УДК 004.021

# ПРОГРАММНОЕ СРЕДСТВО ПОШАГОВОГО АНАЛИЗА АЛГОРИТМОВ ПОИСКА И СОРТИРОВКИ ДАННЫХ

Е.В. ДОВГУЛЕВИЧ, В.С. ШУКА, О.Г. ШЕВЧУК

*Белорусский государственный университет информатики и радиоэлектроники, Республика Беларусь*

*Поступила в редакцию 18 марта 2024*

**Аннотация.** Разаработанно программное средство пошагового анализа алгоритмов поиска и сортировки данных. Описаны основные возможности и принцип работы программного средства. Приведено описание используемых алгоритмов поиска и сортировки. Рассмотрены методы оценки вычислительной сложности алгоритмов.

*Ключевые слова:* алгоритмы поиска, алгоритмы сортировки, вычислительная сложность алгоритмов.

## Введение

При разработке различных программных средств инфокоммуникаций (ПСИ) фудаментальное значение имеет выбор и обоснование алгоритмов обработки и структур данных. Существует различные типы алгоритмов обработки данных: поиска и сортировки, динамического программирования, математические, побитовые, поиска шаблонов, жадные алгоритмы, алгоритмы машинного обучения и т.д, которые могут быть использованны при реализации эффективных решений различных проблем.

Сортировка и поиск являются основными этапами обработки данных в инфокоммуникационных системах. Выбор подходящих алгоритмов поиска и сортировки данных позволяет в процессе разработки ПСИ наиболее эффективно распределить имеющиеся физические и виртальные ресурсы (внутренняя память, тактовая частота и разрядность процессора, и др.). Для оценки алгоритмов поиска и сортировки используются временная и пространствеена сложность, однако для лучшего их понимания и изучения также необходимо использовать визуальное представление работы алгоритмов, что позволит быстро и наглядно оценить не только сложность, но и более досканально понять основные принципы их работы.

Цель работы – разработка программного средства пошагового анализа основных алгоритмов поиска и сортировки данных.

## Алгоритмы поиска и сортировки данных

Основными классами алгоритмов, представляющих собой фундаментальные подходы к решению задач поиска и сортировки, являются:

1. Линейный поиск [1]. Является одним из самых простых и понятных алгоритмов поиска. Работает путем последовательного изучения каждого элемента в коллекции данных (массива или списка) до тех пор, пока не будет найдено совпадение или пока не будет пройдена вся коллекция.

2. Бинарный поиск [1]. Используется в отсортированном массиве путем многократного деления интервала поиска пополам.

3. Хеширование [1]. Алгоритмы хеширования преобразуют входные данные в уникальные значения, обеспечивая быстрый доступ к ним.

4. Сортировка пузырьком [2]. Метод сортировки массивов и списков путем последовательного сравнения соседних элементов и их обмена, если предшествующий оказывается больше последующего (при сортировке по возрастанию), как показано на рис. 1, *а*.

5. Сортировка слиянием [3]. Рекурсивный алгоритм, при котором сортируемые элементы разбиваются на две группы, каждая из этих меньших групп сортируется рекурсивно, после чего оба отсортированных списка сливаются воедино и их элементы чередуются, образуя полностью отсортированный общий список. Принцип работы алгоритма приведен на рис. 1, *б*.

6. Быстрая сортировка [4]. Выбирает элемент в качестве опорного элемента и разбивает заданный массив вокруг выбранного опорного элемента, помещая опорный элемент в правильное положение в отсортированном массиве, что показано на рис. 1, *в*.

