Министерство образования и науки РФ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Московский технологический университет»

МИРЭА

ИНСТИТУТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

КАФЕДРА ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ

**Отчет по лабораторной работе**

**по дисциплине**

**«Анализ сложности алгоритмов»**

Студент группы ИКБО-05-16 Мальцев М.О

Преподаватель Мирабо Е.И.

Отчет принят и проверен «\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_201\_г.

Москва 2017

**СОДЕРЖАНИЕ**

ВВЕДЕНИЕ4

1. Постановка задачи5
2. Теоретический раздел6
   1. Сортировка простыми обменами6
   2. Быстрая сортировка Хоара7
   3. Сортировка вставками8
   4. Сортировка с помощью двоичного дерева9
   5. Линейный (последовательный поиск)10
   6. Интерполяционный поиск11
   7. Двоичное дерево поиска12
3. Практический раздел13
   1. Блок схема…………………………………………………………..13
   2. Алгоритмы сортировки и поиска14

3.3 Скриншоты……………………………………………………………16

ЗАКЛЮЧЕНИЕ20

**ВВЕДЕНИЕ**

Анализ сложности алгоритмов — бурно развивающаяся область теоретической информатики (theoretical computer science) и охватывает как чисто теоретические вопросы, так и вопросы, непосредственно связанные с практикой. Среди наиболее важных приложений этой теории можно назвать способы построения и анализа эффективных алгоритмов.

Количественная характеристика потребляемых ресурсов, необходимых программе или алгоритму для работы (успешного решения задачи) — это и есть сложность алгоритма.

Основные ресурсы: время (временная сложность) и объем памяти (ёмкостная сложность). Наиболее важной (критической) характеристикой является время. Очевидно, что для разных экземпляров задачи (для разных входных данных) алгоритму может требоваться разное количество ресурсов.

Цель данной работы заключается в реализации популярных алгоритмов на языке программирования и определения наиболее эффективных среди представленных.

1. **ПОСТАНОВКА ЗАДАЧИ**

Анализ вычислительной сложности алгоритма сортировки массива методом простого обмена, быстрой сортировки, сортировки вставками, сортировки по двоичному дереву. Реализовать и исследовать следующие алгоритмы: линейный поиск, интерполяционный поиск, поиск по дереву.

В программе должны быть предусмотрены три варианта формирования исходного массива (выбирается в начале работы программы):

* вводом с клавиатуры (для тестового прогона программы), n=10;
* с помощью генератора псевдослучайных чисел (для рабочего прогона программы), n=10000, 30000, 50000, 70000 и 90000.
* С помощью чтения из файла
* Провести теоретическую и практическую оценки вычислительной сложности алгоритмов.

Примечания:

* Практическая оценка вычислительной сложности алгоритма производится путем вычисления количества выполненных операций сравнения С и перемещения М,
* Формирование исходного массива с помощью генератора псевдослучайных чисел оформить в виде процедуры (функции).

1. **ТЕОРЕТИЧЕСКИЙ РАЗДЕЛ**
   1. **Сортировка простыми обменами**

Сортировка простыми обменами, сортировка пузырьком (англ. bubble sort) — простой алгоритм сортировки. Для понимания и реализации этот алгоритм — простейший, но эффективен он лишь для небольших массивов. Сложность алгоритма: O(n2).

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 N-1 раз или до тех пор, пока на очередном проходе не окажется, что обмены больше не нужны, что означает — массив отсортирован.

При каждом проходе алгоритма по внутреннему циклу, очередной наибольший элемент массива ставится на своё место в конце массива рядом с предыдущим «наибольшим элементом», а наименьший элемент перемещается на одну позицию к началу массива («всплывает» до нужной позиции, как пузырёк в воде, отсюда и название алгоритма).

* 1. **Быстрая сортировка Хоара**

Быстрая сортировка, сортировка Хоара (англ. quicksort), часто называемая qsort (по имени в стандартной библиотеке языка Си) — широко известный алгоритм сортировки, разработанный английским информатиком Чарльзом Хоаром во время его работы в МГУ в 1960 году.

Один из самых быстрых известных универсальных алгоритмов сортировки массивов: в среднем O(n log n) обменов при упорядочении n элементов; из-за наличия ряда недостатков на практике обычно используется с некоторыми доработками.

Общая идея алгоритма состоит в следующем:

* выбрать из массива элемент, называемый опорным. Это может быть любой из элементов массива. От выбора опорного элемента не зависит корректность алгоритма, но в отдельных случаях может сильно зависеть его эффективность (см.ниже);
* сравнить все остальные элементы с опорным и переставить их в массиве так, чтобы разбить массив на три непрерывных отрезка, следующие друг за другом: «меньшие опорного», «равные» и «большие»;
* для отрезков «меньших» и «больших» значений выполнить рекурсивно ту же последовательность операций, если длина отрезка больше единицы.
  1. **Сортировка вставками**

Сортировка вставками (англ. Insertion sort) — алгоритм сортировки, в котором элементы входной последовательности просматриваются по одному, и каждый новый поступивший элемент размещается в подходящее место среди ранее упорядоченных элементов. Вычислительная сложность — O(n2).

