МИНОБРНАУКИ РОССИИ

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

РТУ МИРЭА

Отчет по выполнению практического задания № 2

Тема:

«ЭМПИРИЧЕСКИЙ АНАЛИЗ СЛОЖНОСТИ ПРОСТЫХ АЛГОРИТМОВ СОРТИРОВКИ»

Дисциплина: «Структуры и алгоритмы обработки данных»

Выполнил студент: Боргачев Т.М.

Группа: ИНБО-10-23

Москва – 2024

СОДЕРЖАНИЕ

1 ФОРМУЛИРОВКА ЗАДАЧИ

1.1 Цель

Актуализация знаний и приобретение практических умений по эмпирическому определению вычислительной сложности алгоритмов.

1.2 Задание 1

Оценить эмпирически вычислительную сложность алгоритма простой сортировки на массиве, заполненном случайными числами (средний случай).

1. Составить функцию простой сортировки одномерного целочисленного массива A[n], используя алгоритм согласно варианту индивидуального задания (2): 1, 2 задание – алгоритм простого обмена («пузырек», Exchange sort); 3 задание – алгоритм простой вставки (Insertion Sort). Провести тестирование программы на исходном массиве n=10.

2. Используя теоретический подход, определить для алгоритма:

   a. Что будет ситуациями лучшего, среднего и худшего случаев.

   b. Функции роста времени работы алгоритма от объёма входа для лучшего и худшего случаев.

3. Провести контрольные прогоны программы массивов случайных чисел при n = 100, 1000, 10000, 100000 и 1000000 элементов с вычислением времени выполнения T(n) – (в миллисекундах/секундах).

4. Провести эмпирическую оценку вычислительной сложности алгоритма, для чего предусмотреть в программе подсчет фактического количества критических операций $T_{\text{п}}$ как сумму сравнений $\text{C}_{\text{п}}$ и перемещений $\text{M}_{\text{п}}$.

5. Построить график функции роста $\text{T}_{\text{п}}$ этого алгоритма от размера массива n.

6. Определить ёмкостную сложность алгоритма.

7. Сделать вывод об эмпирической вычислительной сложности алгоритма на основе скорости роста функции роста.

1.3 Задание 2

Оценить вычислительную сложность алгоритма простой сортировки в

наихудшем и наилучшем случаях.

1. Провести дополнительные прогоны программы на массивах при n = 100, 1000, 10000, 100000 и 1000000 элементов, отсортированных:

a. строго в убывающем порядке значений.

b. строго в возрастающем порядке значений.

2. Сделать вывод о зависимости (или независимости) алгоритма сортировки от исходной упорядоченности массива.

1.4 Задание 3

Сравнить эффективность алгоритмов простых сортировок

1. Выполнить разработку и программную реализацию второго алгоритма согласно индивидуальному варианту (2).

2. Аналогично заданиям 1 и 2 сформировать таблицы с результатами эмпирического исследования второго алгоритма в среднем, лучшем и худшем случаях (на тех же массивах, что и в заданиях 1 и 2).

3. Определить ёмкостную сложность алгоритма от n.

4. На одном сравнительном графике отобразить функции $T_п(n)$ двух алгоритмов сортировки в худшем случае.

5. Аналогично на другом общем графике отобразить функции $T_п(n)$ двух алгоритмов сортировки для лучшего случая.

6. Выполнить сравнительный анализ полученных результатов для двух алгоритмов.

2 ПРОЕКТИРОВАНИЕ И РЕАЛИЗАЦИЯ

2.1 Задание 1

2.1.1 Реализация алгоритма в виде функции

Напишем отдельно функции для заполнения массива случайными значениями рис. 1 и вывода массива на экран рис. 2.

```
int* RandChisla(int n) {
    srand(time(NULL));
    int* x = new int[n];
    for (int i = 0; i < n; i++) {
        x[i] = rand() % 100;
    }
    return(x);
}
```

Рисунок 1 – Функция заполнения массива случайными значениями

```
void Output(int* y, int n) {
    for (int i = 0; i < n; i++) {
        cout << y[i] << " ";
    }
    cout << endl;
}
```

Рисунок 2 – Функция вывода массива на экран

Функция, реализующая сортировку пузырьком представлена на рис. 3.

```
void bubble(int* x, int n)
{
    for (int i = 0; i < n - 1; i++)
        for (int j = 0; j < n - i - 1; j++)
            if (x[j] > x[j + 1])
                swap(x[j], x[j + 1]);
}
```

Рисунок 3 – Реализация алгоритма в виде функции

Результаты тестирования алгоритма на массиве со случайными элементами размера 10 представлены на рис. 4.

