|  |  |
| --- | --- |
| для прик эмбл | |
| МИНОБРНАУКИ РОССИИ | |
| Федеральное государственное бюджетное образовательное учреждение  высшего профессионального образования  **"Московский технологический университет"**  **МИРЭА** | |
| Институт информационных технологий (ИТ) | |
|  | |

|  |  |
| --- | --- |
| **ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ** | |
| **по дисциплине** | |
| **«**Анализ сложности алгоритмов**»** | |
|  | |
| Выполнил студент группы ИКБО-08-16 | Саитов Л.А. |
|  |  |

|  |  |  |
| --- | --- | --- |
| Лабораторные работы выполнены | «\_\_\_»\_\_\_\_\_\_\_2017 г. | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
|  |  |  |
| «Зачтено» | «\_\_\_»\_\_\_\_\_\_\_2017 г. | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |

Москва 2017

Содержание:

[Постановка задачи. 3](#_Toc483515632)

[Алгоритмы сортировок. 4](#_Toc483515633)

Сортировка пузырьком……………………………………………………………………………………………………………………….4

Сортировка вставками………………………………………………………………………………………………………………………..4

Быстрая сортировка………………………………………………………………………………………………………………………..….5

Сортировка с помощью двоичного дерева ………………………………………………………………………………..…….5

[Алгоритмы поиска. 6](#_Toc483515634)

Линейный поиск………………………………………………………………………………………………………………………………....6

Бинарный поиск …………………………………………………………………………………………………………………………..…...6

Интерполяционный поиск………………………………………………………………………………………………………………….6

Поиск по двоичному дереву………………………………………………………………………………………………….……………7

[Ход работы 8](#_Toc483515635)

1. Программа на языке С…………………………………………………………………………………………………………………….8

2. 10000 элементов…………………………………………………………………………………………………………………..…………8

3. 30000 элементов……………………………………………………………………………………………………………………….…….9

4. 50000 элементов……………………………………………………………………………………………………………………………10

5. 70000 элементов……………………………………………………………………………………………………………………………11

6. 90000 элементов……………………………………………………………………………………………………………………………12

[Обработка результатов 13](#_Toc483515636)

[Заключение 15](#_Toc483515637)

## Постановка задачи.

1. Изучить и реализовать алгоритмы сортировок:

* Сортировка пузырьком
* Сортировка вставками
* Быстрая сортировка
* Сортировка с помощью двоичного дерева.

1. Изучить и реализовать алгоритмы поиска:

* Линейный поиск
* Бинарный поиск
* Интерполяционный поиск
* Поиск по двоичному дереву.

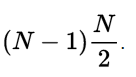
1. Предусмотреть возможность ввода элементов с клавиатуры, чтения из файла.
2. Для каждого алгоритма посчитать время выполнения, количество перестановок и сравнений элементов при работе с файлами, содержащими 10000, 30000, 50000, 70000, 90000. На основе полученных данных сделать таблицу, построить графики.
3. Сделать вывод.

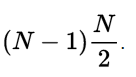
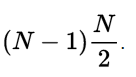
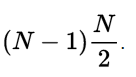
# Алгоритмы сортировок.

Сортировка пузырьком.

Алгоритм состоит из повторяющихся проходов по сортируемому массиву. За каждый проход элементы последовательно сравниваются попарно и, если порядок в паре неверный, выполняется обмен элементов. Проходы по массиву повторяются **{\displaystyle N-1}N-1** раз или до тех пор, пока на очередном проходе не окажется, что обмены больше не нужны, что означает — массив отсортирован.



Сложность: {\displaystyle O(n^{2})} 

* Число сравнений в теле цикла равно {\displaystyle (N-1){\frac {N}{2}}} .
* Число сравнений в заголовках циклов равно {\displaystyle (N-1){\frac {N}{2}}}.
* Суммарное число сравнений равно {\displaystyle (N-1)N}
* Число присваиваний в заголовках циклов равно {\displaystyle (N-1){\frac {N}{2}}}
* Число обменов равно {\displaystyle (N-1){\frac {N}{2}}}

Сортировка вставками.

