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 Хожеев Н.В.

Преподаватель Мирабо Е.И.

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

Москва 2017

Оглавление

[**Введение** 3](#_Toc483766365)

[**Постановка задачи** 4](#_Toc483766366)

[**Теория** 5](#_Toc483766367)

[Быстрая сортировка Хоара 5](#_Toc483766368)

[Сортировка вставками 6](#_Toc483766369)

[Сортировка методом простого обмена 7](#_Toc483766370)

[Сортировка с помощью бинарного дерева 8](#_Toc483766371)

[Интерполяционный поиск 9](#_Toc483766372)

[Линейный поиск 10](#_Toc483766373)

[Двоичное дерево поиска 11](#_Toc483766374)

[**ПРАКТИЧЕСКИЙ РАЗДЕЛ** 12](#_Toc483766375)

[Сортировка 12](#_Toc483766376)

[Поиск 14](#_Toc483766377)

[**Вывод** 18](#_Toc483766378)

# **Введение**

Анализ сложности алгоритмов — бурно развивающаяся область теоретической информатики. Охватывает как чисто теоретические вопросы, так и вопросы, непосредственно связанные с практикой. Количественная характеристика потребляемых ресурсов, необходимых программе или алгоритму для работы (успешного решения задачи) — это и есть сложность алгоритма. Среди наиболее важных приложений этой теории можно назвать способы построения и анализа эффективных алгоритмов.

Основные ресурсы: время и объем памяти. Наиболее важной характеристикой является время. Очевидно, что для разных экземпляров задачи алгоритму может требоваться разное количество ресурсов.

Целью данной работы является реализация известных алгоритмов на языке программирования и определения наиболее эффективных среди них.

# **Постановка задачи**

Анализировать вычислительную сложность алгоритмов сортировки массива методом быстрой сортировки Хоара, сортировки вставками, простого обмена, сортировки по двоичному дереву. Сделать реализацию и исследовать алгоритмы, линейного поиска, интерполяционного поиска, поиск по бинарному дереву.

В программе будет предоставлено три различных способа формирования изначального массива:

* Ввод элементов массива вручную с клавиатуры
* Заполнение массива при помощи генератора псевдослучайных чисел
* С помощью чтения из файла

Также будет проведена теоретическая и практическая (при помощи расчета количества сравнений и перемещений) оценка вычислительной сложности алгоритмов.

# **Теория**

## Быстрая сортировка Хоара

Быстрая сортировка, сортировка Хоара, часто называемая qsort (по имени в стандартной [библиотеке](https://ru.wikipedia.org/wiki/%D0%91%D0%B8%D0%B1%D0%BB%D0%B8%D0%BE%D1%82%D0%B5%D0%BA%D0%B0_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) языка [Си](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F))) — широко известный [алгоритм сортировки](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D1%81%D0%BE%D1%80%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%BA%D0%B8), разработанный английским информатиком [Чарльзом Хоаром](https://ru.wikipedia.org/wiki/%D0%A5%D0%BE%D0%B0%D1%80,_%D0%A7%D0%B0%D1%80%D0%BB%D1%8C%D0%B7_%D0%AD%D0%BD%D1%82%D0%BE%D0%BD%D0%B8_%D0%A0%D0%B8%D1%87%D0%B0%D1%80%D0%B4).

[Один из самых быстрых](https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D0%BE%D1%80%D0%B8%D1%8F_%D1%81%D0%BB%D0%BE%D0%B6%D0%BD%D0%BE%D1%81%D1%82%D0%B8_%D0%B2%D1%8B%D1%87%D0%B8%D1%81%D0%BB%D0%B5%D0%BD%D0%B8%D0%B9) известных универсальных алгоритмов сортировки массивов.

QuickSort является существенно улучшенным вариантом алгоритма сортировки с помощью прямого обмена, известного, в том числе, своей низкой эффективностью.