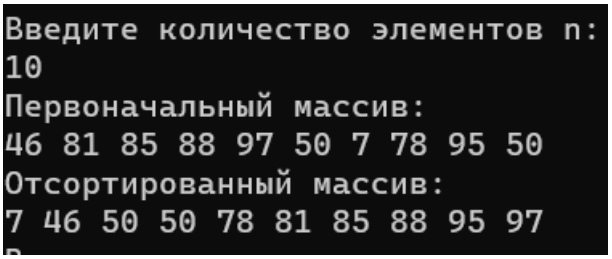

Рисунок 4 – Результаты тестирования алгоритма

Алгоритм работает корректно.

2.1.2 Определение функции роста алгоритма

Лучший случай для алгоритма: в массиве только один элемент. Средний случай для алгоритма: массив заполнен случайно. Худший случай: элементы массива расположены по убыванию. Посчитаем количество операций для лучшего и худшего случаев с помощью табл. 1.

Таблица 1 – Подсчет количества операций

Оператор	Кол-во выполнений оператора в строке	
	в лучшем случае	в худшем случае
for (int i=0; i<n-1; i++) {	1	n
for (int j=0; j<n-i-1; j++) {	0	$(\sum_{j=0}^{n-1} t_j) + 1$
if (x[j]>x[j+1]) {	0	$(\sum_{j=0}^{n-1} t_j)$
swap(x[j], x[j+1]) }	0	$(\sum_{j=0}^{n-1} t_j)$
}		
}		

Таким образом, функция роста алгоритма T(n) в лучшем случае будет равна 1, а в худшем случае $T(n) = n + 1 + (3\sum_{j=0}^{n-1} t_j) = n + 1 + (3*n*(n-1)/2) = n + 1 + 1.5n^2 – 1.5n = 1.5n^2 - 0.5n + 1$.

2.1.3 Вычислительная сложность алгоритма

Для проведения эмпирической оценкой вычислительной сложности алгоритма заполним и будем использовать табл. 2.

Таблица 2 – Время выполнения алгоритма и количество операций

n	**T(n), мс**	$T_п=C_п+M_п$
100	1	9373
1000	2	1012653
10000	216	99673661
100000	28571	1380270839
1000000	2984380	15302708390

График зависимости функции роста T(n) от значений n представлен на рис. 5.

