УДК 004.021

СРАВНИТЕЛЬНЫЕ ХАРАКТЕРИСТИКИ АЛГОРИТМОВ ПОИСКА И СОРТИРОВОК НА МАССИВАХ

Малюш Д.О., студент гр. 250505

Борисенко К.Н., студент гр. 250501

Белорусский государственный университет информатики и радиоэлектроники

г. Минск, Республика Беларусь

Сидорович А.С. – старший преподаватель

**Аннотация.** В научной работе обсуждается выбор оптимальных алгоритмов поиска и сортировки данных в массивах. Это предполагает сравнительный анализ алгоритмов.

**Ключевые слова.** Алгоритмы сортировки, алгоритмы поиска, сравнительный анализ, достоинства и недостатки, скорость алгоритмов, простота алгоритмов, выбор алгоритма сортировки, выбор алгоритма поиска.

Алгоритм поиска в массиве.

Существует множество способов поиска необходимой позиции в массиве. Далее будут рассмотрены примеры поиска числа в массиве, состоящем из 100 символов.

Способы поиска, на примере которых мы проведём сравнение:  
1. Линейный поиск

2. Индексно-последовательный поиск  
3. Бинарный поиск (или же двоичный поиск)

4. Поиск прыжками

**Линейный поиск.** Линейный поиск — это простой алгоритм поиска, который работает путем последовательного просмотра каждого элемента в массиве и сравнения его с искомым значением. Если элемент найден, алгоритм возвращает его индекс[5].

Линейный поиск имеет сложность O(n), где n — это количество элементов в массиве. Это означает, что в худшем случае алгоритму придется просмотреть все элементы массива, чтобы найти искомое значение.

Одним из преимуществ линейного поиска является его простота и легкость реализации. Он также может быть эффективным для поиска в небольших массивах или в массивах, где искомый элемент находится близко к началу.

Однако для больших массивов или для массивов, где искомый элемент находится близко к концу, линейный поиск может быть неэффективным.

Линейный поиск может быть улучшен путем использования различных техник. Например, можно использовать двунаправленный линейный поиск (также известный как “коктейльный поиск”), который работает путем одновременного просмотра элементов с начала и с конца массива. Это может ускорить поиск в некоторых случаях.

Время работы этой функции составляет O(n), где n - размер массива.

**Индексно-последовательный поиск.** Индексно-последовательный поиск — это алгоритм поиска, который использует индекс для ускорения поиска элемента в списке или массиве. Он работает путем создания индекса, который содержит ключи и указатели на элементы в списке или массиве. Затем алгоритм использует индекс для быстрого нахождения диапазона элементов, которые могут содержать искомое значение. После этого алгоритм выполняет последовательный поиск в этом диапазоне для нахождения искомого элемента. Для использования данного алгоритма массив изначально должен быть отсортирован.

Индекс создается путем выбора ключей из списка или массива и создания структуры данных, которая содержит эти ключи и указатели на соответствующие элементы в списке или массиве. Ключи обычно выбираются таким образом, чтобы они были равномерно распределены по всему списку или массиву. Затем индекс сортируется по ключам, чтобы обеспечить быстрый поиск диапазона элементов, которые могут содержать искомое значение. Создание индекса может занять некоторое время, но после его создания он может быть использован для ускорения поиска.

Одним из преимуществ индексно-последовательного поиска является ускорение поиска элемента в списке или массиве. Использование индекса позволяет алгоритму быстро находить диапазон элементов, которые могут содержать искомое значение, что сокращает время поиска. Кроме того, индексно-последовательный поиск может быть эффективным для поиска в больших объемах данных, так как он позволяет уменьшить количество элементов, которые необходимо проверить.

Одним из недостатков индексно-последовательного поиска является необходимость создания и поддержания индекса. Это может занять дополнительное время и ресурсы, особенно если данные часто изменяются. Кроме того, индекс может занимать дополнительное место в памяти. Еще одним недостатком может быть то, что индексно-последовательный поиск может быть менее эффективным для поиска в небольших объемах данных, так как создание и использование индекса может занять больше времени, чем простой последовательный поиск.

