

МИНОБРНАУКИ РОССИИ

*Федеральное государственное бюджетное образовательное учреждение высшего образования*

***«МИРЭА – Российский технологический университет»***

**РТУ МИРЭА**

Отчет по выполнению практического задания № 1.2

Тема:

«Оценка вычислительной сложности алгоритма»

Дисциплина: «Структуры и алгоритмы обработки данных»

Выполнил студент: Cмирнов А.Ю.

Фамилия И.О.

Группа: ИКБО-52-23

Номер группы

Москва – 2024

Москва - 2024

**Оглавление**

[1. Цель работы 3](#_Toc9431)

[2. Постановка задачи 1 3](#_Toc32408)

[3. Постановка задачи 2 4](#_Toc20051)

[4. Постановка задачи 3 4](#_Toc10396)

[5. Индивидуальный вариант 4](#_Toc23561)

[6. Решение задачи 1 5](#_Toc24281)

[6.1 Алгоритм сортировки простого выбора (Selection sort) 5](#_Toc4946)

[6.2 Теоретический подход для оценки функции роста алгоритма 5](#_Toc17050)

[6.3 Контрольные прогоны программы массивов случайных чисел при n = 100, 1000, 10000, 100000 и 1000000 элементов 6](#_Toc1115)

[6.4 Эмпирическая оценка вычислительной сложности алгоритма 7](#_Toc28472)

[6.5 График функции роста алгоритма 7](#_Toc371)

[6.6 Емкостная сложность алгоритма 8](#_Toc26811)

[6.7 Вывод 8](#_Toc21932)

[7. Решение задачи 2 8](#_Toc28780)

[7.1 Оценка вычислительной сложности алгоритма простой сортировки в наихудшем и наилучшем случаях. 8](#_Toc23971)

[7.2 Вывод 10](#_Toc30472)

[8. Решение задачи 3 10](#_Toc31596)

[8.2 Теоретический подход для оценки функции роста алгоритма 11](#_Toc24949)

[8.3 Эмпирическая оценка вычислительной сложности алгоритма 12](#_Toc26354)

[8.4 Емкостная сложность алгоритма 14](#_Toc15393)

[8.5 Графики функции T(n) в худшем случае 15](#_Toc9809)

[8.6 Графики функции T(n) в лучшем случае 15](#_Toc6783)

[8.7 Вывод 16](#_Toc5610)

1. Цель работы

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

1. Постановка задачи 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. 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). Полученные результаты свести в сводную таблицу.

4. Провести эмпирическую оценку вычислительной сложности алгоритма, для чего предусмотреть в программе подсчет фактического количества критических операций Тп как сумму сравнений Сп и перемещений Мп. Полученные результаты вставить в сводную таблицу.

5. Построить график функции роста Тп 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. Постановка задачи 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, результаты представить в сводной таблице

b. строго в возрастающем порядке значений, результат представить в сводной таблице

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

1. Постановка задачи 3

Сравнить эффективность алгоритмов простых сортировок

1. Выполнить разработку и программную реализацию второго алгоритма

согласно индивидуальному варианту

2. Аналогично заданиям 1 и 2 сформировать таблицы с результатами эмпирического исследования второго алгоритма в среднем, лучшем и худшем случаях

3. Определить ёмкостную сложность алгоритма от n.

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

5. Аналогично на другом общем графике отобразить функции Тп(n) двух

алгоритмов сортировки для лучшего случая.

6. Выполнить сравнительный анализ полученных результатов для двух алгоритмов.

1. Индивидуальный вариант

Алгоритмы сортировки:

Задание 1 и 2 - Простого выбора (Selection sort)

Задание 3 - Простого обмена («пузырек», Exchange sort)

1. Решение задачи 1
   1. Алгоритм сортировки простого выбора (Selection sort)

Реализация алгоритма представлена на рисунке 1.