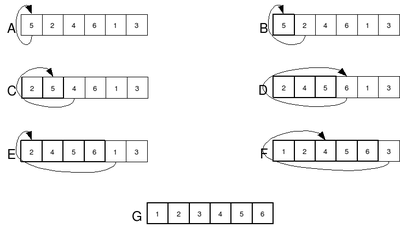


Рисунок 1 Алгоритм сортировки вставками

В начальный момент отсортированная последовательность пуста. На каждом шаге алгоритма выбирается один из элементов входных данных и помещается на нужную позицию в уже отсортированной последовательности до тех пор, пока набор входных данных не будет исчерпан. В любой момент времени в отсортированной последовательности элементы удовлетворяют требованиям к выходным данным алгоритма.

Количество сравнений, очевидно, зависит не только от N, но и от их расположения. В худшем случае элементы массива расположены в обратном порядке, и алгоритм выполняет максимум сравнений и перестановок. В этом случае и количество сравнений, и количество перестановок равно N(N-1)/2.

Быстрая сортировка.

Быстрая сортировка относится к алгоритмам «разделяй и властвуй».

Алгоритм состоит из трёх шагов:

1. Выбрать элемент из массива. Назовём его опорным.
2. *Разбиение*: перераспределение элементов в массиве таким образом, что элементы меньше опорного помещаются перед ним, а больше или равные после.
3. Рекурсивно применить первые два шага к двум подмассивам слева и справа от опорного элемента. Рекурсия не применяется к массиву, в котором только один или отсутствуют элементы.

Сложность в среднем составляет O(n log(n)).

Сортировка с помощью двоичного дерева.

Универсальный алгоритм сортировки, заключающийся в построении двоичного дерева поиска по ключам массива (списка), с последующей сборкой результирующего массива путём обхода узлов построенного дерева в необходимом порядке следования ключей. Данная сортировка является оптимальной при получении данных путём непосредственного чтения с потока (например, из файла, сокета или консоли).

Алгоритм:

1. Построение двоичного дерева.

2. Сборка результирующего массива путём обхода узлов в необходимом порядке следования ключей.

Для n объектов сложность будет составлять O(n log(n)).

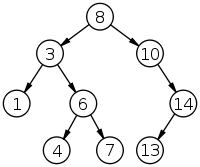


Рисунок 2 Пример двоичного дерева

# Алгоритмы поиска.

Линейный поиск.

Поиск значения функции осуществляется простым сравнением очередного рассматриваемого значения (как правило, поиск происходит слева направо, то есть от меньших значений аргумента к большим) и, если значения совпадают (с той или иной точностью), то поиск считается завершённым.

Асимптотическая сложность алгоритма —

Для списка из n элементов лучшим случаем будет тот, при котором искомое значение равно первому элементу списка и требуется только одно сравнение. Худший случай будет тогда, когда значения в списке нет (или оно находится в самом конце списка), в случае чего необходимо n сравнений.

Бинарный поиск

Алгоритм поиска элемента в отсортированном массиве (векторе), использующий дробление массива на половины.

1. Определение значения элемента в середине структуры данных. Полученное значение сравнивается с ключом.
2. Если ключ меньше значения середины, то поиск осуществляется в первой половине элементов, иначе - во второй.
3. Поиск сводится к тому, что вновь определяется значение серединного элемента в выбранной половине и сравнивается с ключом.
4. Процесс продолжается до тех пор, пока не будет найден элемент со значением ключа или не станет пустым интервал для поиска.

Интерполяционный поиск

Основан на принципе поиска в телефонной книге или, например, в словаре. Вместо сравнения каждого элемента с искомым, как при линейном поиске, данный алгоритм производит предсказание местонахождения элемента: поиск происходит подобно двоичному поиску, но вместо деления области поиска на две части, интерполирующий поиск производит оценку новой области поиска по расстоянию между ключом и текущим значением элемента. Другими словами, бинарный поиск учитывает лишь знак разности между ключом и текущим значением, а интерполирующий ещё учитывает и модуль этой разности и по данному значению производит предсказание позиции следующего элемента для проверки. В среднем интерполирующий поиск производит log(log(N)) операций, где N есть число элементов, среди которых производится поиск.