**Бинарный (двоичный) поиск.** Бинарный поиск также известен как логарифмический поиск. Он работает путем деления массива на две части и сравнения искомого значения с элементом в середине[1]. Если искомое значение меньше значения в середине массива, то поиск продолжается в левой половине массива. Если искомое значение больше значения в середине массива, то поиск продолжается в правой половине массива. Этот процесс повторяется до тех пор, пока не будет найден искомый элемент или пока не останется ни одного элемента для поиска.

Время выполнения бинарного поиска логарифмически зависит от размера массива. Это означает, что для массива размером N, бинарный поиск будет выполняться за O(log N) времени. Это делает его эффективным для больших массивов.

Однако бинарный поиск требует, чтобы массив был отсортирован перед началом поиска. Если массив не отсортирован, то бинарный поиск может дать неверные результаты.

Бинарный поиск может быть использован не только для поиска конкретного значения в массиве, но и для решения других задач. Например, он может быть использован для нахождения первого элемента в массиве, который больше или равен заданному значению (lower bound) или последнего элемента в массиве, который меньше или равен заданному значению (upper bound).

Также бинарный поиск может быть использован для решения задач оптимизации. Например, если у нас есть монотонная функция f(x) и мы хотим найти минимальное значение x, при котором f(x) >= k, то мы можем использовать бинарный поиск для нахождения этого значения x.

В общем случае бинарный поиск может быть использован для решения любой задачи, которая удовлетворяет свойству монотонности. Это означает, что если условие выполняется для некоторого значения x, то оно также будет выполняться для всех значений больше x.

Бинарный поиск может быть реализован как рекурсивно, так и итеративно. В рекурсивной реализации функция бинарного поиска вызывает сама себя с новыми границами поиска, пока не будет найден искомый элемент или пока не останется ни одного элемента для поиска. В итеративной реализации используется цикл while для повторения процесса поиска до тех пор, пока не будет найден искомый элемент или пока не останется ни одного элемента для поиска.

Время работы этой функции составляет O(log N), где n - размер массива.

**Поиск прыжками**. Поиск прыжками - это алгоритм поиска, который использует подход “разделяй и властвуй” и требует предварительной сортировки набора данных[2]. [В отличие от двоичного поиска, этот алгоритм движется только вперед](https://proglib.io/p/6-search-algorithms-java).

Алгоритм работает путем прыжков через фиксированное количество элементов в массиве и сравнения каждого прыжка с элементом, который мы должны найти. Если текущий прыжок больше искомого элемента, то поиск продолжается в предыдущем интервале прыжка с помощью линейного поиска.

Например, если необходимо найти элемент x в отсортированном массиве arr размером n с шагом прыжка m, то алгоритм будет выполнять следующие шаги:

1. Следует начать с первого элемента массива arr[0].

2. Далее необходимо "прыгать" через m элементов и сравниваем arr[m] с x.

3. Если arr[m] меньше x, то повторить шаг 2.

4. Если arr[m] больше x, то выполнить линейный поиск между arr[m-m] и arr[m].

5. Если элемент не найден, то необходимо вернуть -1.

Один из способов реализации поиска прыжками - это использование оптимального размера блока. Оптимальный размер блока можно вычислить как m = sqrt(n), где n - размер массива. Это позволяет алгоритму достичь временной сложности O(√n).

Поиск прыжками может быть эффективным в ситуациях, когда двоичный поиск слишком медленный из-за большого количества операций сравнения. Он также может быть полезен в ситуациях, когда линейный поиск слишком медленный из-за большого размера массива.

Однако поиск прыжками не всегда является лучшим выбором. Например, если элементы массива распределены неравномерно, то поиск прыжками может быть менее эффективным, чем другие алгоритмы поиска.

Сравнение скорости работы перечисленных алгоритмов поиска представлено на [рисунке 1](#bookmark)

Сравнение памяти, необходимой для работы перечисленных алгоритмов поиска представлено на [рисунке 2](#bookmark1).

