
100000	17052	5005847788	4999950000	10005797788
200000	72374	19948195340	19999900000	39948095340
500000	469204	124747048248	124999750000	249746798248
1000000	1896364	499007031436	499999500000	999006531436

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

2.4



2.5

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

2.6

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

Задание 3
3.1

а) В строго убывающем порядке:

Худший случай (ExchangeSort)				
n	Время, мс	Мп	Сп	Тп
100	0	9900	4950	14850
200	0	39800	19900	59700
500	0	249500	124750	374250
1000	1	999000	499500	1498500
2000	5	3998000	1999000	5997000
5000	35	24995000	12497500	37492500
10000	136	99990000	49995000	149985000
100000	13495	9999900000	4999950000	14999850000
200000	54704	39999800000	19999900000	59999700000
500000	341174	249999500000	124999750000	374999250000
1000000	1367758	999999000000	499999500000	1499998500000

б) В строго возрастающем порядке:

Лучший случай (ExchangeSort)				
n	Время, мс	Мп	Сп	Тп
100	0	0	4950	4950
200	0	0	19900	19900
500	0	0	124750	124750
1000	0	0	499500	499500
2000	2	0	1999000	1999000
5000	15	0	12497500	12497500
10000	62	0	49995000	49995000
100000	6203	0	4999950000	4999950000
200000	24666	0	19999900000	19999900000
500000	154428	0	124999750000	124999750000
1000000	621099	0	499999500000	499999500000

3.2

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

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

Худший случай (InsertionSort)				
n	Время, мс	Мп	Сп	Тп
100	0	5050	4950	10000
200	0	20100	19900	40000
500	0	125250	124750	250000
1000	1	500500	499500	1000000
2000	3	2001000	1999000	4000000
5000	20	12502500	12497500	25000000
10000	78	50005000	49995000	100000000
100000	7884	5000050000	4999950000	10000000000
200000	31321	20000100000	19999900000	40000000000
500000	200909	125000250000	124999750000	250000000000
1000000	785226	500000500000	499999500000	1000000000000

Средний случай (InsertionSort)				
n	Время, мс	Мп	Сп	Тп
100	0	2628	2625	5253
200	0	10344	10338	20682
500	0	59295	59285	118580
1000	0	253053	253048	506101
2000	1	990664	990655	1981319
5000	9	6135793	6135787	12271580
10000	40	25367494	25367483	50734977
100000	3969	2503023894	2503023887	5006047781
200000	16152	9974297670	9974297659	19948595329
500000	98405	62374024124	62374024114	124748048238
1000000	395420	249504515718	249504515713	499009031431

Лучший случай (InsertionSort)				
n	Время, мс	Мп	Сп	Тп
100	0	100	99	199
200	0	200	199	399
500	0	500	499	999
1000	0	1000	999	1999
2000	0	2000	1999	3999
5000	0	5000	4999	9999
10000	0	10000	9999	19999
100000	0	100000	99999	199999
200000	0	200000	199999	399999
500000	1	500000	499999	999999
1000000	2	1000000	999999	1999999

4.3

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

4.4

4.5

4.6

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).