Поиск по двоичному дереву.

**Задача**: проверить, есть ли узел с ключом K в дереве Т, и если да, то вернуть ссылку на этот узел.

**Алгоритм**:

* Если дерево пусто, сообщить, что узел не найден, и остановиться.
* Иначе сравнить K со значением ключа корневого узла X.
  + Если K=X, выдать ссылку на этот узел и остановиться.
  + Если K>X, рекурсивно искать ключ K в правом поддереве Т.
  + Если K<X, рекурсивно искать ключ K в левом поддереве Т.

# Ход работы:

1. Программа на языке С

При запуске программы пользователю предлагается выбор способа ввода элементов. Если выбрано считывание с файла, то необходимо ввести имя файла. После программа выполняет последовательно все сортировки. Далее предлагается выбрать элемент для поиска. Есть возможность выбрать случайный элемент из массива. После выполняется поиск данного элемента с помощью всех алгоритмов поиска.

Для каждого алгоритма программа считает затраченное на выполнение время, количество сравнений и перестановок.

2. 10000 элементов.

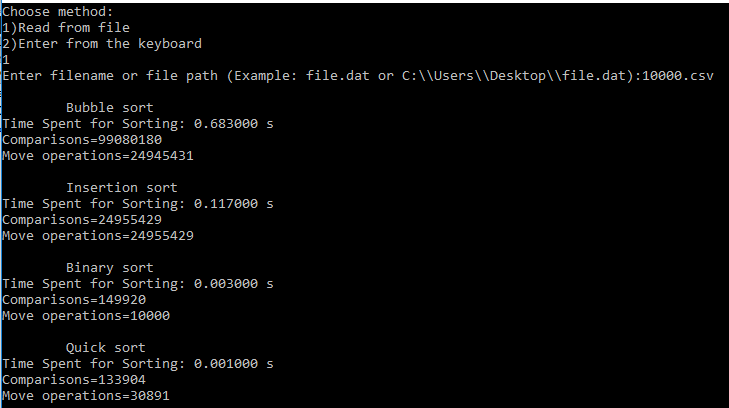


Рисунок 3 Результаты выполнения алгоритмов сортировки для 10000 элементов

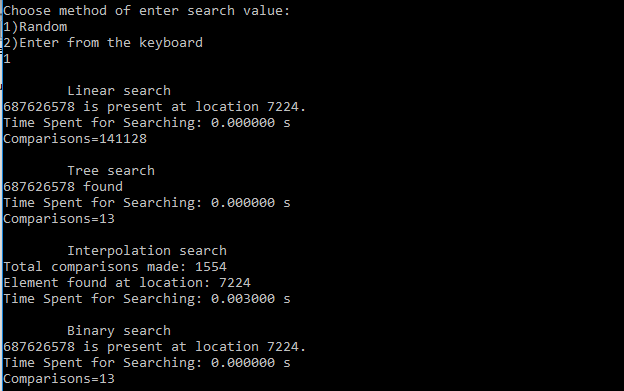


Рисунок 4 Результаты выполнения алгоритмов поиска для 10000 элементов

3. 30000 элементов

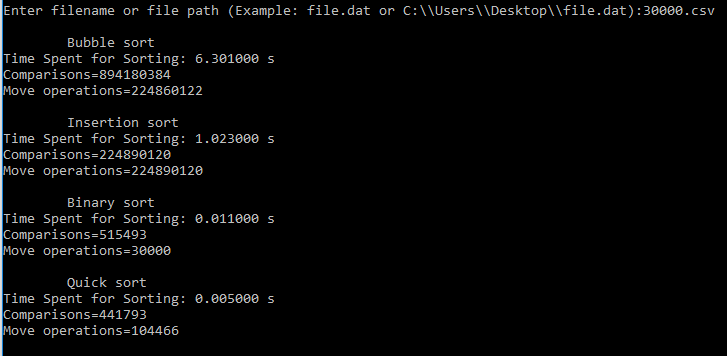


Рисунок 5 Результаты выполнения алгоритмов сортировки для 30000 элементов

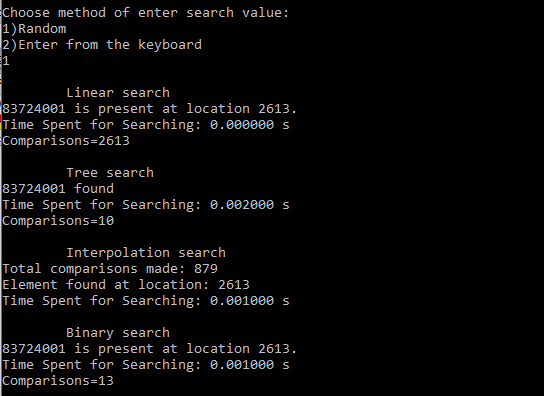