Принципиальное отличие состоит в том, что в первую очередь производятся перестановки на наибольшем возможном расстоянии и после каждого прохода элементы делятся на две независимые группы. Любопытный факт: улучшение самого неэффективного прямого метода сортировки дало в результате один из наиболее эффективных улучшенных методов.

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.

## Сортировка вставками

Сортировка вставками — достаточно простой алгоритм. Как в и любом другом алгоритме сортировки, с увеличением размера сортируемого массива увеличивается и время сортировки. Основным преимуществом алгоритма сортировки вставками является возможность сортировать массив по мере его получения.То есть имея часть массива, можно начинать его сортировать. В параллельном программирование такая особенность играет не маловажную роль.

Сортируемый массив можно разделить на две части — отсортированная часть и неотсортированная. В начале сортировки первый элемент массива считается отсортированным, все остальные — не отсортированные. Начиная со второго элемента массива и заканчивая последним, алгоритм вставляет неотсортированный элемент массива в нужную позицию в отсортированной части массива. Таким образом, за один шаг сортировки отсортированная часть массива увеличивается на один элемент, а неотсортированная часть массива уменьшается на один элемент.

{\displaystyle a\_{1}^{'}\leqslant a\_{2}^{'}\leqslant ...\leqslant a\_{n}^{'}}

## Сортировка методом простого обмена

Сортировка простыми обменами, сортиро́вка пузырько́м ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) bubble sort) — простой [алгоритм сортировки](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D1%81%D0%BE%D1%80%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%BA%D0%B8). Для понимания и реализации этот алгоритм — простейший, но эффективен он лишь для небольших массивов.

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. 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).

## Сортировка с помощью бинарного дерева

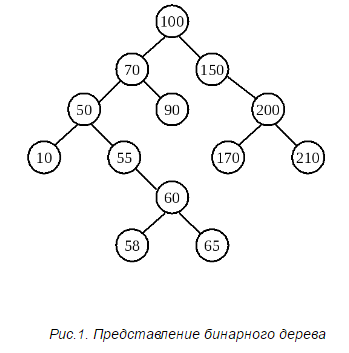
Сортировка с помощью двоичного дерева (сортировка двоичным деревом, сортировка деревом, древесная сортировка, сортировка с помощью бинарного дерева, [англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) tree sort) — универсальный [алгоритм сортировки](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D1%81%D0%BE%D1%80%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%BA%D0%B8), заключающийся в построении [двоичного дерева поиска](https://ru.wikipedia.org/wiki/%D0%94%D0%B2%D0%BE%D0%B8%D1%87%D0%BD%D0%BE%D0%B5_%D0%B4%D0%B5%D1%80%D0%B5%D0%B2%D0%BE_%D0%BF%D0%BE%D0%B8%D1%81%D0%BA%D0%B0) 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](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D1%82%D0%BE%D0%BA_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85) (например из файла, сокета или консоли).

Алгоритм:

1. Построение двоичного дерева.

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

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». Однако, сложность добавления объекта в разбалансированное дерево может достигать O(n), что может привести к общей сложности порядка O(n²).

При физическом развёртывании древовидной структуры в памяти требуется не менее чем 4n ячеек дополнительной памяти (каждый узел должен содержать ссылки на элемент исходного массива, на родительский элемент, на левый и правый лист), однако, существуют способы уменьшить требуемую дополнительную память. 

## 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](https://ru.wikipedia.org/wiki/%D0%9B%D0%B8%D0%BD%D0%B5%D0%B9%D0%BD%D1%8B%D0%B9_%D0%BF%D0%BE%D0%B8%D1%81%D0%BA), данный алгоритм производит предсказание местонахождения элемента: поиск происходит подобно [двоичному поиску](https://ru.wikipedia.org/wiki/%D0%94%D0%B2%D0%BE%D0%B8%D1%87%D0%BD%D1%8B%D0%B9_%D0%BF%D0%BE%D0%B8%D1%81%D0%BA), 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.