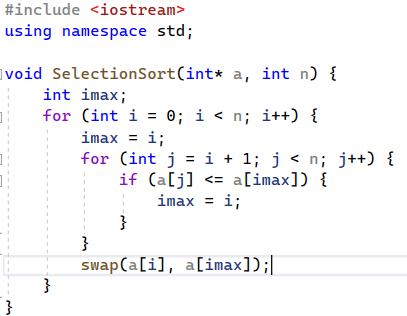


Рисунок 1 - код функции сортировки простого выбора

Тестирование алгоритма при n = 10 представлено на рисунке 2.

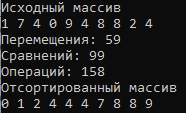


Рисунок 2 - результат тестирования при n = 10

* 1. Теоретический подход для оценки функции роста алгоритма

В худшем случае каждый раз выполняются сравнение, перемещение (внутри вложенного цикла) и swap во внешнем цикле, причём количество обменов M(n) = 3(n - 1), т.к. на каждый обмен требуется 3 операции перемещения.

В лучшем случае (исходный массив уже упорядочен) потребуется поменять местами n-1 элементов.

Сложность алгоритма O(n2) для операций сравнения и O(n) для операций перемещения, т.о. сложность алгоритма O(n2), что не является хорошим результатом.

Таблица 1 - Теоретическая оценка функции роста алгоритма

|  |  |  |
| --- | --- | --- |
| Оператор кода | Количество выполнений оператора | |
| В лучшем случае | В худшем случае |
| for (int i = 0; i < n-1; i++) | n | n |
| imax = i; | n-1 | n-1 |
| for (int j = i + 1; j < n; j++) | (n2-2\*n+1)/2+n-1 | (n2-2\*n+1)/2+n-1 |
| if (a[j] <= a[imax]) | (n2-2\*n+1)/2+n-1 | (n2-2\*n+1)/2+n-1 |
| imax = j; | 0 | (n2-2\*n+1)/2+n-1 |
| swap(a[i], a[imax]); | 3\*(n-1) | 3\*(n-1) |

T(n)(в лучшем случае) = n2 +5n - 5 - квадратичная зависимость - T(n) = n2

T(n)(в худшем случае) = 2n2+6n - 2 - квадратичная зависимость - T(n) = n2

* 1. Контрольные прогоны программы массивов случайных чисел при n = 100, 1000, 10000, 100000 и 1000000 элементов

Выполнение алгоритма с разным количеством случайных элементов.

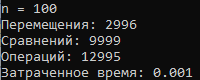


Рисунок 3 - Результаты тестирования при n = 100

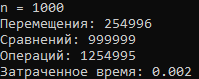


Рисунок 4 - Результаты тестирования при n = 1000

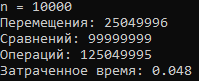


Рисунок 5 - Результаты тестирования при n = 10000

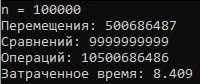


Рисунок 6 - Результаты тестирования при n = 100000

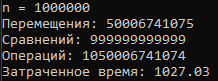


Рисунок 7 - Результаты тестирования при n = 1000000

* 1. Эмпирическая оценка вычислительной сложности алгоритма

Заполним сводную таблицу (табл. 2) результатов с полученных данных (рис. 3 - 7)

Таблица 2 - Сводная таблица результатов

|  |  |  |
| --- | --- | --- |
| n | T(n), с | Тп = Сп + Мп |
| 100 | 0.001 | 12995 |
| 1000 | 0.002 | 1254995 |
| 10000 | 0.048 | 125049995 |
| 100000 | 8.409 | 10500686486 |
| 1000000 | 1027.03 | 1050006741074 |

* 1. График функции роста алгоритма

Построим график функции роста Т(n) этого алгоритма от размера массива n. (рис. 8)

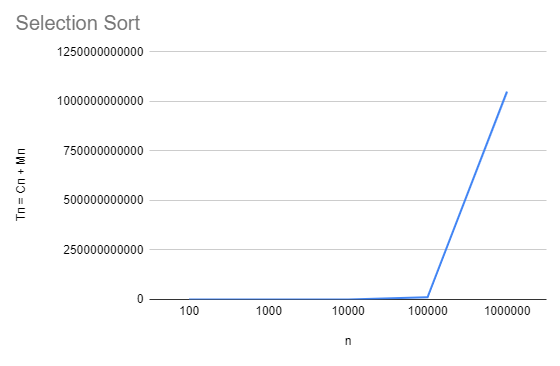


Рисунок 8 - График функции роста алгоритма

* 1. Емкостная сложность алгоритма

Емкостная сложность алгоритма – O(n). В функции используется дополнительная переменная j, которая зависит от переменной i.