Рисунок 6 Результаты выполнения алгоритмов поиска для 30000 элементов

4. 50000 элементов

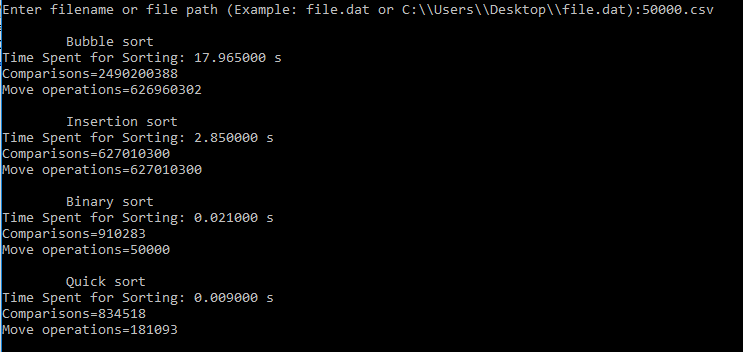


Рисунок 7 Результаты выполнения алгоритмов сортировки для 50000 элементов

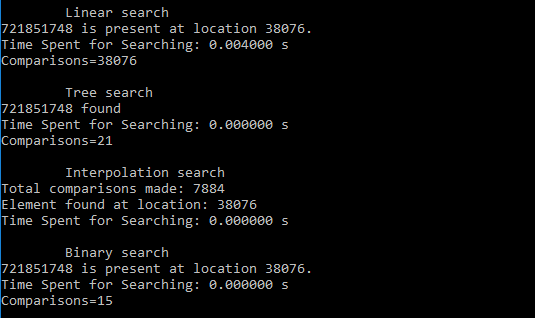


Рисунок 8 Результаты выполнения алгоритмов поиска для 50000 элементов

5. 70000 элементов

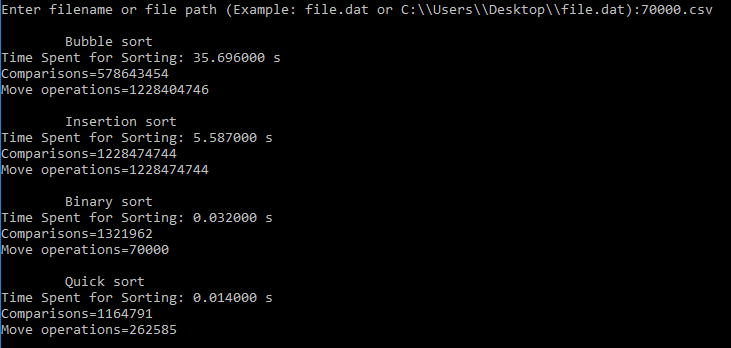


Рисунок 9 Результаты выполнения алгоритмов сортировки для 70000 элементов

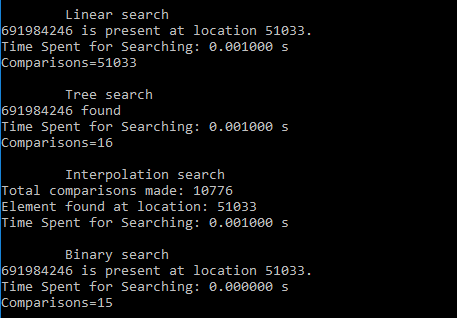


Рисунок 10 Результаты выполнения алгоритмов поиска для 70000 элементов

6. 90000 элементов

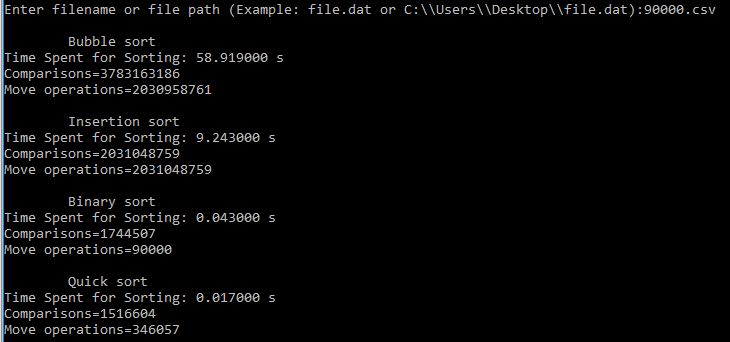


Рисунок 11 Результаты выполнения алгоритмов сортировки для 90000 элементов

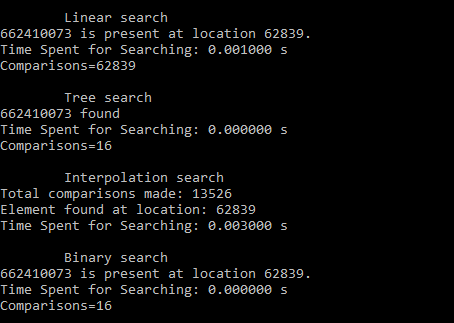


Рисунок 12 Результаты выполнения алгоритмов поиска для 90000 элементов

# Обработка результатов

Время выполнения.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **10000** | **30000** | **50000** | **70000** | **90000** |
| **Сортировка пузырьком** | 0.683 сек | 6.301 сек | 17.965 сек | 35.696 сек | 58.919 сек |
| **Сортировка**  **вставками** | 0.117 сек | 1.023 сек | 2.850 сек | 5.587 сек | 9.243 сек |
| **Сортировка двоичным деревом** | 0.003 сек | 0.011 сек | 0.021 сек | 0.032 сек | 0.043 сек |
| **Быстрая сортировка** | 0.001 сек | 0.005 сек | 0.009 сек | 0.014 сек | 0.017 сек |
| **Линейный**  **поиск** | 0.000 сек | 0.000 сек | 0.004 сек | 0.001 сек | 0.001 сек |