На вход алгоритма подаётся последовательность n чисел: a1, a2, …, an. Сортируемые числа также называют ключами. Входная последовательность на практике представляется в виде массива с n элементами. На выходе алгоритм должен вернуть перестановку исходной последовательности a’1, a’2, …, a’n, чтобы выполнялось следующее соотношение a’1 <= a’2 <= … <= a’n.

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.

* 1. **Сортировка с помощью двоичного дерева**

Сортировка с помощью двоичного дерева (сортировка двоичным деревом, сортировка деревом, древесная сортировка, сортировка с помощью бинарного дерева, англ. tree sort) — 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).

Процедура добавления объекта в бинарное дерево имеет среднюю алгоритмическую сложность порядка O(log(n)). Соответственно, для n объектов сложность будет составлять O(n log(n)), что относит сортировку с помощью двоичного дерева к группе «быстрых сортировок».

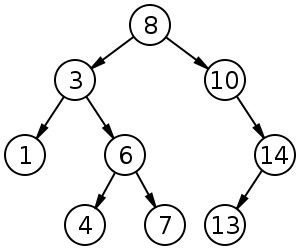


Рисунок 1. Пример двоичного дерева

* 1. **Линейный (последовательный) поиск**

Линейный, последовательный поиск (также известен как поиск методом полного перебора или в простонародье брутфорса) — алгоритм нахождения заданного значения произвольной функции на некотором отрезке.

Данный алгоритм является простейшим алгоритмом поиска и, в отличие, например, от двоичного поиска, не накладывает никаких ограничений на функцию и имеет простейшую реализацию.

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.

Сложность алгоритма – O(n). В связи с малой эффективностью по сравнению с другими алгоритмами линейный поиск обычно используют, только если отрезок поиска содержит очень мало элементов, тем не менее, линейный поиск не требует дополнительной памяти или обработки/анализа функции, так что может работать в потоковом режиме при непосредственном получении данных из любого источника.

Также линейный поиск часто используется в виде линейных алгоритмов поиска максимума/минимума.

* 1. **Интерполяционный поиск**

Интерполяционный поиск (интерполирующий 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. Число необходимых операций зависит от равномерности распределения значений среди элементов.

В плохом случае (например, когда значения элементов экспоненциально возрастают) интерполяционный поиск может потребовать до O(N) операций.

* 1. **Двоичное дерево поиска**

Двоичное дерево поиска (англ. binary search tree, BST) — это двоичное дерево, для которого выполняются следующие дополнительные условия (свойства дерева поиска):

* оба поддерева — левое и правое — являются двоичными деревьями поиска;
* у всех узлов левого поддерева произвольного узла X значения ключей данных меньше, нежели значение ключа данных самого узла X;
* у всех узлов правого поддерева произвольного узла X значения ключей данных больше либо равно, нежели значение ключа данных самого узла X.

Поиск элемента:

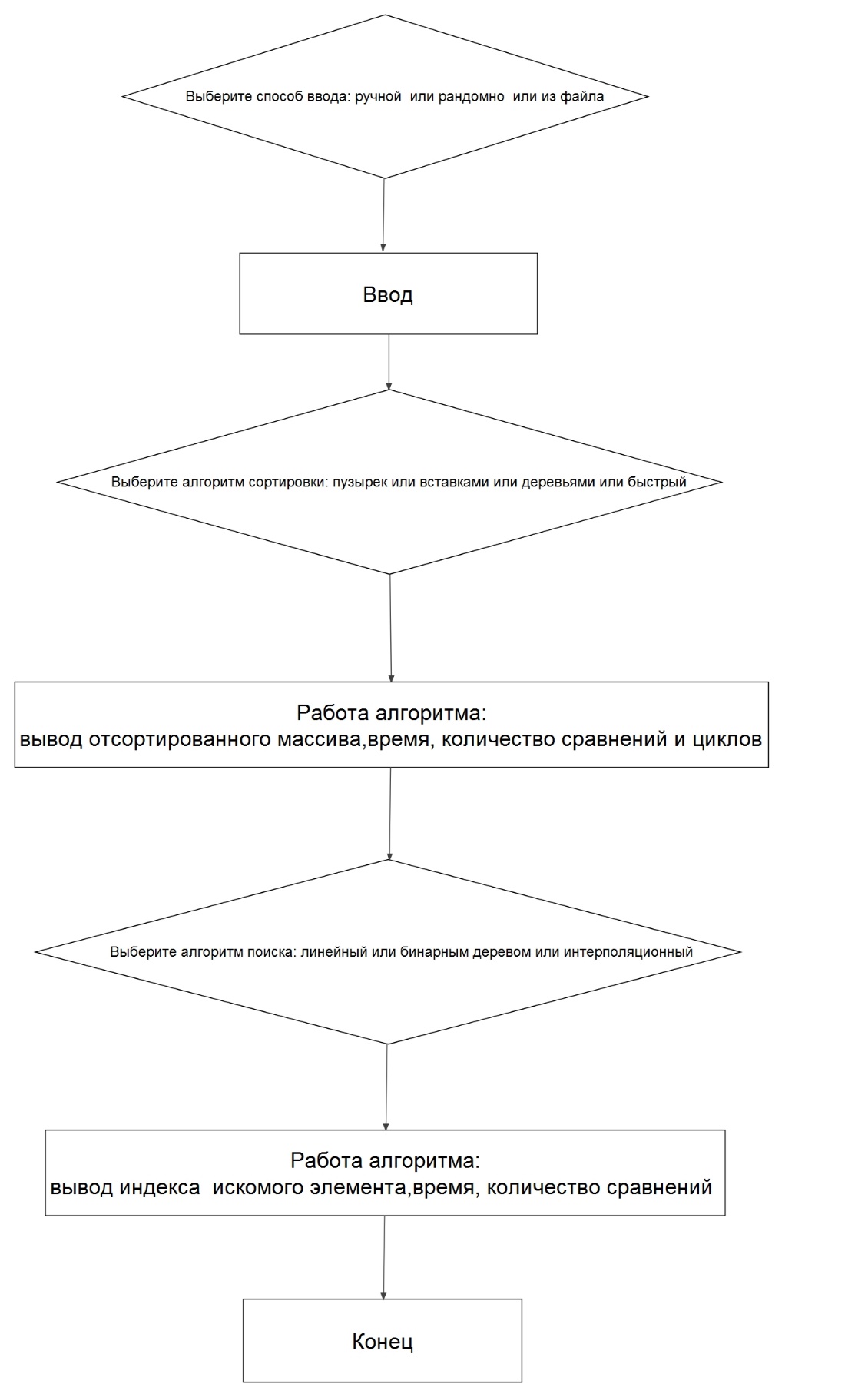
Дано: дерево Т и ключ K.