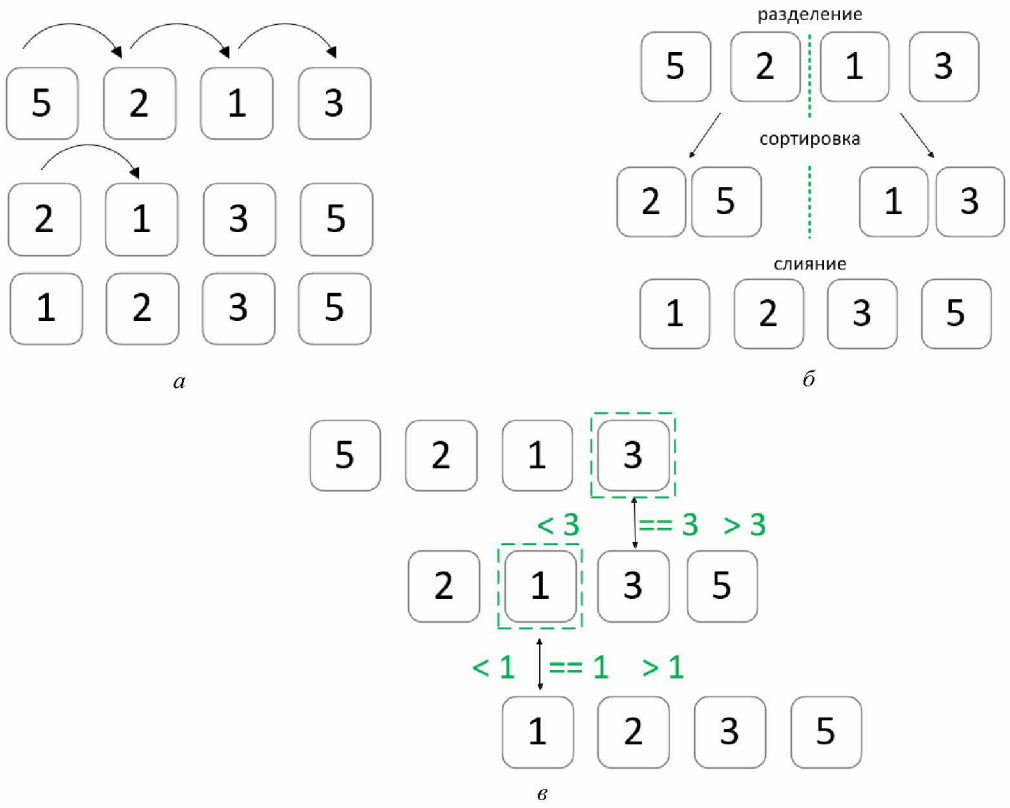



Рис. 1. Принцип сортировки элементов в порядке возрастания следующими алгоритмами: *а* – пузырьковая сортировка, *б* – сортировка слиянием, *в* – быстрая сортировка (опорным элементом выбран последний элемент массива)

## Оценка сложности алгоритмов поиска и сортировки данных

Выбор подходящего алгоритма для конкретных требований должен быть основан на понимании того, как быстро выполняется алгоритм. Сложность алгоритмов описывает степень трудности или затраты ресурсов при выполнении алгоритма. Анализ сложности помогает оптимизировать производительность программы, экономить ресурсы и создавать качественные приложения. На практике существуют два вида оценки сложности алгоритмов:

1. Временная сложность. Позволяет оценить время, необходимое для выполнения алгоритма в зависимости от размера входных данных. Для анализа используются такие методы, как оценка в худшем случае (worst-case), среднем случае (average-case) и лучшем случае (best-case). Наиболее распространенным является оценка в худшем случае, так как она дает наиболее пессимистичные результаты. Оценка временной сложности обычно выражается в виде «О-большое» нотации [5].

2. Пространственная сложность. Оценивает объем памяти, необходимый для его выполнения и включает в себя анализ структур данных, массивов, списков и других объектов, которые используются в алгоритме, что позволяет определить, как алгоритм будет влиять на объем занимаемой им памяти при различных входных данных.

Оптимизация пространственной сложности включает в себя поиск способов уменьшения потребления памяти алгоритмом без ущерба его функциональности, может включать в себя

выбор более эффективных структур данных или использование алгоритмов, которые требуют меньшего объема памяти [6].

Распространенные варианты сложности алгоритмов и их краткое описание приведены в табл. 1.

Табл. 1. **Варианты сложности алгоритмов**

| Сложность | Описание | Пример |
|---|---|---|
| $O(1)$ | Константная сложность. Время выполнения алгоритма остается постоянным и не зависит от объема данных. | Получение элемента массива по индексу |
| $O(\log n)$ | Логарифмическая сложность. Время выполнения алгоритма увеличивается логарифмически с увеличением размера входных данных $(n)$. | Бинарный поиск |
| $O(n)$ | Линейная сложность. Оценка временной сложности $O(n)$ означает, что время выполнения алгоритма растет линейно с увеличением размера входных данных. | Поиск элемента в неотсортированном массиве |
| $O(n\log n)$ | Линейно-логарифмическая сложность | Быстрая сортировка |
| $O(n^2)$ | Квадратичная | Сортировка выбором |
| $O(n^3)$ | Кубическая | Перемножение матриц |
| $O(C^n)$ | Экспоненциальная | Встречается в алгоритмах, которые решают проблемы методом "разделяй и властвуй" или используют рекурсию без оптимизации. |
| $O(n!)$ | Факториальная | Комбинаторные алгоритмы |

Основными факторами, влияющими на сложность алгоритмов, являются:

1. Входные данные.