| **Поиск по дереву** | 0.000 сек | 0.002 сек | 0.000 сек | 0.001 сек | 0.000 сек |
| **Интерполяционный поиск** | 0.003 сек | 0.001 сек | 0.000 сек | 0.001 сек | 0.003 сек |
| **Бинарный поиск** | 0.000 сек | 0.001 сек | 0.000 сек | 0.000 сек | 0.000 сек |

Количество сравнений и перестановок

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **10000** | **30000** | **50000** | **70000** | **90000** |
| **Сортировка пузырьком** | 124025611 | 1119040506 | 3117160690 | 1807048200 | 5814121947 |
| **Сортировка**  **вставками** | 49910858 | 449780240 | 1254020600 | 2456949488 | 4062097518 |
| **Сортировка двоичным деревом** | 159920 | 545493 | 960283 | 1391962 | 1834507 |
| **Быстрая сортировка** | 164795 | 546259 | 1015611 | 1427376 | 1862661 |
| **Линейный**  **поиск** | 141128 | 2613 | 38076 | 51033 | 62839 |
| **Поиск по дереву** | 13 | 10 | 21 | 16 | 16 |
| **Интерполяционный поиск** | 1554 | 879 | 7884 | 10776 | 13526 |
| **Бинарный поиск** | 13 | 13 | 15 | 15 | 16 |

# Заключение

Выполнив данную лабораторную работу, я ознакомился с алгоритмами сортировки и поиска, а также с понятием вычислительной сложности алгоритмов. Была показана роль вычислительной сложности алгоритма как определяющего фактора, влияющего на быстродействие решения задач. Данная работа позволяет сделать вывод, что необходимо осознанно подходить к выбору алгоритма для решения задачи.