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.

Блок схема алгоритма:



1. **ПРАКТИЧЕСКИЙ РАЗДЕЛ**
   1. **Алгоритмы сортировки и поиска**

В процессе завершения работы над реализацией алгоритмов была произведена оценка алгоритмов. Были зафиксированы следующие параметры: время , сравнения , переходы в цикле .

Функция случайной генерации чисел создает массив случайных чисел диапазоном от 0 до 10 000.

В результате сравнения каждого алгоритма при разном наборе данных была получена следующая таблица:

Таблица 1. Оценка практической работы алгоритмов сортировок

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Пузырьковая сортировка | «Быстрая» сортировка | Сортировка вставками | Сортировка по бинарному дереву |
| N = 10 000 элементов | | | | |
| Время выполнения (сек.) | 0.304 | 0.002 | 0.095 | 0.035 |
| Кол-во сравнений | 25 341 218 | 45 127 | 25 351 127 | 176033 |
| Кол-во перемещений | 49 995 000 | 100981 | 24 940 636 | 166 034 |
| N = 30 000 элементов | | | | |
| Время выполнения (сек.) | 2.992 | 0.007 | 0.694 | 0.157 |
| Кол-во сравнений | 225 173 350 | 150 460 | 225 203 349 | 560 413 |
| Кол-во перемещений | 449 985 000 | 326 922 | 225 203 349 | 530 414 |

Таблица 1. Продолжение. Оценка практической работы алгоритмов сортировок

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| N = 50 000 элементов | | | | |
| Время выполнения (сек.) | 33.713 | 0.047 | 7.499 | 0.286957 |
| Кол-во сравнений | 624460895 | 263415 | 624460895 | 1005221 |
| Кол-во перемещений | 1249975000 | 566190 | 624510894 | 955222 |
| N = 70 000 элементов | | | | |
| Время выполнения (сек.) | 65.023 | 0.063 | 14.766 | 0.431527 |
| Кол-во сравнений | 1222322490 | 380448 | 1222322490 | 1489770 |
| Кол-во перемещений | 2449965000 | 813918 | 1222392489 | 1419771 |
| N = 90 000 элементов | | | | |
| Время выполнения (сек.) | 111.448 | 0.078 | 24.11 | 0.625672 |
| Кол-во сравнений | 2024746779 | 1066854 | 2024746779 | 2014578 |
| Кол-во перемещений | 4049955000 | 500628 | 2024836778 | 1924579 |