1.1. Размер и формат входных данных оказывают значительное влияние на сложность алгоритмов. Алгоритмы могут иметь различную производительность в зависимости от объема данных, с которыми им приходится работать. Например, алгоритмы, имеющие линейную сложность, могут столкнуться с затруднениями при обработке больших объемов данных, в то время как алгоритмы с логарифмической сложностью могут быть более оптимальными в таких случаях.

1.2. Степень упорядоченности данных также влияет на сложность алгоритмов. Например, для упорядоченных данных алгоритмы сортировки могут работать более эффективно, в то время как для неупорядоченных данных могут потребоваться более сложные алгоритмы.

2. Структура данных. Выбор подходящих структур данных является ключевым фактором в проектировании алгоритмов. Различные структуры данных, такие как массивы, списки, деревья и хэш-таблицы, могут использоваться для различных задач. Например, для поиска элемента в большом массиве продуктивным может быть использование бинарного поиска. Эффективное использование структур данных также связано с оптимальным расходованием памяти и времени.

3. Операции и операторы. Выбор операций включает в себя использование подходящих математических и логических операторов для выполнения необходимых действий. Оптимизация алгоритмов включает в себя исключение избыточных шагов для достижения лучшей производительности.

Предположим, необходимо решить задачу по поиску наименьшего элемента в неупорядоченном массиве. Рассмотрим два алгоритма для ее решения: линейный поиск и бинарный поиск:

1. Линейный поиск:

1.1. Временная сложность: $O(n)$, где $n$ – размер массива.

1.2. Пространственная сложность: $O(1)$, так как алгоритм не использует дополнительную память.

2. Бинарный поиск:

2.1. Временная сложность: $O(\log n)$, при условии, что массив отсортирован.

2.2. Пространственная сложность: $O(1)$, так как алгоритм также не требует дополнительной памяти.

В данном случае, если массив уже отсортирован, бинарный поиск будет более подходящим, особенно при больших объемах данных. Однако, если сортировка массива занимает слишком много времени, линейный поиск может быть предпочтительным выбором. Такой подход к выбору алгоритма помогает учесть различные аспекты задачи и оптимизировать ее решение с учетом требований к производительности и ресурсам.

## Программное средство оценки алгоритмов поиска и сортировки

Разработанно программное средство (AlgoExplorer) пошагового визуального анализа основных алгоритмов поиска и сортировки данных. Диаграмма вариантов использования программного средства AlgoExplorer представлена на рис. 2.