* 1. Вывод

Мы можем наблюдать стремительный рост временных затрат при увеличении количества элементов в массиве. Алгоритм имеет сложность в среднем T(n) = n2, что является плохим показателем. Следует вывод, что сортировку простого выбора следует использовать при гарантированном низком количестве элементов.

1. Решение задачи 2
   1. Оценка вычислительной сложности алгоритма простой сортировки в наихудшем и наилучшем случаях.

В наихудшем случае все элементы массива расположены в убывающем порядке, в наилучшем - в возрастающем. Результаты тестирования представлены на рисунках 8-17. В таблице 3 представлены результаты тестирования.

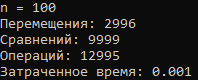


Рисунок 8 - Результаты тестирования при n = 100 в убывающем порядке

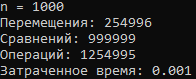


Рисунок 9 - Результаты тестирования при n = 1000 в убывающем порядке

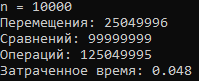


Рисунок 10 - Результаты тестирования при n = 10000 в убывающем порядке

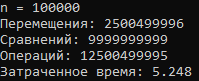


Рисунок 11 - Результаты тестирования при n = 100000 в убывающем порядке

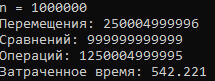


Рисунок 12 - Результаты тестирования при n = 100000 в убывающем порядке

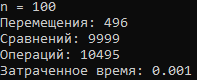


Рисунок 13 - Результаты тестирования при n = 100 в возрастающем порядке

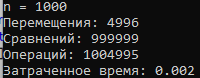


Рисунок 14 - Результаты тестирования при n = 1000 в возрастающем порядке

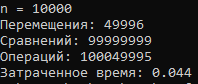


Рисунок 15 - Результаты тестирования при n = 10000 в возрастающем порядке

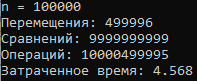


Рисунок 16 - Результаты тестирования при n = 100000 в возрастающем порядке

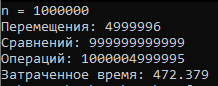


Рисунок 17 - Результаты тестирования при n = 1000000 в возрастающем порядке

Таблица 3 - сводная таблица результатов

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| n | В убывающем  порядке | | В возрастающем порядке | | Случайный массив | |
| T(n), с | Тп = Сп + Мп | T(n), с | Тп = Сп + Мп | T(n), с | Тп = Сп + Мп |
| 100 | 0.001 | 12995 | 0.001 | 10495 | 0.001 | 12995 |
| 1000 | 0.002 | 12554995 | 0.002 | 1004995 | 0.002 | 1254995 |
| 10000 | 0.048 | 125049995 | 0.044 | 100049995 | 0.048 | 125049995 |
| 100000 | 5.248 | 12500499995 | 4.568 | 10000499995 | 8.409 | 10500686486 |
| 1000000 | 542.21 | 1250004999995 | 472.37 | 1000004999995 | 1027.03 | 1050006741074 |

* 1. Вывод

Можно сделать вывод о том, что чем более отсортирован массив, тем быстрее будет выполняться сортировка.

1. Решение задачи 3
   1. Реализация алгоритма сортировки

Алгоритм сортировки простого обмена Exchange Sort представлена на рисунке 18.