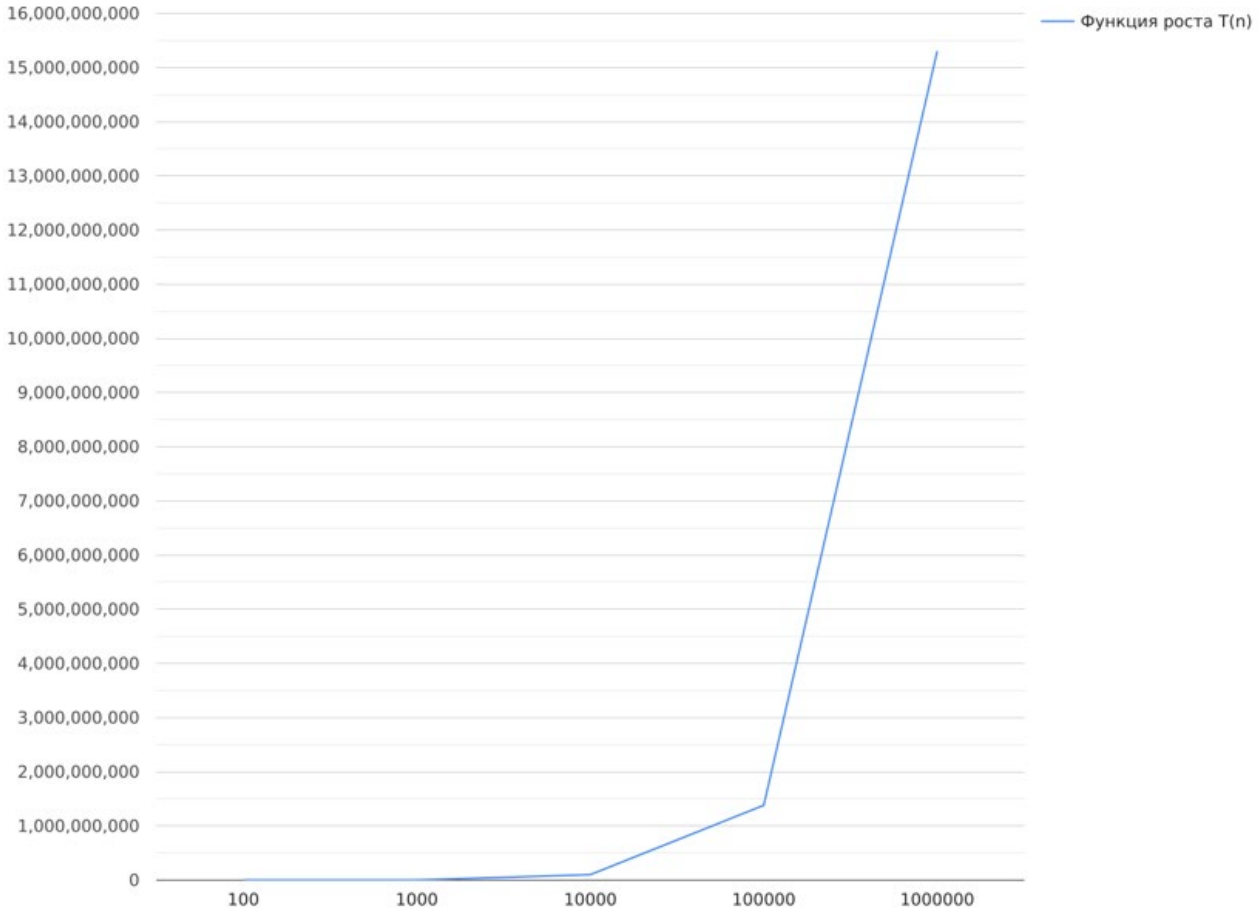

Рисунок 5 – График зависимости функции роста от n

2.1.4 Ёмкостная сложность алгоритма

Так как не требуется дополнительного массива для решения задачи, то требуется только один массив длиной n (ёмкостная сложность n).

2.2 Задание 2

2.2.1 Оценка вычислительной сложности алгоритма простой сортировки в наихудшем и наилучшем случаях

Результаты тестирования алгоритма в наихудшем и наилучшем случаях представлены соответственно в табл. 3 и 4.

Таблица 3 – Тестирование алгоритма в наихудшем случае

n	**T(n), мс**	$T_п = C_п + M_п$
100	1	14861
1000	3	1489457
10000	300	148997699
100000	34609	2015051561
1000000	3434910	2999992000005

Таблица 4 – Тестирование алгоритма в наилучшем случае

n	**T(n), мс**	$T_п = C_п + M_п$
100	1	5049
1000	2	500499
10000	72	50004999
100000	7201	705082703
1000000	766918	1784293663

Код, реализующий сортировку отсортированных массивов представлен на рис. 6.

```
int n = 10;
cout << "n = " << n << endl;
int* x = RandChisla(n);
sort(x, x + n);
cout << "Отсортированный изначальный массив: \n";
Output(x, n);
time_t begin = clock();
bubble1(x, n);
time_t end = clock();
double TIME = (double)(end - begin) / CLOCKS_PER_SEC;
cout << "Затраченное время: " << TIME * 1000 << endl;
cout << "n = " << n << endl;
int* y = RandChisla(n);
cout << "Отсортированный конечный массив: \n";
Output(y, n);
sort(y, y + n, comp());
time_t begin1 = clock();
bubble1(y, n);
time_t end1 = clock();
cout << "Отсортированный конечный массив: \n";
Output(y, n);
double TIME1 = (double)(end1 - begin1) / CLOCKS_PER_SEC;
cout << "Затраченное время: " << TIME1 * 1000 << endl;
```

Рисунок 6 – Код сортировки лучшего и худшего случаев

Результат работы кода представлен на рис. 7.

```
n = 10
Отсортированный изначальный массив:
16 17 28 33 33 63 66 82 93 95
Количество сравнений и перестановок: 54
Отсортированный конечный массив:
16 17 28 33 33 63 66 82 93 95
Затраченное время: 1
n = 10
Отсортированный изначальный массив:
95 93 82 66 63 33 33 28 17 16
Количество сравнений и перестановок: 142
Отсортированный конечный массив:
16 17 28 33 33 63 66 82 93 95
Затраченное время: 1
```

Рисунок 7 – Результат работы алгоритма в лучшем и худшем случаях

Таким образом, можно утверждать, что время работы алгоритма линейно зависит от исходной упорядоченности массива.