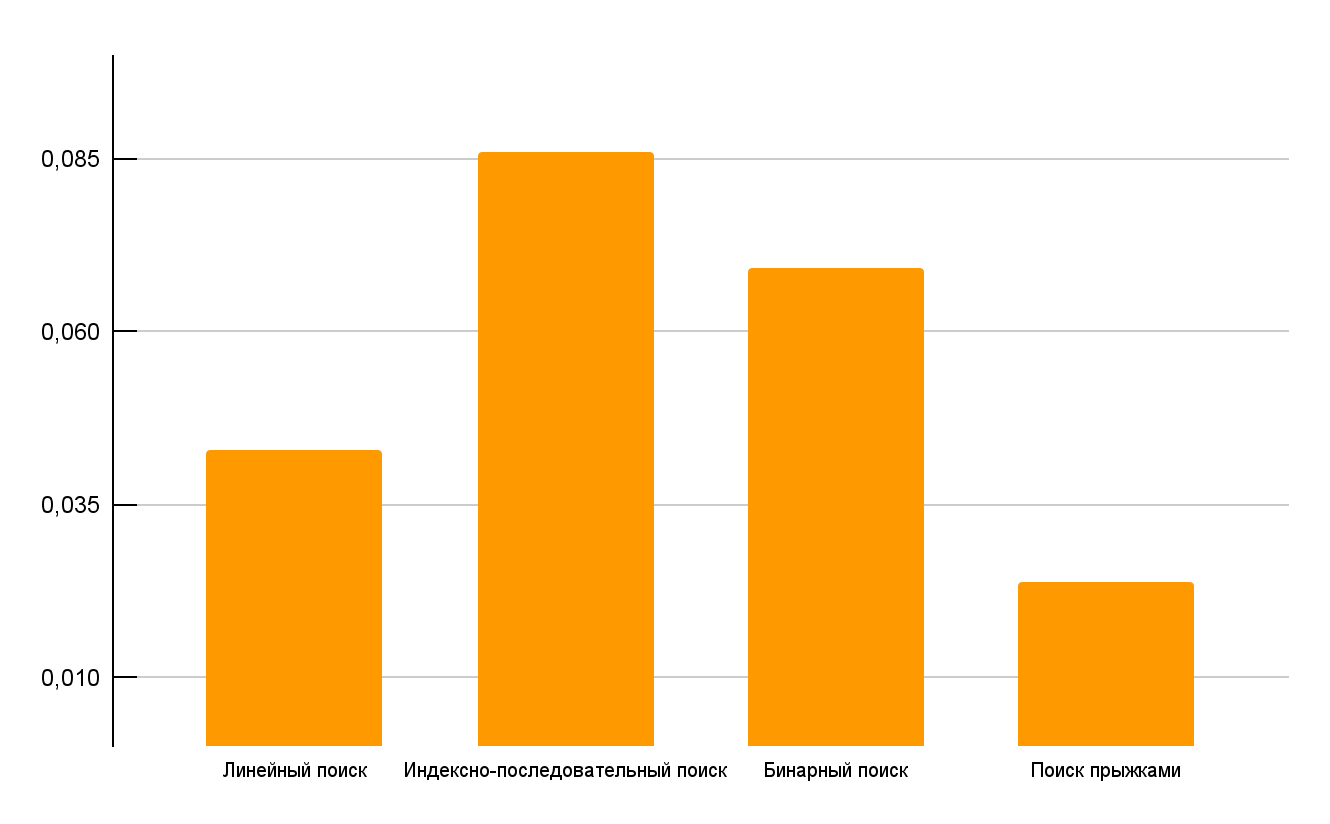
**

Рисунок 1 – Диаграмма, отображающая различие в скорости работы рассматриваемых алгоритмов поиска