На практике интерполяционный поиск часто быстрее бинарного, так как с вычислительной стороны их отличают лишь применяемые арифметические операции: [интерполирование](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D0%BF%D0%BE%D0%BB%D1%8F%D1%86%D0%B8%D1%8F) — в интерполирующем поиске и деление на два — в двоичном, а скорость их вычисления отличается незначительно, с другой стороны интерполирующий поиск использует такое принципиальное свойство данных, как однородность распределения значений. Ключом может быть не только номер, число, но и, например, текстовая строка, тогда становится понятна аналогия с телефонной книгой: если мы ищем имя в телефонной книге, начинающееся на «А», следовательно, нужно искать его в начале, но никак не в середине. В принципе, ключом может быть всё что угодно, так как те же строки, например, запросто кодируются посимвольно, в простейшем случае символ можно закодировать значением от 1 до 33 (только русские символы) или, например, от 1 до 26 (только латинский алфавит) и т. д.

## Линейный поиск

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, т.е. от меньших значений аргумента к большим) и, если значения совпадают (с той или иной точностью), то поиск считается завершенным. 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. Также, линейный поиск часто используется в виде линейных алгоритмов поиска максимума/минимума.

Алгоритм последовательного поиска:

Шаг1. Полагаем, что значение переменной цикла i=0.

Шаг2. Если значение элемента массива x[i] равно значению ключа key, то возвращаем значение, равное номеру искомого элемента, и алгоритм завершает работу. В противном случае значение переменной цикла увеличивается на единицу i=i+1.

Шаг3. Если i<k, где k-число элементов массива х, то выполняется шаг2, в противном случае – работа алгоритма завершена и возвращается значение равное -1. При наличии в массиве нескольких элементов со значением key алгоритм находит только первый из них (с наименьшим индексом).

## Двоичное дерево поиска

Двоичное дерево поиска — это [двоичное дерево](https://ru.wikipedia.org/wiki/%D0%94%D0%B2%D0%BE%D0%B8%D1%87%D0%BD%D0%BE%D0%B5_%D0%B4%D0%B5%D1%80%D0%B5%D0%B2%D0%BE), 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.

Очевидно, данные в каждом узле должны обладать ключами, на которых определена операция сравнения меньше.

Как правило, информация, представляющая каждый узел, является записью, а не единственным полем данных. Однако это касается реализации, а не природы двоичного дерева поиска.

Для целей реализации двоичное дерево поиска можно определить так:

* Двоичное дерево состоит из узлов (вершин) — записей вида (data, left, right), где data — некоторые данные, привязанные к узлу, left и right — ссылки на узлы, являющиеся детьми данного узла — левый и правый сыновья соответственно. Для оптимизации алгоритмов конкретные реализации предполагают также определения поля parent в каждом узле (кроме корневого) — ссылки на родительский элемент.
* Данные (data) обладают ключом (key), на котором определена операция сравнения «меньше». В конкретных реализациях это может быть пара (key, value) — (ключ и значение), или ссылка на такую пару, или простое определение операции сравнения на необходимой структуре данных или ссылке на неё.
* Для любого узла X выполняются свойства дерева поиска: key[left[X]] < key[X] ≤ key[right[X]], то есть ключи данных родительского узла больше ключей данных левого сына и нестрого меньше ключей данных правого.

# **ПРАКТИЧЕСКИЙ РАЗДЕЛ**

В ходе работы была проведена оценка алгоритмов сортировки и поиска, путем фиксирования таких параметров, как время работы алгоритма, количество сравнений, количество переходов в цикле.

В результате анализа алгоритмов при разных наборах данных мы получили:

## Сортировка

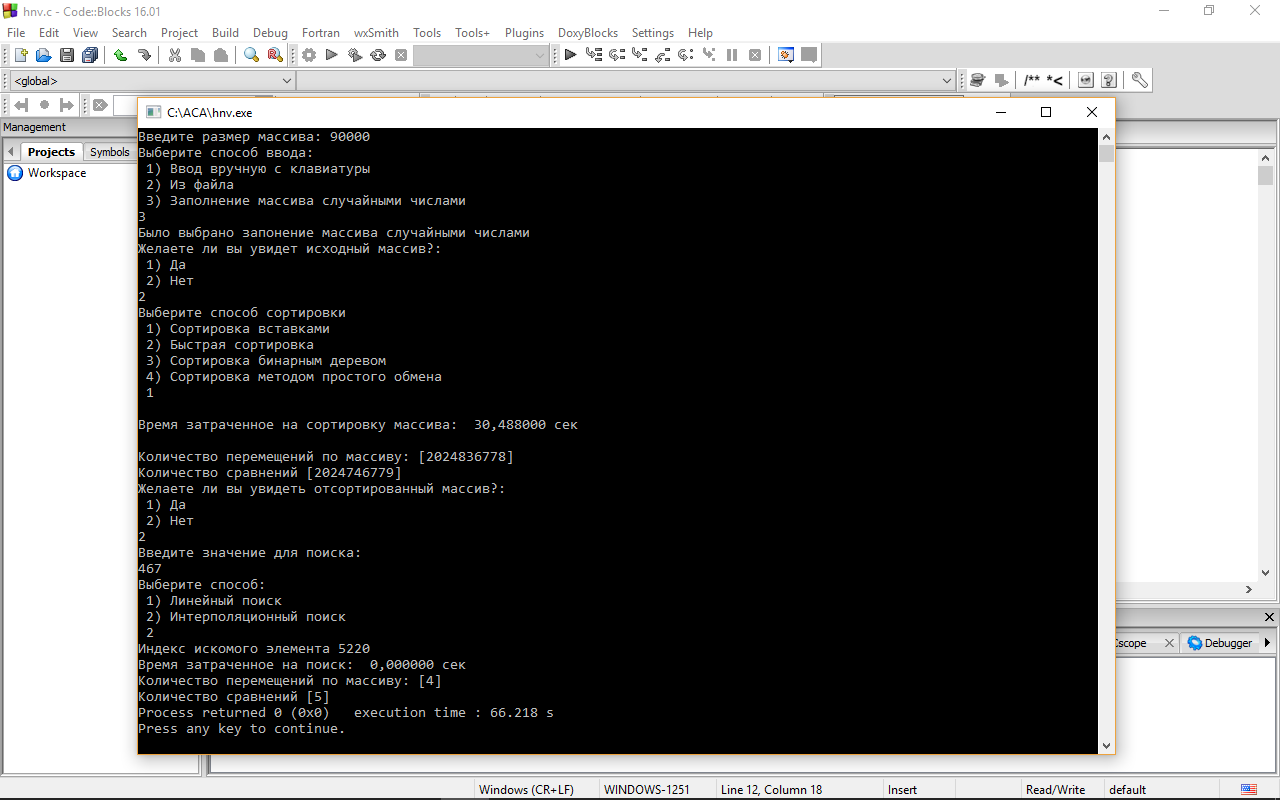
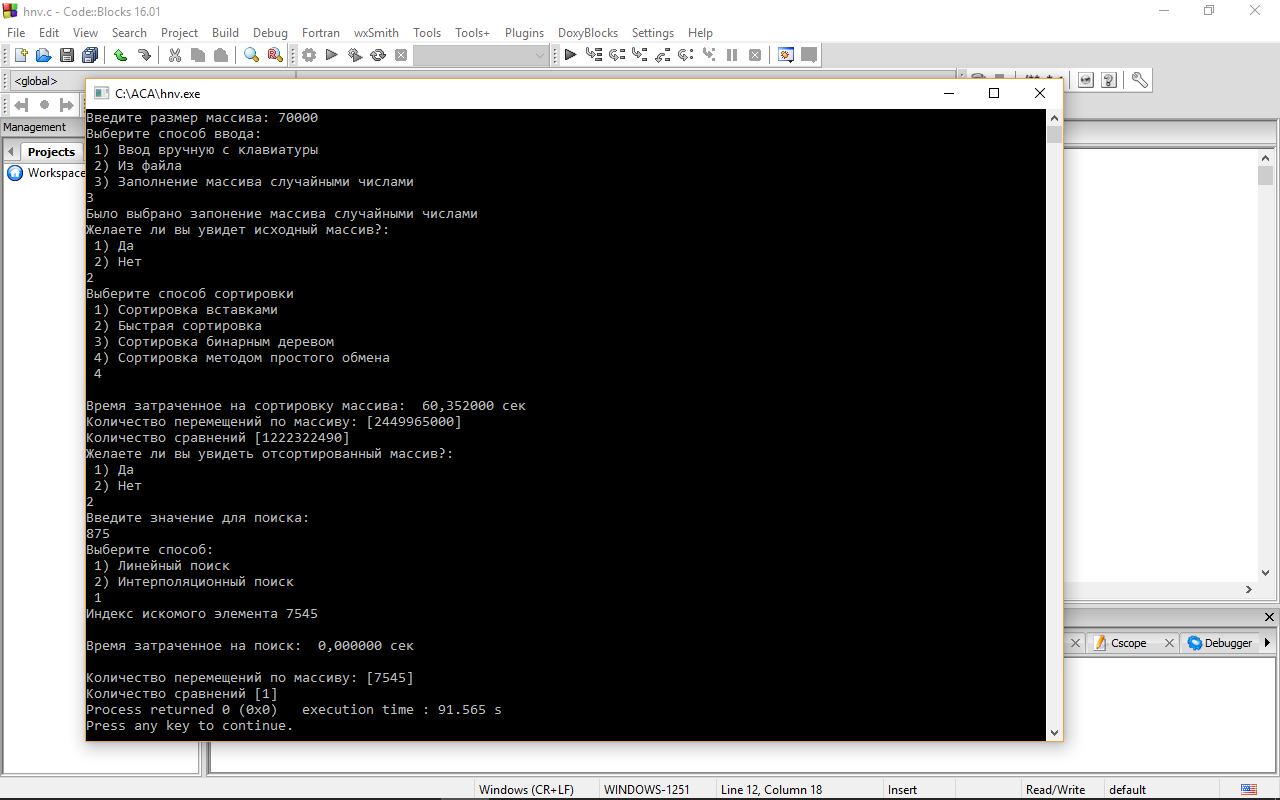
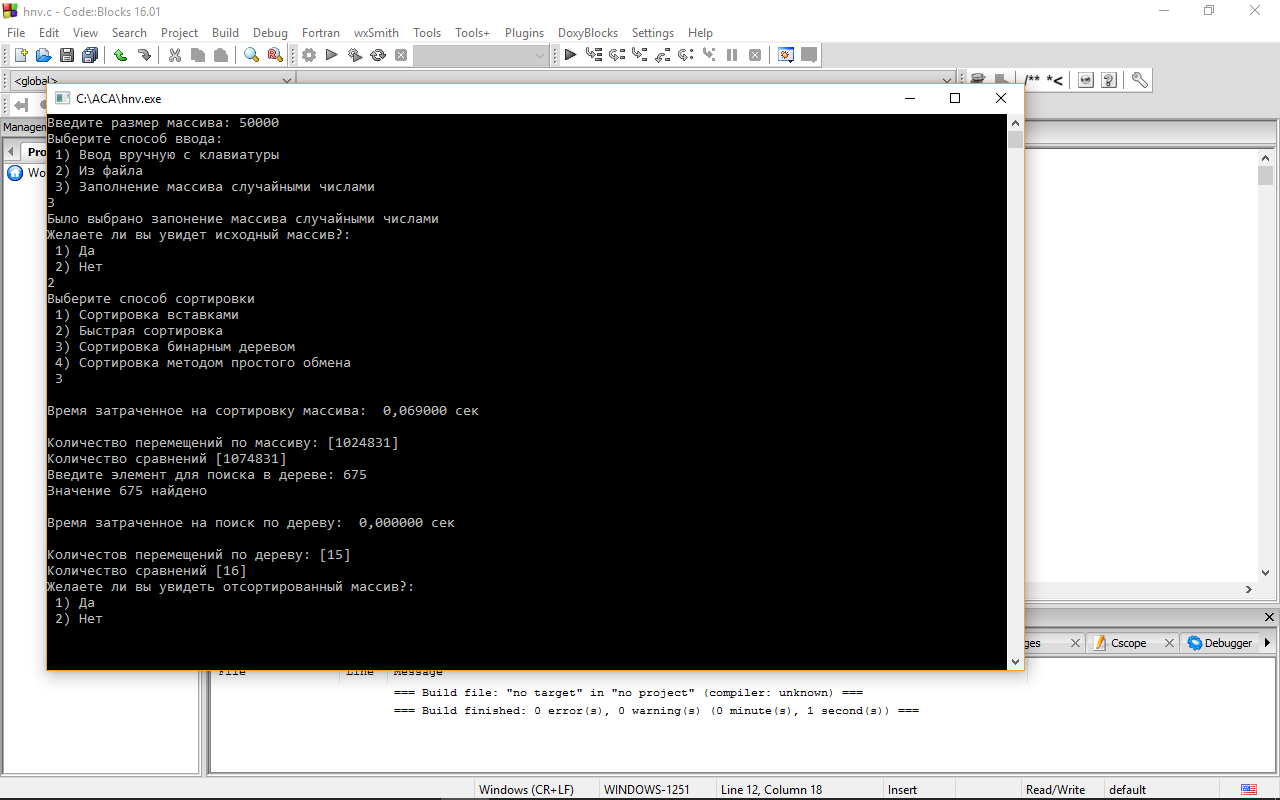
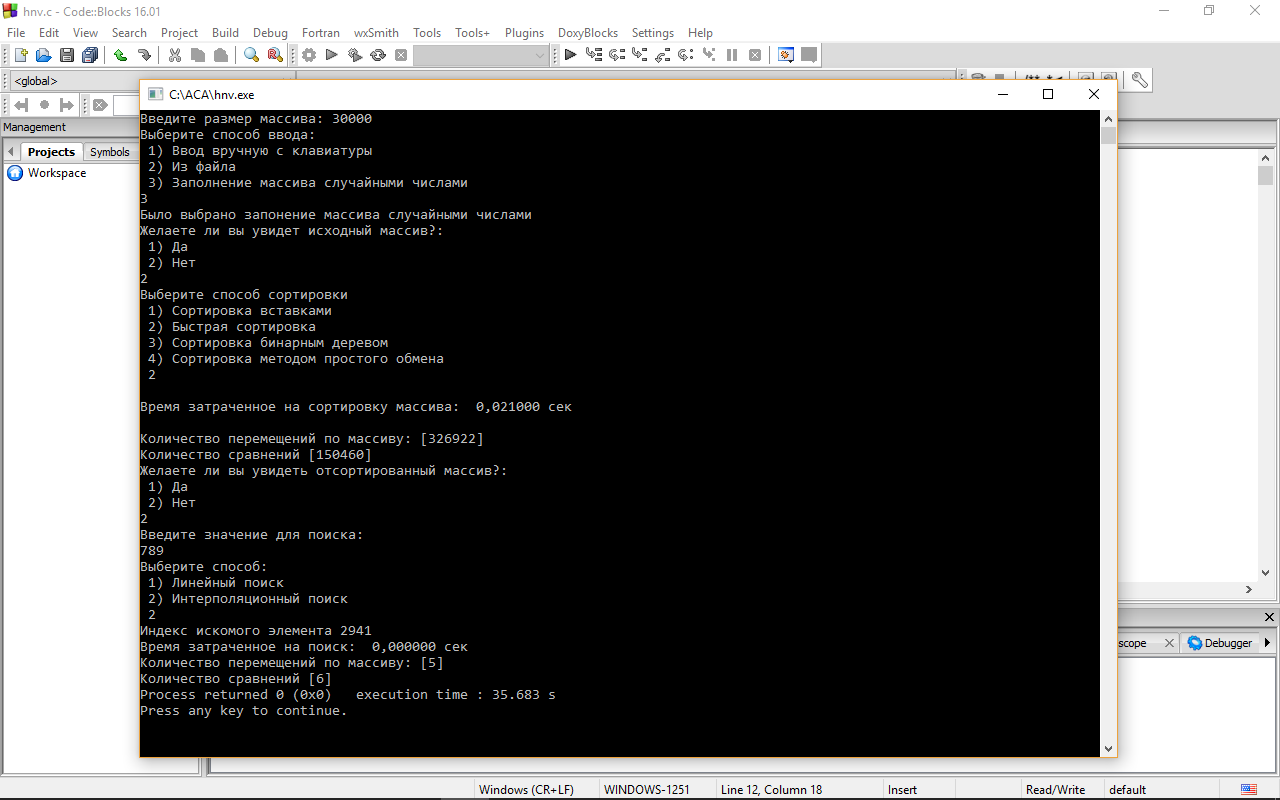
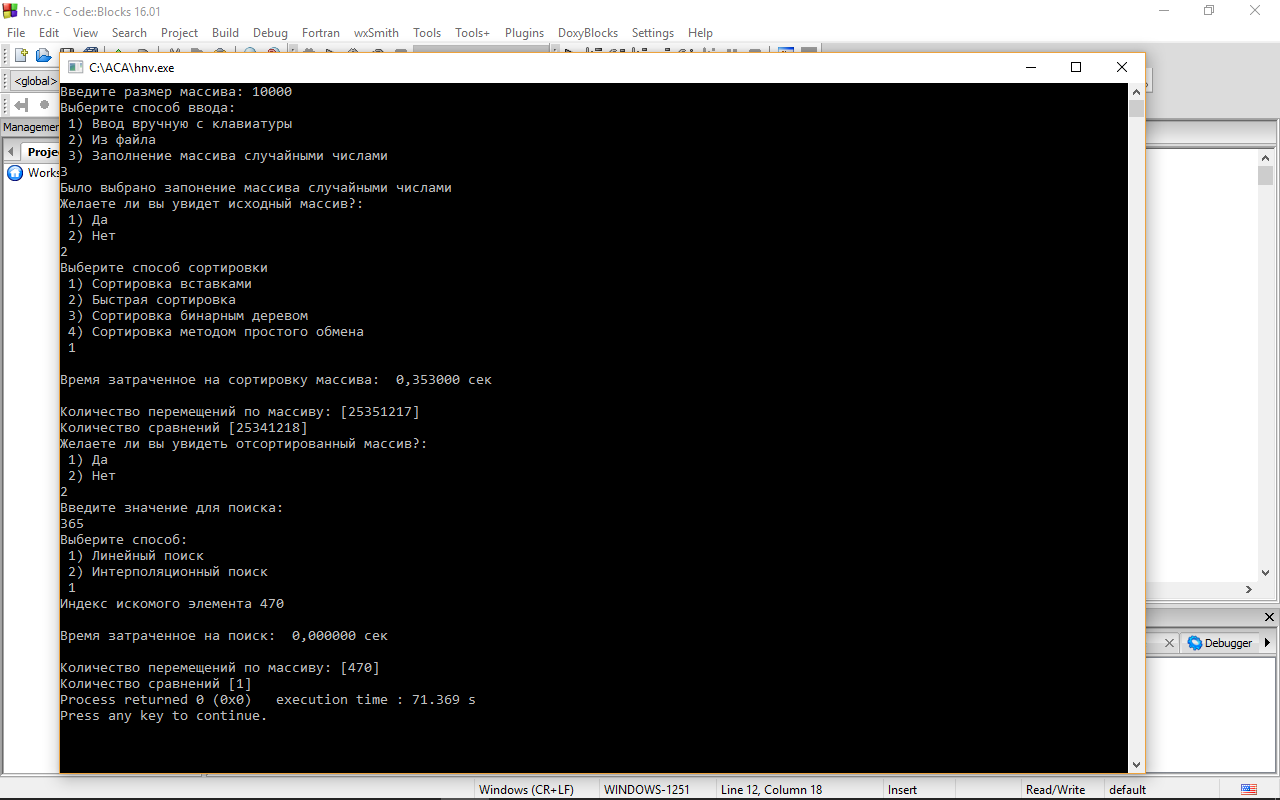
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Сортировка методом простого обмена | Быстрая сортировка Хоара | Сортировка вставками | Сортировка по бинарному дереву |
| N = 10 000 элементов | | | | |