2.3 Задание 3

2.3.1 Разработка алгоритма

Код на языке программирования C++, реализующей алгоритм сортировки простой вставки представлен на рис. 8.

```
void insertion(int* x, int n)
{
    int key,j;
    for (int i = 1; i < n; i++) {
        key = x[i];
        j = i - 1;
        while (j >= 0 && x[j] > key) {
            x[j + 1] = x[j];
            j--;
        }
        x[j + 1] = key;
    }
}
```

Рисунок 8 – Алгоритм простой вставки

Результат тестирования алгоритма при n=10 представлены на рис.9.

```
n=10
Первоначальный массив:
2354 432 3241 532 1235 674 253 324 134 7543
Отсортированный массив:
134 253 324 432 532 674 1235 2354 3241 7543
```

Рисунок 9 – Тест алгоритма

Можно сделать вывод о корректной работе алгоритма.

2.3.2 Определение функции роста алгоритма

Рассчитаем функцию роста алгоритма, определив количество выполняемых операций с помощью табл. 5.

Таблица 5 - Количество операций алгоритма

Оператор	Кол-во выполнений оператора в строке	
	в лучшем случае	в худшем случае
int key, j;	2	2
for (int i = 1; i < n; i++) {	1	n
key = x[i];	0	n - 1

Оператор	Кол-во выполнений оператора в строке	
	в лучшем случае	в худшем случае
j = i - 1;	0	n - 1
while (j >= 0 && x[j] > key) {	0	$(\sum_0^{n-2} t_j)$
x[j + 1] = x[j];	0	$(\sum_0^{n-2} t_j)$
j--}	0	$(\sum_0^{n-2} t_j)$
x[j + 1] = key;}	0	n-1

Таким образом, функция роста в худшем случае T(n) = 4n-1 + (n-2)*(n-1)/2= 4n – 1 + 0.5n^2 -1.5n + 1 = 0.5n^2 +2.5n, а в лучшем случае T(n) = 3.

2.3.3 Ёмкостная сложность алгоритма

Так как не требуется дополнительного массива для решения задачи, то требуется только один массив длиной n (ёмкостная сложность n).

2.3.4 Эмпирическое исследование второго алгоритма

Результаты тестирования алгоритма в среднем, наилучшем и наихудшем случаях представлены соответственно в табл. 5, 6 и 7.

Таблица 5 – Тестирование алгоритма в среднем случае

n	**T(n), мс**	$T_п=C_п+M_п$
100	1	5366
1000	1	499076
10000	43	49605128
100000	4198	648864620
1000000	428095	1228144492

Таблица 6 – Тестирование алгоритма в наилучшем случае

n	**T(n), мс**	$T_п=C_п+M_п$
100	1	396
1000	1	3996
10000	1	39996
100000	1	399996

n	T(n), мс	$T_п=C_п+M_п$
1000000	3	3999996

Таблица 7 – Тестирование алгоритма в наихудшем случае

n	T(n), мс	$T_п=C_п+M_п$
100	1	10202
1000	1	993240
10000	80	99030152
100000	8425	1310373614
1000000	856645	250002500000

2.3.5 Графики функции роста T(n) двух алгоритмов сортировки

График функций роста алгоритмов в худшем случае представлен на рис. 10.