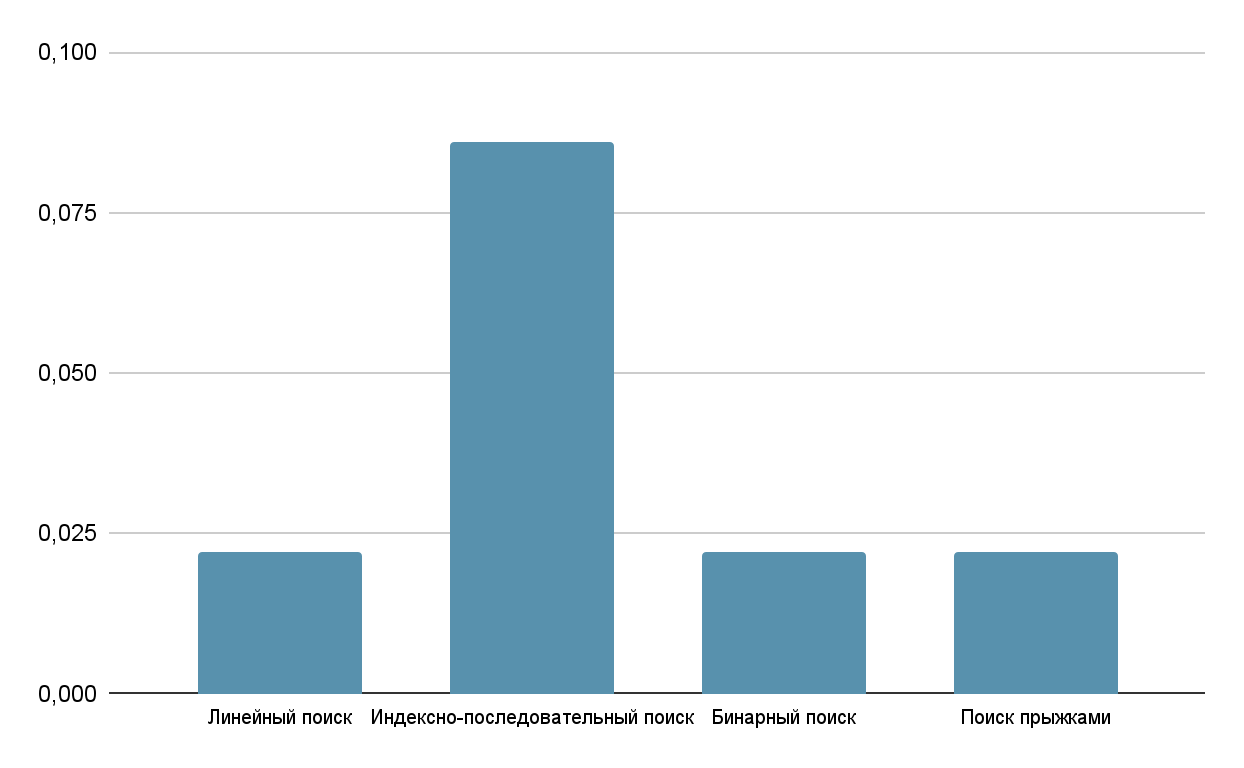


Рисунок 2 – Диаграмма, отображающая различие в количестве памяти, требуемой для работы рассматриваемых алгоритмов поиска

Для сравнения алгоритмов поиска использовался массив на 100 символов.

**Выводы по поиску в массиве.** Самым простым алгоритмом поиска нужного элемента в массиве является линейный поиск, он же занимает меньше всего строк кода. Бинарный поиск и поиск прыжками - это два довольно хороших алгоритмов поиска. Выбор между ними зависит от конкретной ситуации. Бинарный поиск работает лучше на отсортированных массивах с произвольным доступом, таких как массивы и списки. Поиск прыжками может быть более эффективным на больших массивах с ограниченным доступом, таких как связные списки. Стоит отметить, что три исследуемых алгоритма используют одинаковое значение памяти, поскольку не создают дополнительных массивов, в отличии от Индексно-последовательного алгоритма поиска. В нашем случае самым эффективным по всем параметрам оказался именно алгоритм поиска прыжками.

**Сортировка массива.**

Способов сортировки даже больше, чем способов поиска значения в массиве. Далее мы перечислим их и рассмотрим примеры сортировки массива.

Способы поиска, о которых будет идти речь:  
1. Сортировка пузырьком  
2. Быстрая сортировка (алгоритм Хоара)

3. Сортировка методом Шелла

4. Сортировка вставками

Сравнение скорости работы перечисленных алгоритмов сортировки представлено на [рисунке 3](#bookmark2)

Сравнение памяти, необходимой для работы перечисленных алгоритмов сортировки представлено на [рисунке 4](#bookmark3)

**Сортировка пузырьком.** Сортировка пузырьком - это простой алгоритм сортировки. Он работает путем многократного сравнения и обмена соседних элементов, если они находятся в неправильном порядке[3]. Алгоритм состоит из повторяющихся проходов по сортируемому массиву. За каждый проход элементы последовательно сравниваются попарно и, если порядок в паре неверный, выполняется перестановка 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).