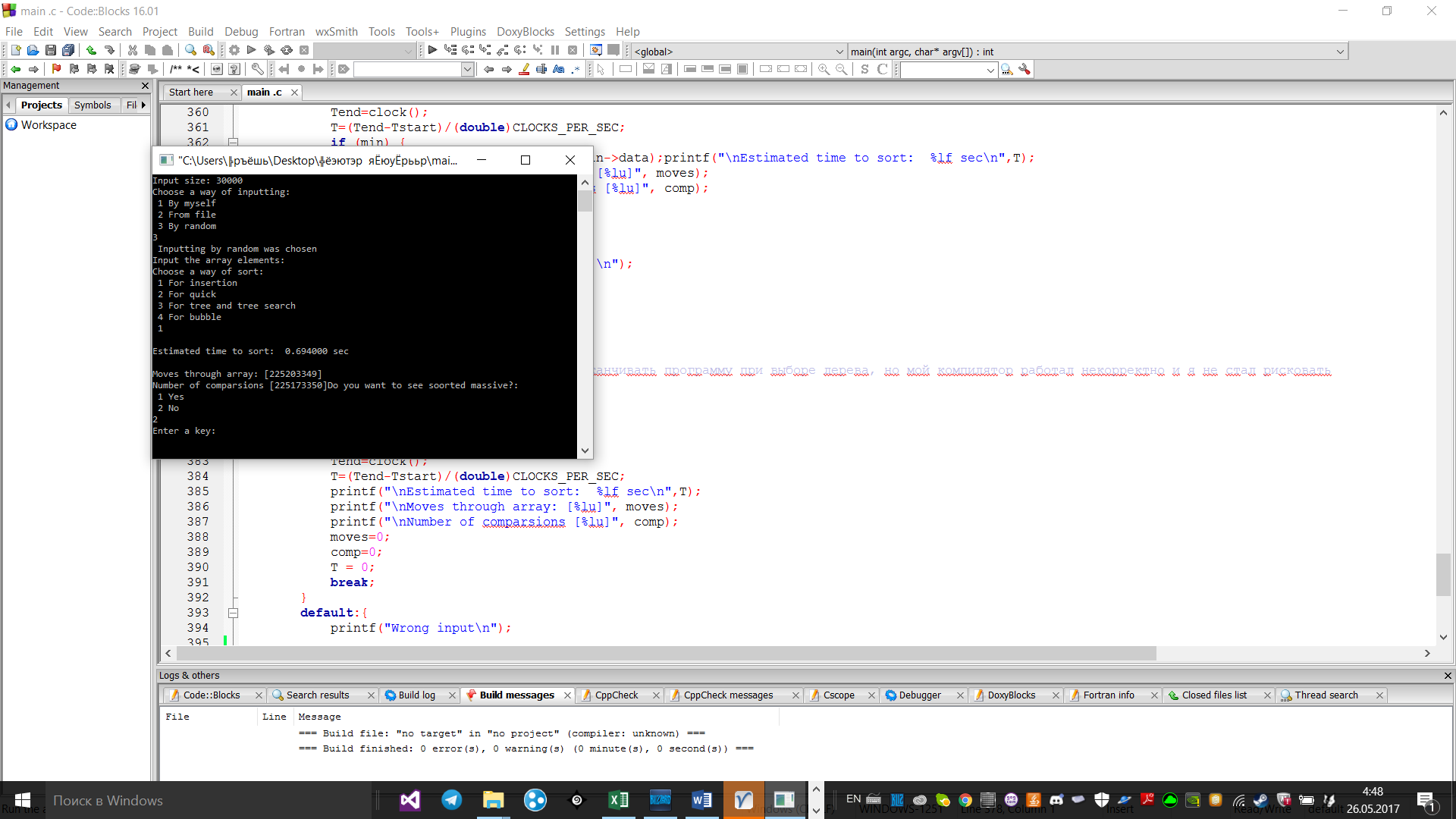
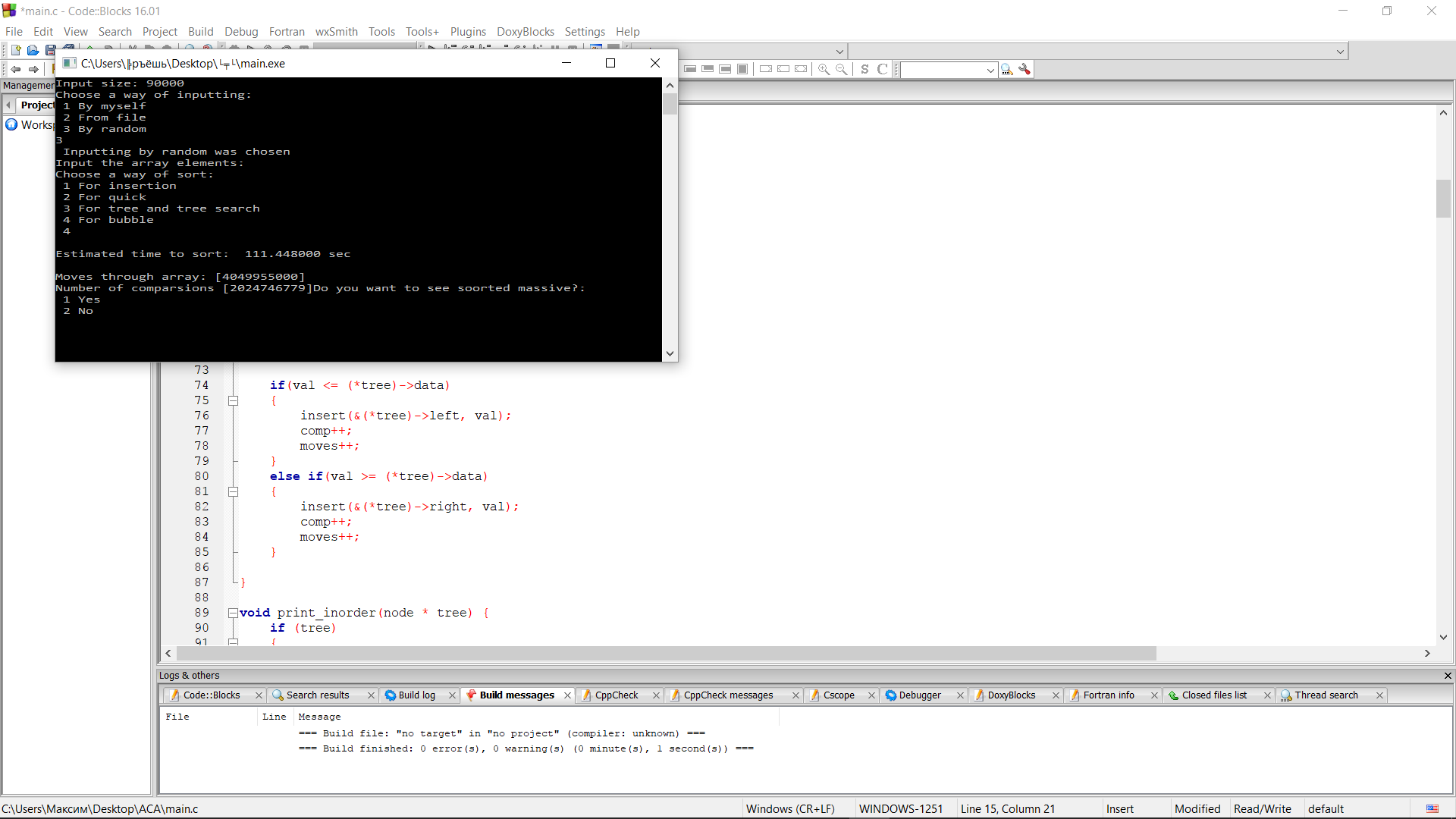
Таблица 2. Оценка практической работы алгоритмов поиска