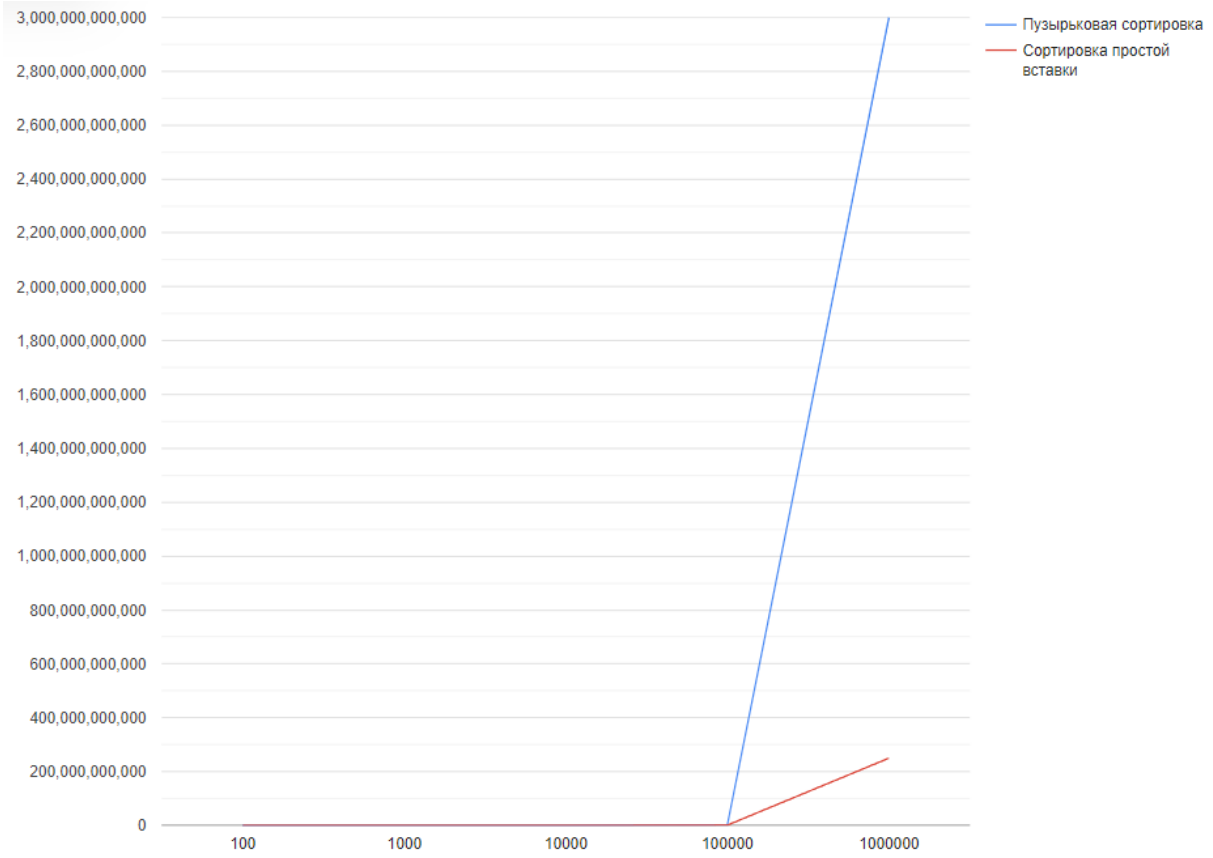

Рисунок 10 – Графики функций роста алгоритмов в худшем случае

График функций роста алгоритмов в лучшем случае представлен на рис. 11.