| Время выполнения (сек.) | 1.192 | 0.016 | 0.380 | <0.001 |
| Кол-во сравнений | 25 341 218 | 45 127 | 25 351 127 | 176 056 |
| Кол-во перемещений | 49 995 000 | 100 981 | 24 940 636 | 166 056 |
| N = 30 000 элементов | | | | |
| Время выполнения (сек.) | 11.954 | 0.016 | 3.578 | 0.039 |
| Кол-во сравнений | 225 173 350 | 150 460 | 225 173 350 | 601 813 |
| Кол-во перемещений | 449 985 000 | 326 922 | 225 203 349 | 571 813 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| N = 50 000 элементов | | | | |
| Время выполнения (сек.) | 33.138 | 0.038 | 10.265 | 0.069 |
| Кол-во сравнений | 624 460 895 | 263 415 | 624 460 895 | 1 074 831 |
| Кол-во перемещений | 1 249 975 000 | 566 190 | 624 510 894 | 1 024 831 |
| N = 70 000 элементов | | | | |
| Время выполнения (сек.) | 66.056 | 0.066 | 20.101 | 0.087 |
| Кол-во сравнений | 1 222 322 490 | 380 448 | 1 222 322 490 | 1 588 880 |
| Кол-во перемещений | 2 449 965 000 | 813 918 | 1 222 392 489 | 1 518 880 |
| N = 90 000 элементов | | | | |
| Время выполнения (сек.) | 108.483 | 0.069 | 33.805 | 0.122 |
| Кол-во сравнений | 2 024 746 779 | 500 628 | 2 024 746 779 | 2 144 419 |
| Кол-во перемещений | 4 049 955 000 | 1 066 854 | 2 024 836 778 | 2 054 419 |