Сложность алгоритма: O(n^2). Алгоритм считается учебным и практически не применяется вне учебной литературы, вместо него на практике применяются более эффективные алгоритмы сортировки.

Время работы этой функции составляет O(n^2), где n - размер массива.

**Быстрая сортировка (сортировка Хоара):** Быстрая сортировка — это алгоритм сортировки, разработанный английским информатиком Тони Хоаром в 1960 году во время его работы в МГУ. Он относится к алгоритмам «разделяй и властвуй» и состоит из трех шагов: выбор опорного элемента из массива, разбиение массива таким образом, чтобы элементы меньше опорного помещались перед ним, а большие или равные - после, и рекурсивное повторение этих действий для левой и правой части массива[3].

В худшем случае время работы алгоритма составляет O(n^2), но в среднем - O(n log n). Эффективность алгоритма может сильно зависеть от выбора опорного элемента.

Одно из преимуществ быстрой сортировки - ее скорость. Она является одним из самых быстрых известных универсальных алгоритмов сортировки массивов. Однако у нее есть и недостатки. Например, она не является устойчивой сортировкой, то есть она может изменять относительный порядок равных элементов.

Быстрая сортировка может быть применена для сортировки массивов различных типов данных. В языке Си можно использовать функцию qsort, которая описана в стандартной библиотеке ANSI C и объявлена в заголовочном файле stdlib.h.

Время работы этой функции изменяется от O(n log n) - самый частый результат, до O(n^2) - в худшем случае, где n - размер массива.

**Сортировка методом вставок.** Сортировка вставками - это алгоритм сортировки, который работает путем перемещения текущего элемента входной последовательности на нужную позицию в уже отсортированном массиве. Отсортированный массив строится по одному элементу за раз[4]. Сортировка вставками имеет большую вычислительную сложность, эффективна на небольших наборах данных. В реализации сортировки вставками на языке СИ обычно используется функция, которая принимает два аргумента: число элементов в массиве и сам массив. Внутри функции используется цикл, который начинается с первого (не нулевого) значения массива и продолжается до конца массива. На каждой итерации цикла выбирается новый элемент из массива и сравнивается с элементами в уже отсортированном списке. Если порядок между сравниваемыми элементами нарушен, то меняем их местами. Этот процесс продолжается до тех пор, пока не будет найдено подходящее место для нового элемента.

Сортировка вставками имеет сложность O(n^2) в худшем случае, когда входные данные отсортированы в обратном порядке. В лучшем случае, когда данные уже отсортированы, сложность алгоритма составляет O(n). Сортировка вставками эффективна на небольших наборах данных или когда данные уже частично отсортированы.

Время работы этой функции изменяется от O(n) - в лучшем случае, до O(n^2) - в худшем случае.

**Сортировка методом Шелла.** Сортировка Шелла — алгоритм сортировки, являющийся усовершенствованным вариантом сортировки вставками. [Идея метода Шелла состоит в сравнении элементов, стоящих не только рядом, но и на определённом расстоянии друг от друга](https://ru.wikipedia.org/wiki/%2525D0%2525A1%2525D0%2525BE%2525D1%252580%2525D1%252582%2525D0%2525B8%2525D1%252580%2525D0%2525BE%2525D0%2525B2%2525D0%2525BA%2525D0%2525B0_%2525D0%2525A8%2525D0%2525B5%2525D0%2525BB%2525D0%2525BB%2525D0%2525B0)[3].

При сортировке Шелла сначала сравниваются и сортируются между собой значения, стоящие один от другого на некотором расстоянии. [После этого процедура повторяется для некоторых меньших значений, а завершается сортировка Шелла упорядочиванием элементов при (то есть обычной сортировкой вставками)](https://ru.wikipedia.org/wiki/%2525D0%2525A1%2525D0%2525BE%2525D1%252580%2525D1%252582%2525D0%2525B8%2525D1%252580%2525D0%2525BE%2525D0%2525B2%2525D0%2525BA%2525D0%2525B0_%2525D0%2525A8%2525D0%2525B5%2525D0%2525BB%2525D0%2525BB%2525D0%2525B0).