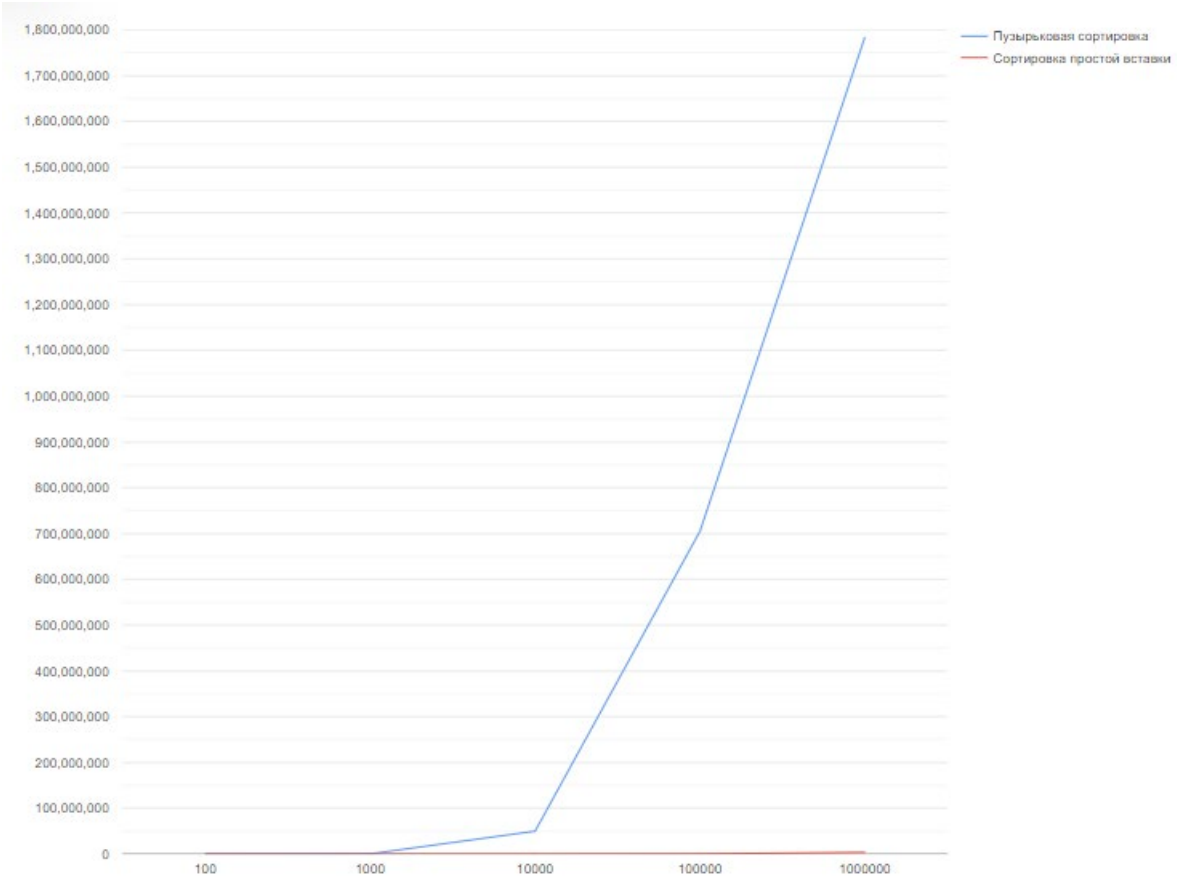

Рисунок 11 - Графики функций роста алгоритмов в лучшем случае

3 ВЫВОДЫ

На основе скорости роста функции роста первого и второго алгоритма, можно сделать вывод о том, что их эмпирическая вычислительная сложность квадратичная.

Время работы обоих алгоритмов также линейно зависит от того, насколько отсортирован первоначальный массив.

На основе графиков функций роста двух алгоритмов, можно сделать вывод о том, что алгоритм пузырьковой сортировки как в лучшем, так и в худшем случае требует все большей вычислительной мощности с ростом значения n, нежели алгоритм простой вставки.

Таким образом, алгоритм сортировки простой вставки эффективнее алгоритма пузырьковой сортировки.

4 ИНФОРМАЦИОННЫЕ ИСТОЧНИКИ

1. Сартаков М.В., ПР-1.1 (Теоретическая сложность алгоритма) М., МИРЭА — Российский технологический университет – 12 с. - URL: https://online-edu.mirea.ru/pluginfile.php?file=%2F1042738%2Fmod_assign%2Fintroattachment%2F0%2FПР1.1%20%28Теоретическая%20сложность%20алгоритма%29.pdf&forcedownload=1 (дата обращения: 15.02.2024). - Режим доступа: Электронно-облачная система – Cloud MIREA РТУ МИРЭА. - Текст: электронный.

2. Рысин М.Л., Сартаков М.В., Туманова М.Б., Введение в структуры и алгоритмы обработки данных. Ч. 1 - учебное пособие, 2022, МИРЭА – Российский технологический университет. – 2022, 109с. – URL: file:///C:/Users/borga/Downloads/Рысин%20М.Л.%20и%20др.%20Введение%20в%20структуры%20и%20алгоритмы%20обработки%20данных.%20Ч.%201%20-%20учебное%20пособие,%202022.pdf (дата обращения: 15.02.2024). – Режим доступа: Электронно-облачная система – Cloud MIREA РТУ МИРЭА. - Текст: электронный.