МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«МИРЭА – Российский технологический университет»

РТУ МИРЭА

Отчет по выполнению практического задания №1.2

Тема:

Эмпирический анализ сложности простых алгоритмов сортировки

Дисциплина: «Структуры и алгоритмы обработки данных»

Выполнил студент: Враженко Д.О.

Группа: ИКБО-10-23

Вариант: 1

Москва – 2024

ЦЕЛЬ РАБОТЫ

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

ХОД РАБОТЫ

1. Задание 1

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

1.1 Алгоритм сортировки

На рис. 1 представлена функция простой сортировки одномерного целочисленного массива, используя алгоритм согласно варианту индивидуального задания.

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

Рисунок 1 - Функция Insertion sort

Пример тестирования при n = 10 представлен на рис. 2.

```
Введите n: 10

Неотсортированный массив:
30 24 56 2 16 67 28 34 38 10

Отсортированный массив:
2 10 16 24 28 30 34 38 56 67
```

Рисунок 2 - Пример тестирования при n = 10

1.2 Функция роста

Оператор	Кол-во выполнений операторов в	
	лучшем случае	худшем случае
for (int i = 1; i < n; i++) {	n	n
int key = array[i];	n-1	n-1
int j = i - 1;	n-1	n-1
while (j >= 0 && array[j] > key) {	2n-2	$\sum_{j=0}^{n-2}(2j+3)$
array[j + 1] = array[j];	0	$\sum_{j=0}^{n-2}(j+1)$
j--; }	0	$\sum_{j=0}^{n-2}(j+1)$
array[j + 1] = key; }	n-1	n-1

В лучшем случае:

$T(n)=6n-5$

В худшем случае:

$T(n)=4n-3+(n+1)(n-1)+n(n-1)=(n-1)(2n+1)+4n-3=2n^2+3n-4$

В среднем случае:

$T(n)_{лучшее} \le T(n)_{среднее} \le T(n)_{худшее}$

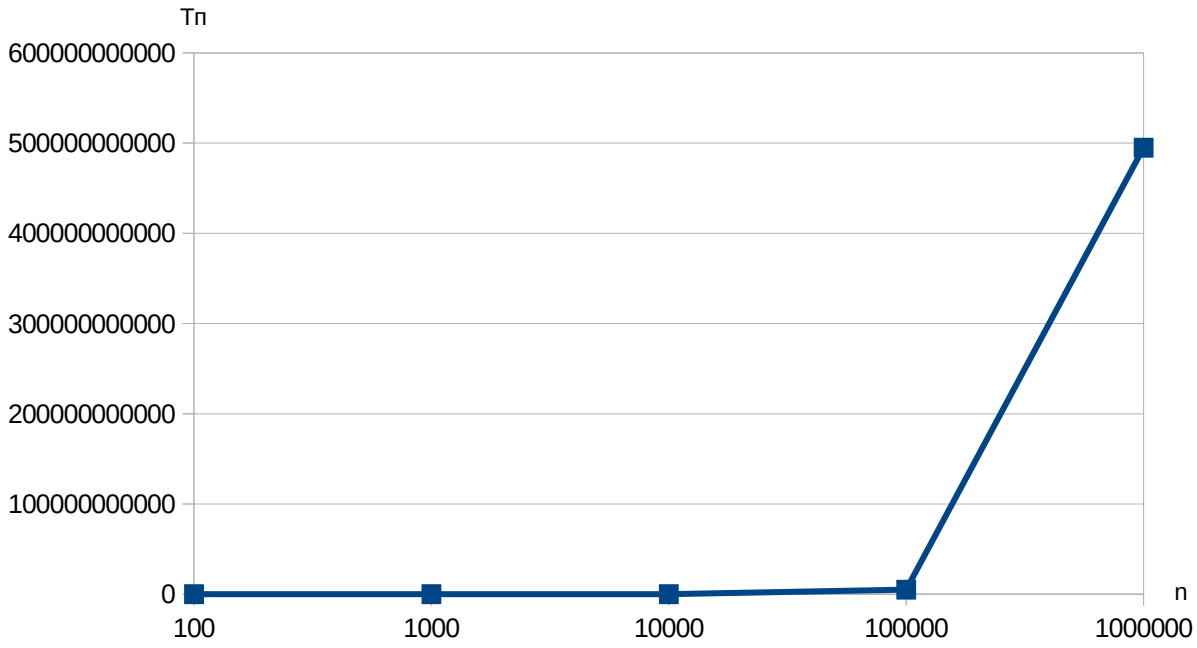
1.3 Сводная таблица

В таблице ниже указаны контрольные замеры программы массивов случайных чисел при n = 100, 1000, 10000 и 1000000.

n	T(n), мс	$T_т = C + M$	$T_п = C_п + M_п$
100	0	-	5195
1000	0	-	474933
10000	54	-	49944575
100000	5255	-	4961308945
1000000	526438	-	494863798701

1.4 График

График в соответствии с таблицей выше представлен снизу.