## Поиск

|  |  |  |  |
| --- | --- | --- | --- |
|  | Линейный поиск | Интерполяционный поиск | Поиск по бинарному дереву |
| N = 10 000 элементов | | | |
| Время выполнения (сек.) | <0.001 | <0.001 | <0.001 |
| Кол-во сравнений | 1 | 4 | 16 |
| Кол-во перемещений | 470 | 3 | 15 |
| N = 30 000 элементов | | | |
| Время выполнения (сек.) | <0.001 | <0.001 | <0.001 |
| Кол-во сравнений | 1 | 6 | 20 |
| Кол-во перемещений | 2 126 | 5 | 19 |
| N = 50 000 элементов | | | |
| Время выполнения (сек.) | <0.001 | <0.001 | <0.001 |
| Кол-во сравнений | 1 | 4 | 18 |
| Кол-во перемещений | 709 | 3 | 17 |

|  |  |  |  |
| --- | --- | --- | --- |
| N = 70 000 элементов | | | |
| Время выполнения (сек.) | <0.001 | <0.001 | <0.001 |
| Кол-во сравнений | 1 | 6 | 19 |
| Кол-во перемещений | 13 199 | 5 | 18 |
| N = 90 000 элементов | | | |
| Время выполнения (сек.) | <0.001 | <0.001 | <0.001 |
| Кол-во сравнений | 1 | 3 | 12 |
| Кол-во перемещений | 74751 | 2 | 11 |



# **Вывод**

В результате проведения лабораторной работы были получены следующие результаты: самый эффективный метод сортировки массивов - Быстрая сортировка Хоара; самый эффективный метод поиска элементов массивов – поиск по бинарному дереву.

Данные алгоритмы сортировки и поиска показали наилучшие результаты, как по времени работы, так и по количеству сравнений и перемещений в массиве.