[Эффективность сортировки Шелла в определённых случаях обеспечивается тем, что элементы «быстрее» встают на свои места (в простых методах сортировки, например, пузырьковой, каждая перестановка двух элементов уменьшает количество инверсий в списке максимум на 1, а при сортировке Шелла это число может быть больше)](https://ru.wikipedia.org/wiki/%2525D0%2525A1%2525D0%2525BE%2525D1%252580%2525D1%252582%2525D0%2525B8%2525D1%252580%2525D0%2525BE%2525D0%2525B2%2525D0%2525BA%2525D0%2525B0_%2525D0%2525A8%2525D0%2525B5%2525D0%2525BB%2525D0%2525BB%2525D0%2525B0).

[Невзирая на то, что сортировка Шелла во многих случаях медленнее, чем быстрая сортировка, она имеет ряд преимуществ: отсутствие потребности в памяти под стек; отсутствие деградации при неудачных наборах данных — быстрая сортировка легко деградирует до O(n²), что хуже, чем худшее гарантированное время для сортировки Шелла](https://ru.wikipedia.org/wiki/%2525D0%2525A1%2525D0%2525BE%2525D1%252580%2525D1%252582%2525D0%2525B8%2525D1%252580%2525D0%2525BE%2525D0%2525B2%2525D0%2525BA%2525D0%2525B0_%2525D0%2525A8%2525D0%2525B5%2525D0%2525BB%2525D0%2525BB%2525D0%2525B0).

Время работы этой функции изменяется от O(n log^2 n) - в лучшем случае, до O(n^2) - в худшем случае.