1.5 Ёмкостная сложность

При работе алгоритма используется некоторое количество дополнительных переменных, а также массив размером n поэтому ёмкостная сложность линейная.

1.6 Вывод об эмпирической сложности алгоритма

На основе функции роста была установлена квадратичная зависимость времени выполнения алгоритма от размера поданного массива n.

2. Задание 2

Оценить вычислительную сложность алгоритма простой сортировки в наихудшем и наилучшем случаях.

2.1 Сводная таблица

В таблице представлены и наилучшие и наихудшие случаи.

n	T(n), мс		$T_т = C + M$		$T_п = C_п + M_п$	
	наилуч-ший	наихуд-ший	наилуч-ший	наихуд-ший	наилуч-ший	наихуд-ший
100	0	0	595	20296	595	20296
1000	0	0	5995	2002996	5995	2002996
10000	0	63	59995	200029996	59995	200029996
100000	0	6301	599995	20000299996	599995	20000299996
1000000	3	632121	5999995	2000002999996	5999995	2000002999996

2.3 Вывод

Проанализировав сводную таблицу, можно сделать вывод о том, что вычислительная сложность алгоритма Insertion sort зависит от исходной упорядоченности массива.

3. Задание 3

Сравнить эффективность алгоритмов простых сортировок

3.1 Алгоритм сортировки

На рис. 3 представлена функция простой сортировки одномерного целочисленного массива, используя алгоритм согласно варианту индивидуального задания.

```
void Selection_sort(int* array, int n) {
    for (int i = 0; i < n - 1; i++) {
        int min_idx = i;
        for (int j = i + 1; j < n; j++)
            if (array[j] < array[min_idx])
                min_idx = j;
        if (min_idx != i)
            swap(array[min_idx], array[i]);
    }
}
```

Рисунок 1 - Функция Selection sort

Пример тестирования при n = 10 представлен на рис. 4.

```
Введите n: 10
Неотсортированный массив:
66 13 26 9 97 35 100 43 85 96

Отсортированный массив:
9 13 26 35 43 66 85 96 97 100
```

Рисунок 2 - Пример тестирования при n = 10

3.2 Функция роста

Оператор	Кол-во выполнений операторов в	
	лучшем случае	худшем случае
for (int i = 0; i < n - 1; i++) {	n	n
int min_idx = i;	n-1	n-1
for (int j = i + 1; j < n; j++)	$\sum_{j=1}^{n-1}(n-j+1)$	$\sum_{j=1}^{n-1}(n-j+1)$
if (array[j] < array[min_idx])	$\sum_{j=1}^{n-1}(n-j)$	$\sum_{j=1}^{n-1}(n-j)$
min_idx = j;	0	$\sum_{i=0}^{t_n}(n-1-2i)$
if (min_idx != i)	n-1	n-1
swap(array[min_idx], array[i]); }	0	3(n-1)

Здесь t_n – это $\frac{n-1}{2}$ для нечетных n и $\frac{n}{2}-1$ для четных n.

В лучшем случае:

$$T(n)=5n-4+\frac{(n+2)*(n-1)}{2}+\frac{n*(n-1)}{2}=n^2+5n-5$$

В худшем случае:

$$T(n)=5n-4+(n+2)\frac{(n-1)}{2}+n\frac{(n-1)}{2}+\frac{n^2-1}{4}=\frac{5n^2+20n-21}{4}$$ для нечетных n.

$$T(n)=5n-4+(n+2)\frac{(n-1)}{2}+n\frac{(n-1)}{2}+\frac{n^2}{4}=\frac{5n^2+20n-20}{4}$$ для четных n.

В среднем случае:

$$T(n)_{лучшее}\le T(n)_{среднее}\le T(n)_{худшее}$$

3.3 Сводные таблицы

Ниже представлена таблица для среднего случая.

n	T(n), мс	$T_т = C + M$	$T_п = C_п + M_п$
100	0	-	10828
1000	3	-	1009854
10000	54	-	100101135
100000	4720	-	10001016493
1000000	469740	-	1000010161409

Ниже представлена таблица для наилучшего и наихудшего случаев. Наилучший случай – массив заполнен возрастающими числами. Наихудший случай – массив заполнен убывающими числами.