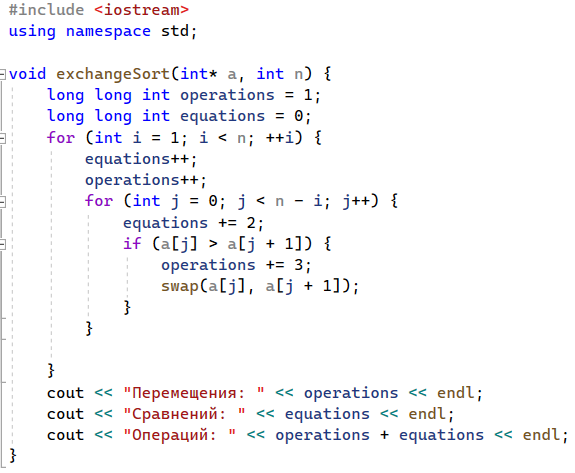


Рисунок 18 - Код алгоритма

Тестирование алгоритма при n = 10 представлено на рисунке 19.

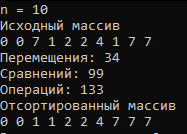


Рисунок 19 - Тестирование алгоритма при n = 10

* 1. Теоретический подход для оценки функции роста алгоритма

В худшем случае каждый раз выполняются сравнение и swap, причём количество обменов M(n) = 3n(n - 1), т.к. на каждый обмен требуется 3 операции перемещения.

В лучшем случае (исходный массив уже упорядочен) swap производить не потребуется.

Сложность алгоритма O(n2) операций сравнения и O(n2) операций перемещения, т.о. в целом сложность алгоритма O(n2), что не является хорошим результатом.

Таблица 4 - Теоретическая оценка функции роста алгоритма

|  |  |  |
| --- | --- | --- |
| Оператор кода | Количество выполнений оператора | |
| В худшем случае | В лучшем случае |
| for (int i = 1; i < n;++i) | n-1 | n-1 |
| for (int j = 0; j < n - 1; j++) | n\*(n-1) | n\*(n-1) |
| if (a[j] > a[j+1]) | n\*(n-1) | n\*(n-1) |
| swap(a[j], a[j+1]); | 3\*n\*(n-1) | 0 |

T(n)(в худшем случае) = 5n2 - 4n - 1 - квадратичная зависимость - T(n) = n2

T(n)(в лучшем случае) = 2n2 - n - 1 - квадратичная зависимость - T(n) = n2

* 1. Эмпирическая оценка вычислительной сложности алгоритма

Выполнение алгоритма с разным количеством случайных элементов.

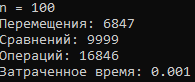


Рисунок 20 - Результаты тестирования при n = 100

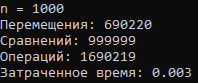


Рисунок 21 - Результаты тестирования при n = 1000

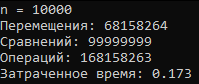


Рисунок 22 - Результаты тестирования при n = 10000

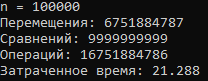


Рисунок 23 - Результаты тестирования при n = 100000

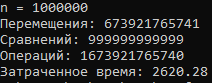


Рисунок 24 - Результаты тестирования при n = 1000000

В наихудшем случае все элементы массива расположены в убывающем порядке, в наилучшем - в возрастающем. Результаты тестирования представлены на рисунках 25-17. В таблице 3 представлены результаты тестирования.