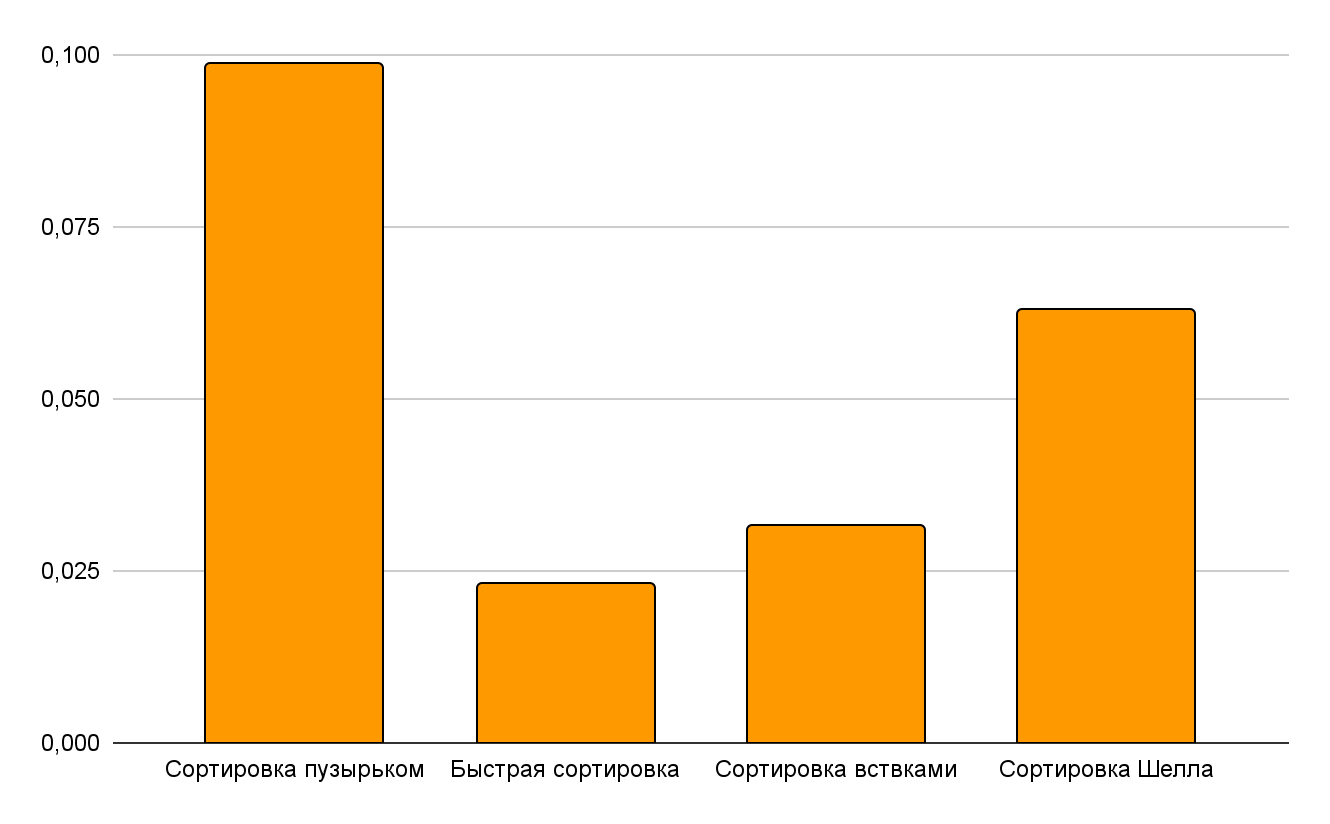


Рисунок 3 – Диаграмма, отображающая различие в скорости работы рассматриваемых алгоритмов сортировок

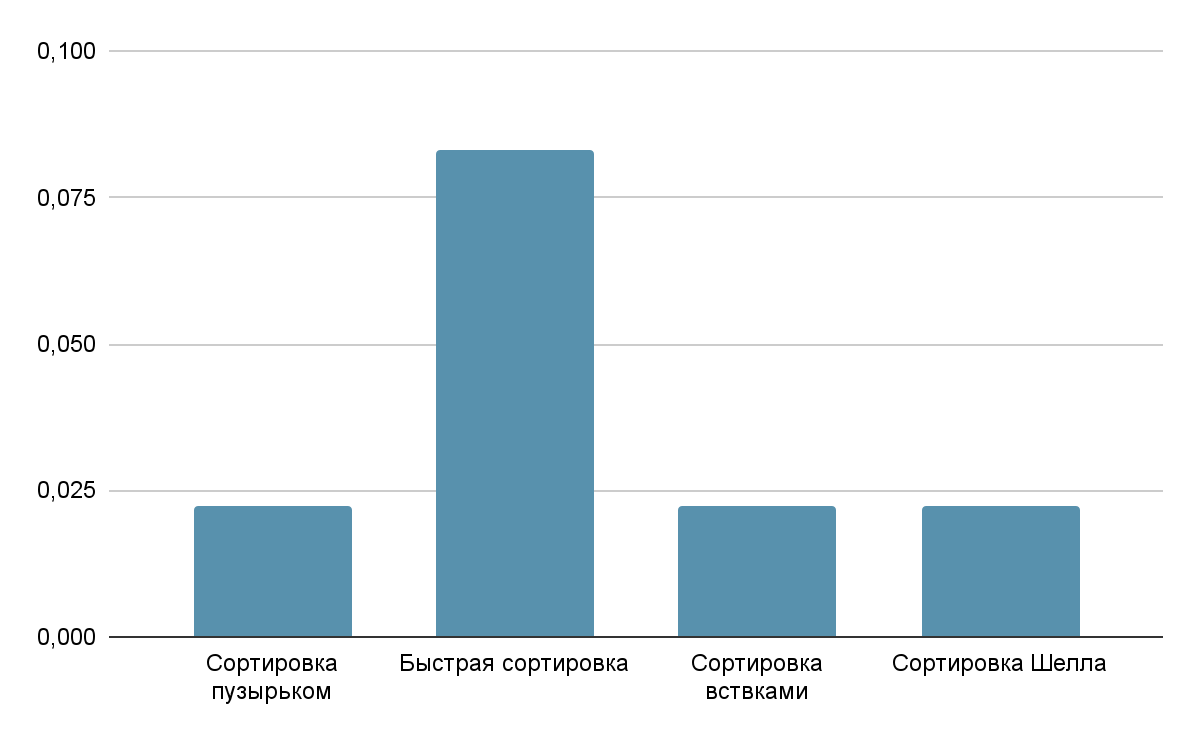


Рисунок 4 – Диаграмма, отображающая различие в количестве памяти, требуемой для работы рассматриваемых алгоритмов сортировки

Для сравнения алгоритмов поиска использовался массив на 100 символов.