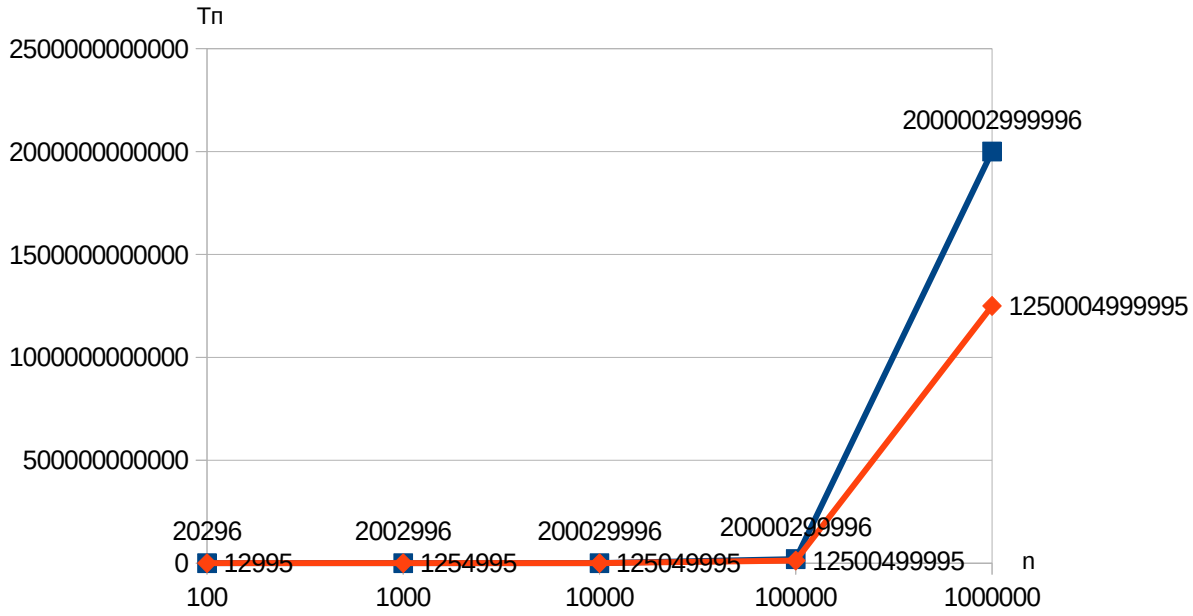
n	T(n), мс		$T_т = C + M$		$T_п = C_п + M_п$	
	наилуч-ший	наихуд-ший	наилуч-ший	наихуд-ший	наилуч-ший	наихуд-ший
100	0	0	10495	12995	10495	12995
1000	0	0	1004995	1254995	1004995	1254995
10000	36	40	100049995	125049995	100049995	125049995
100000	3820	3889	10000499995	12500499995	10000499995	12500499995
1000000	375045	471139	1000004999995	1250004999995	1000004999995	1250004999995

3.4 Ёмкостная сложность алгоритма

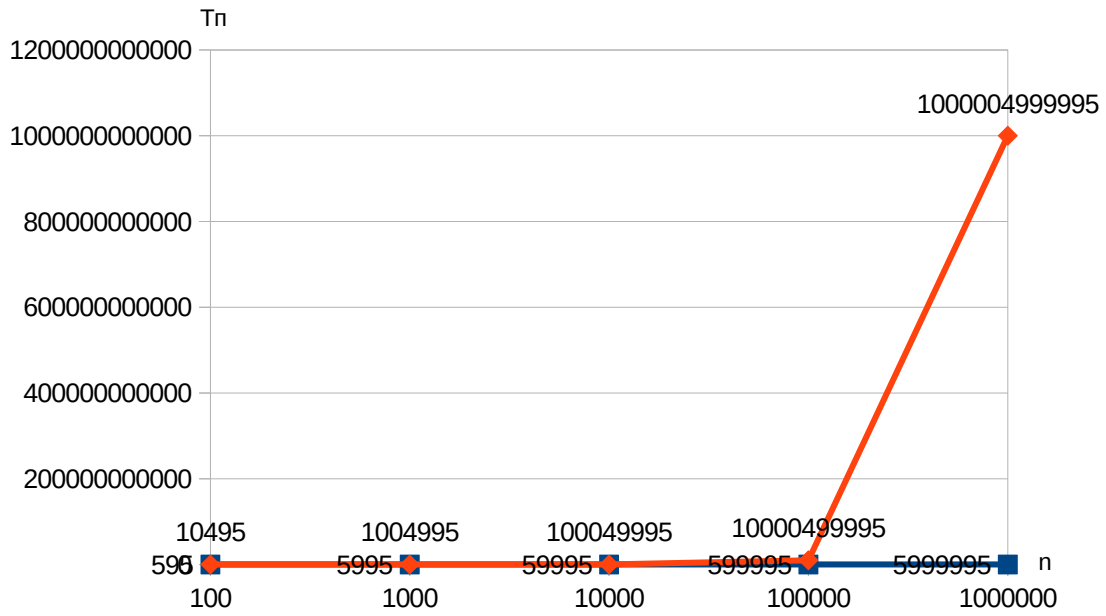
При работе алгоритма используется некоторое количество дополнительных переменных, а также массив размером n поэтому ёмкостная сложность линейная.

3.5 Графики

Изобразим на одном графике функции $T_п(n)$ алгоритмов сортировки Insertion sort и Selection sort в худшем случае.



Изобразим на одном графике функции $T_п(n)$ алгоритмов сортировки Insertion sort и Selection sort в лучшем случае.



3.6 Вывод

На основе полученных данных можно утверждать, что хоть и алгоритм сортировки Insertion sort в худшем случае требует больше вычислительных операций, чем Selection sort, в лучшем случае он требует на порядок меньше операций. Исходя из эмпирических данных более предпочтительно выбирать алгоритм сортировки Insertion sort.