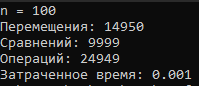


Рисунок 25 - Результаты тестирования при n = 100 в убывающем порядке

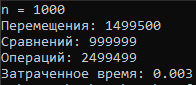


Рисунок 26 - Результаты тестирования при n = 1000 в убывающем порядке

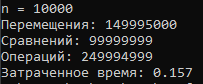


Рисунок 27 - Результаты тестирования при n = 10000 в убывающем порядке

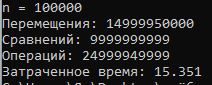


Рисунок 28 - Результаты тестирования при n = 100000 в убывающем порядке

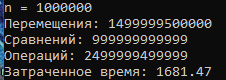


Рисунок 29 - Результаты тестирования при n = 100000 в убывающем порядке

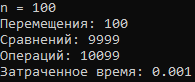


Рисунок 30 - Результаты тестирования при n = 100 в возрастающем порядке

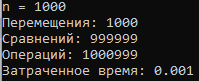


Рисунок 31 - Результаты тестирования при n = 1000 в возрастающем порядке

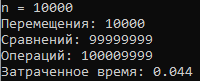


Рисунок 32 - Результаты тестирования при n = 10000 в возрастающем порядке

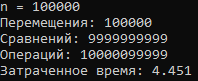


Рисунок 33 - Результаты тестирования при n = 100000 в возрастающем порядке

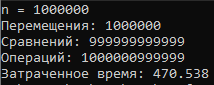


Рисунок 34 - Результаты тестирования при n = 1000000 в возрастающем порядке

Таблица 3 - сводная таблица результатов

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| n | В убывающем  порядке | | В возрастающем порядке | | Случайный массив | |
| T(n), с | Тп = Сп + Мп | T(n), с | Тп = Сп + Мп | T(n), с | Тп = Сп + Мп |
| 100 | 0.001 | 24949 | 0.001 | 10099 | 0.001 | 16846 |
| 1000 | 0.003 | 2499499 | 0.001 | 1000999 | 0.003 | 1690219 |
| 10000 | 0.157 | 249994999 | 0.044 | 100009999 | 0.173 | 168158263 |
| 100000 | 15351 | 24999949999 | 4.451 | 10000099999 | 21.288 | 167518847860 |
| 1000000 | 1681.47 | 2499999499999 | 470.538 | 1000000999999 | 2620.28 | 1673921765740 |

* 1. Емкостная сложность алгоритма

Емкостная сложность алгоритма – O(1). В функции не используются дополнительные переменные.

* 1. Графики функции T(n) в худшем случае

Построим график функции роста Т(n) этих алгоритмов от размера массива n в худшем случае. (рис. 35)

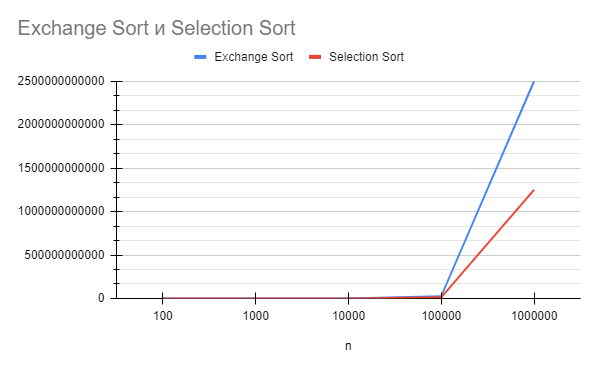


Рисунок 35 - График зависимости T(n)

* 1. Графики функции T(n) в лучшем случае

Построим график функции роста Т(n) этих алгоритмов от размера массива n в лучшем случае. (рис. 36)

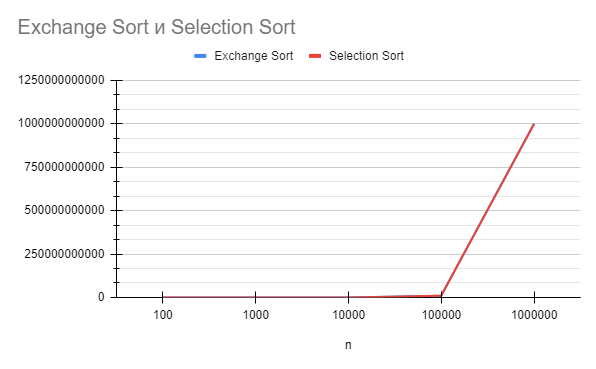


Рисунок 36 - График зависимости T(n)

* 1. Вывод

Исходя из полученных результатов, можно сделать вывод о том, что, имея одинаковую оценочную сложность, алгоритм сортировки Selection Sort более эффективен чем Exchange Sort.