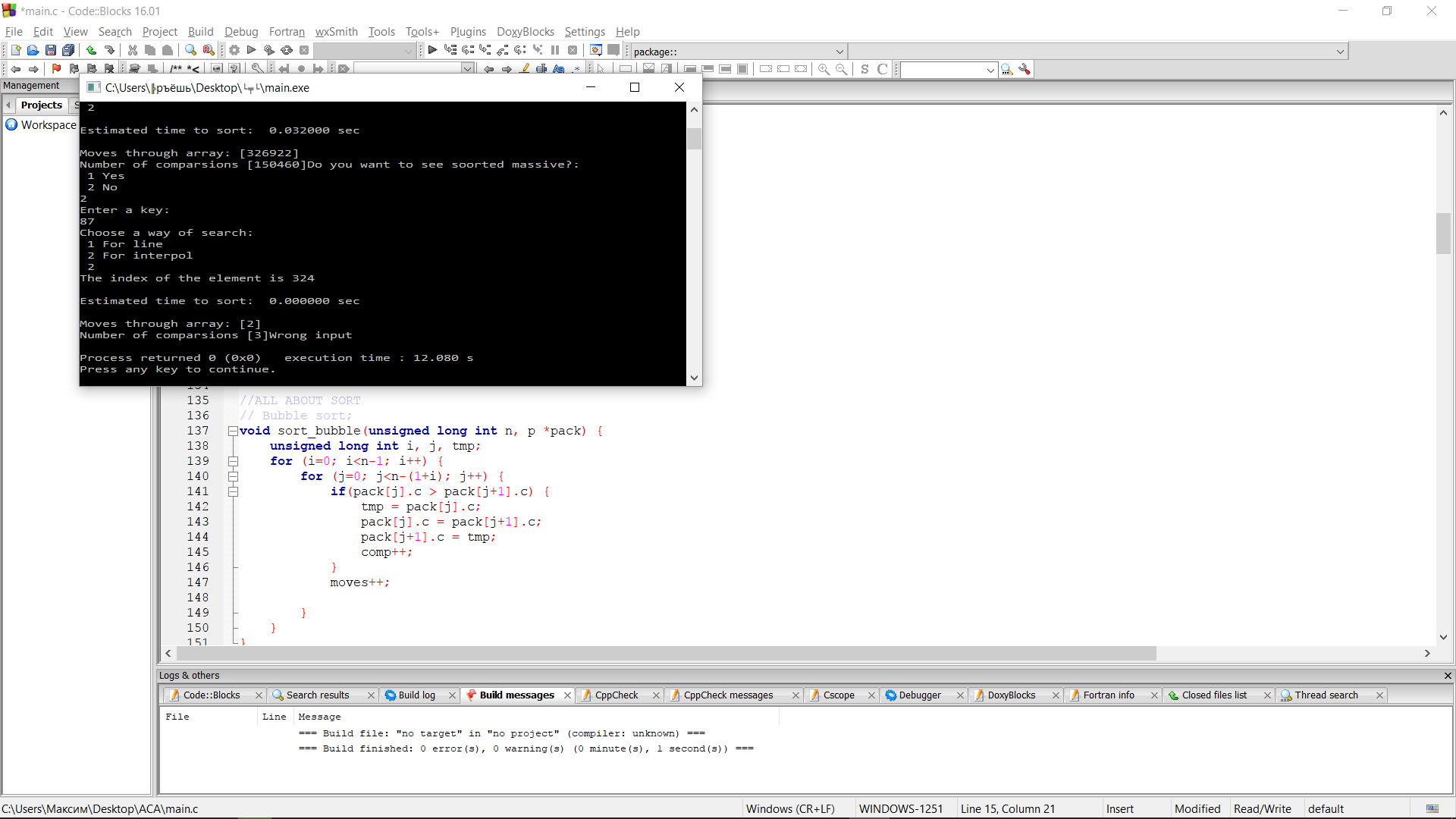
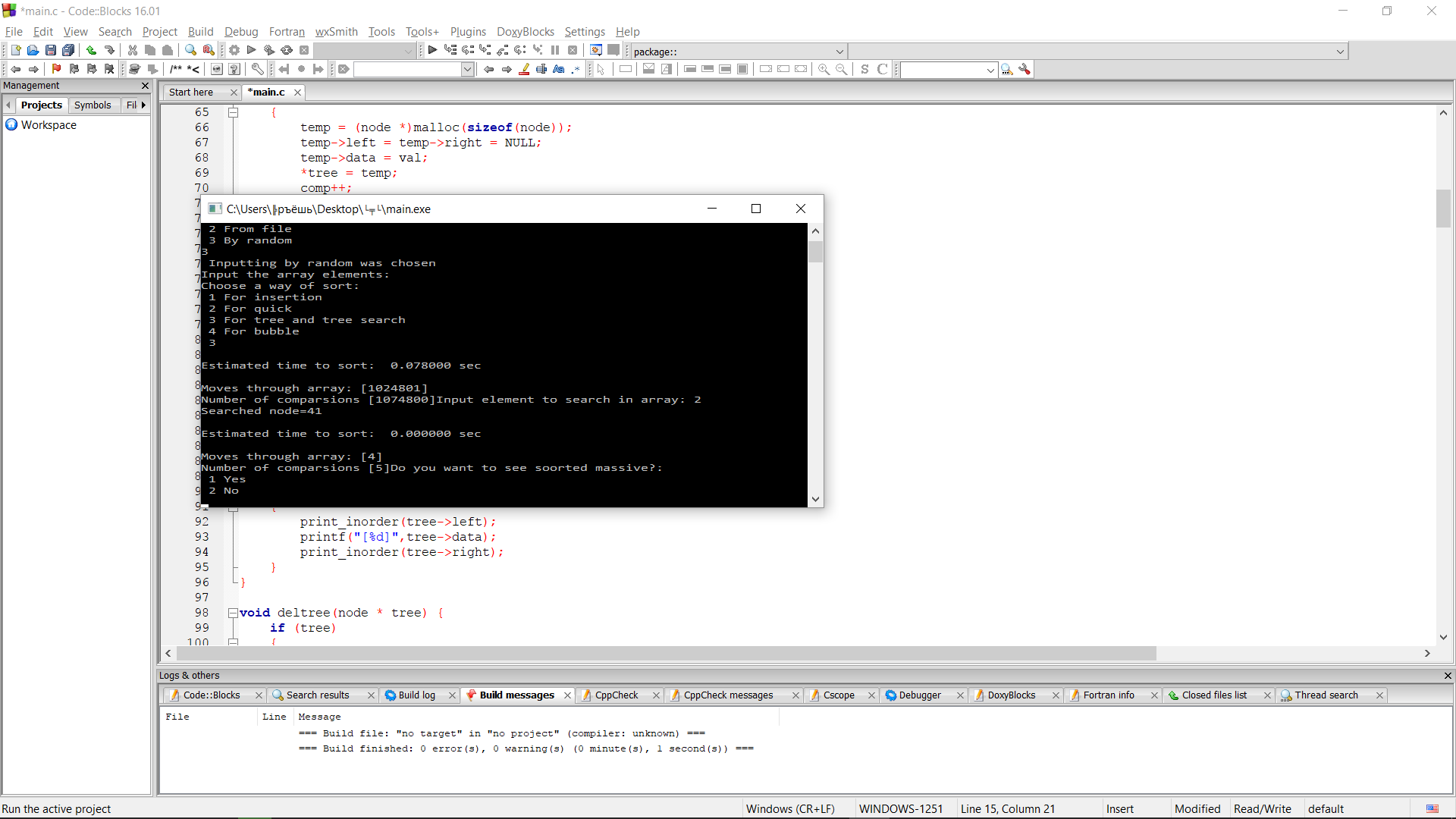
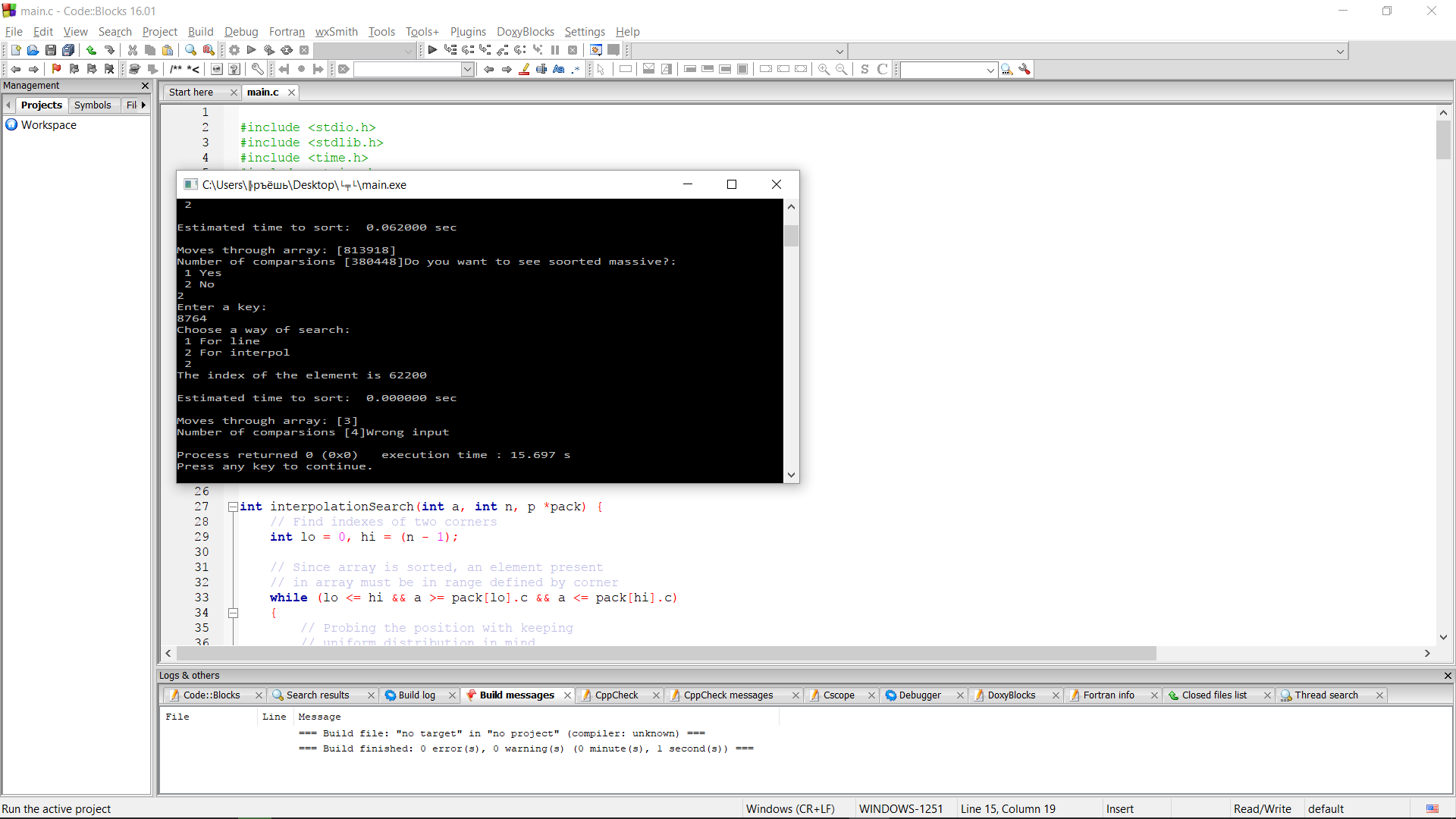
Во всех алгоритмах производился поиск минимального элемента в отсортированном массиве.