**Выводы по сортировкам массива.** Каждый способ сортировки будет уместен в той или иной ситуации. Например, для максимально быстрой сортировки лучше использовать сортировку Хоара (быструю сортировку), ведь этот алгоритм имеет наибольшую скорость. Для сортировки небольших массивов можно использовать сортировку вставками или же сортировку методом Шелла. Ну а в случае, если массив настолько мал, что, независимо от метода сортировки, время выполнения остается почти одинаковым - можно даже использовать сортировку пузырьком, ведь она пишется проще всего. Также стоит отметить, что, как и в случае с алгоритмами поиска, три рассматриваемых алгоритма требуют равное количество памяти, однако алгоритм быстрой сортировки использует рекурсию, из-за чего количество требуемой памяти увеличивается.

**Список использованных источников:**

1. [Левитин А. В.](https://www.wikidata.org/wiki/Q21694518) Глава 4. Метод декомпозиции: Бинарный поиск // [Алгоритмы. Введение в разработку и анализ](https://www.wikidata.org/wiki/Q21694522) — М.: [Вильямс](https://www.wikidata.org/wiki/Q21694521), 2006. — С. 180—183. — 576 с.

2. Скиена С. Алгоритмы. Руководство по разработке. 2-е изд.: Пер. с англ. — СПб.: БХВ-Петербург. 2011. — 720 с.: ил.

3. Липачѐв Е. К. // Технология программирования. Методы сортировки данных // учебное пособие / Е.К. Липачѐв. – Казань: Казан. ун-т, 2017. – 58 с.

4. [Кнут Д. Э.](https://ru.wikipedia.org/wiki/%2525D0%25259A%2525D0%2525BD%2525D1%252583%2525D1%252582,_%2525D0%252594%2525D0%2525BE%2525D0%2525BD%2525D0%2525B0%2525D0%2525BB%2525D1%25258C%2525D0%2525B4_%2525D0%2525AD%2525D1%252580%2525D0%2525B2%2525D0%2525B8%2525D0%2525BD) 5.2 Внутренняя сортировка // [Искусство программирования. Том 3. Сортировка и поиск](https://books.google.ru/books?id=92rW-nktlbgC&printsec=frontcover&dq=editions:spjjKVwoQ3QC&hl=ru&sa=X&ved=0ahUKEwiUjLbc88rQAhVCfiwKHUJlBQoQuwUIIjAB#v=onepage&q&f=false) = The Art of Computer Programming. Volume 3. Sorting and Searching / под ред. В. Т. Тертышного (гл. 5) и И. В. Красикова (гл. 6). — 2-е изд. — Москва: Вильямс, 2007. — Т. 3. — 832 с..

5. [Кнут Д](https://ru.wikipedia.org/wiki/%2525D0%252594%2525D0%2525BE%2525D0%2525BD%2525D0%2525B0%2525D0%2525BB%2525D1%25258C%2525D0%2525B4_%2525D0%25259A%2525D0%2525BD%2525D1%252583%2525D1%252582) . Искусство программирования, том 3. Сортировка и поиск = The Art of Computer Programming, vol.3. Sorting and Searching. — 2-е изд. — М.: [«Вильямс»](https://ru.wikipedia.org/w/index.php?title=%2525D0%252592%2525D0%2525B8%2525D0%2525BB%2525D1%25258C%2525D1%25258F%2525D0%2525BC%2525D1%252581_(%2525D0%2525B8%2525D0%2525B7%2525D0%2525B4%2525D0%2525B0%2525D1%252582%2525D0%2525B5%2525D0%2525BB%2525D1%25258C%2525D1%252581%2525D1%252582%2525D0%2525B2%2525D0%2525BE)&action=edit&redlink=1), 2007. — С. 824

UDC 004.021

COMPARATIVE CHARACTERISTICS OF SEARCH AND SORTING ALGORITHMS ON ARRAYS

Malyush D.O., Borisenko K.N.

Belarusian State University of Informatics and Radioelectronics, Minsk, Republic of Belarus

Sidorovich A.S. - Senior Lecturer

**Annotation.** The scientific work discusses the choice of optimal algorithms for searching and sorting data in arrays. This involves a comparative analysis of algorithms.

**Keywords.** Sorting algorithms, search algorithms, comparative analysis, advantages and disadvantages, speed of algorithms, simplicity of algorithms, choice of sorting algorithm, choice of search algorithm.