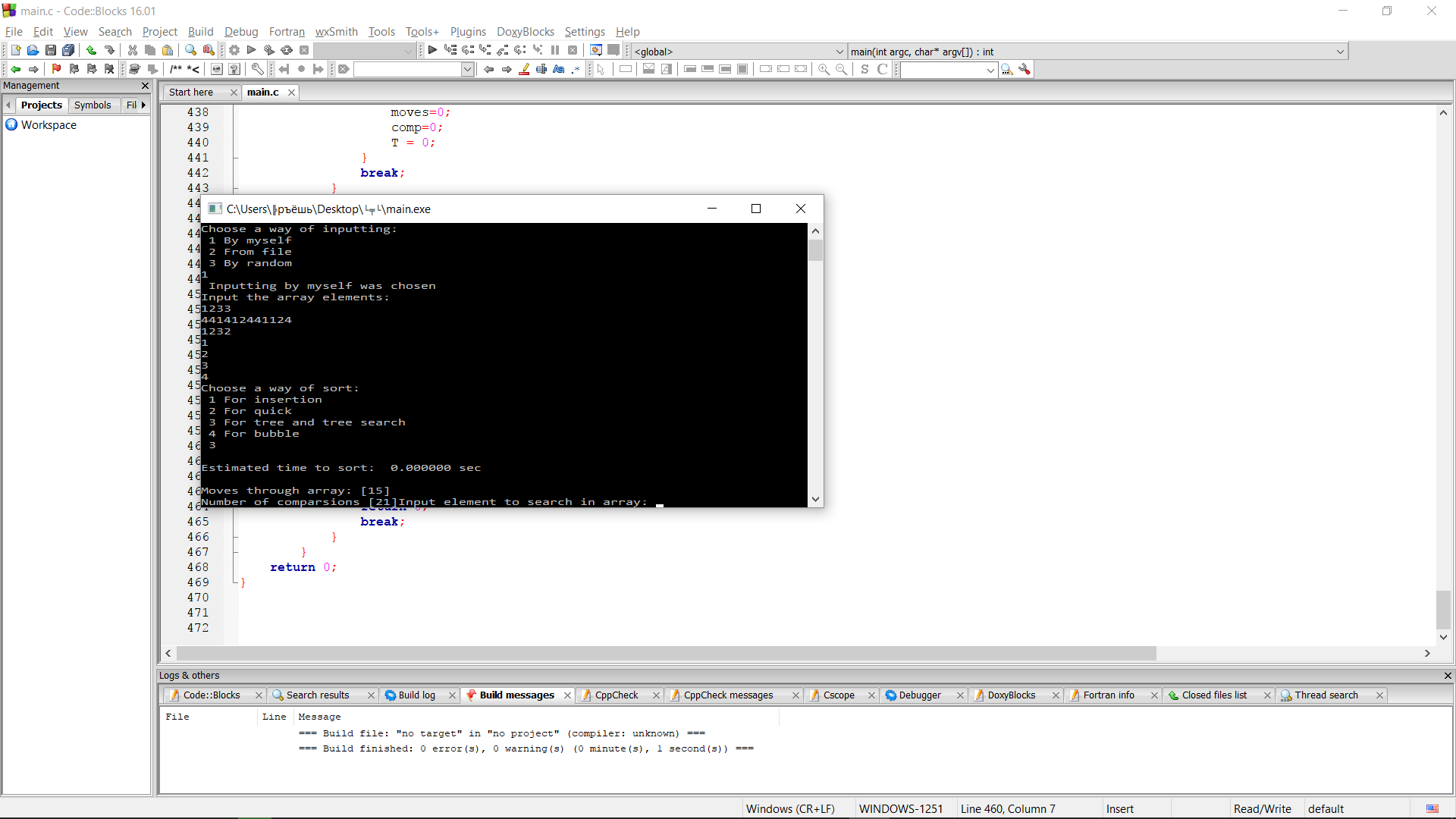
|  |  |  |  |
| --- | --- | --- | --- |
|  | Линейный поиск | Интерполяционный поиск | Поиск по дереву |
| N = 10 000 элементов | | | |
| Время выполнения (сек.) | <0.001 | <0.001 | <0.001 |
| Кол-во сравнений | 1 | 0 | 5 |
| Кол-во перемещений | 9 | 1 | 4 |
| N = 30 000 элементов | | | |
| Время выполнения (сек.) | <0.001 | <0.001 | <0.001 |
| Кол-во сравнений | 1 | 3 | 5 |
| Кол-во перемещений | 3740 | 2 | 4 |
| N = 50 000 элементов | | | |
| Время выполнения (сек.) | <0.001 | <0.001 | <0.001 |
| Кол-во сравнений | 6223 | 3 | 5 |
| Кол-во перемещений | 1 | 4 | 4 |

Таблица 2. Продолжение. Оценка практической работы алгоритмов поиска

|  |  |  |  |
| --- | --- | --- | --- |
| N = 70 000 элементов | | | |
| Время выполнения (сек.) | <0.001 | <0.001 | <0.001 |
| Кол-во сравнений | 1 | 4 | 5 |
| Кол-во перемещений | 10 | 3 | 4 |
| N = 90 000 элементов | | | |
| Время выполнения (сек.) | <0.001 | <0.001 | <0.001 |
| Кол-во сравнений | 1 | 5 | 5 |
| Кол-во перемещений | 11119 | 4 | 4 |

Скриншоты работы программы  
  
  
  
  




**ЗАКЛЮЧЕНИЕ**

В результате изучения теоретических основ, реализации алгоритмов и последующего сравнения их быстродействия были определены эффективные алгоритмы сортировок и поиска.

В условиях лабораторной работы самым эффективным алгоритмом сортировки оказалась «быстрая» сортировка Хоара. Самым эффективным алгоритмом поиска оказался алгоритм поиска по бинарному дереву.

Данные алгоритмы показали наименьшее время на выполнение, а также использовали минимальное количество итераций и сравнений в зависимости от размера массива.