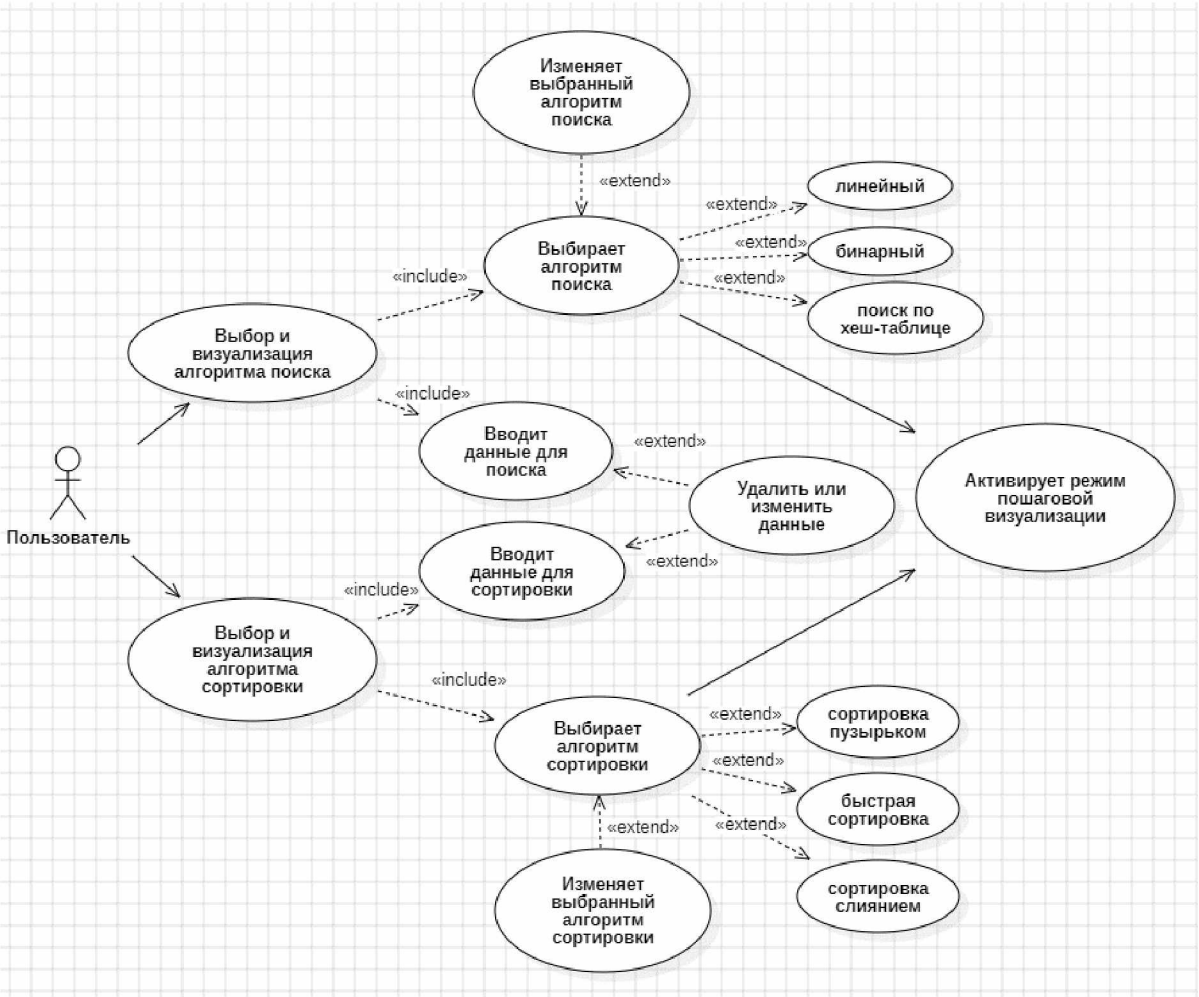



Рис. 2. Диаграмма вариантов использования программного средства AlgoExplorer

Из рис. 2 видно, что AlgoExplorer предоставляет возможность не только наблюдать за выполнением алгоритмов сортировки и поиска, но и взаимодействовать с ними. Пользователь может изменять входные данные и наблюдать, как алгоритмы реагируют на эти изменения. На рис. 3 приведена диаграмма взаимодействия основных компонентов программного средства для процесса выбора и визуализации алгоритма поиска.

Программное средство AlgoExplorer реализованно на языке программирования Python 3.11, с использованием библиотек Flet, что позволило обеспечить интуитивно понятный и легко управляемый интерфейс, доступный на различных операционных системах, что делает визуализацию алгоритмов доступной для широкого круга пользователей.

На рис. 4 отражены основные функциональные компоненты графического интерфейса.

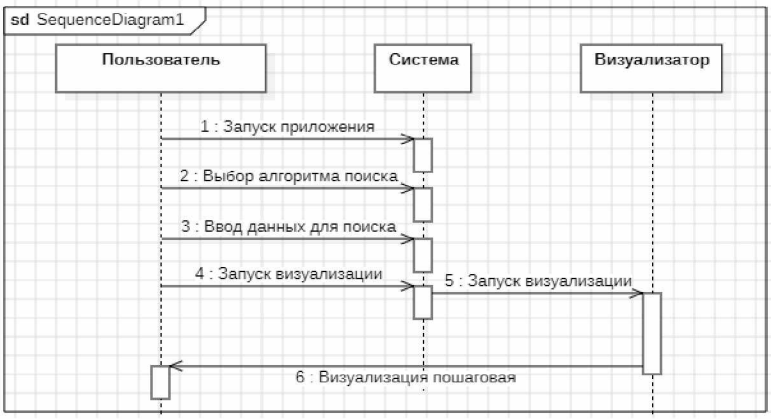



Рис. 3. Диаграмма последовательности для визуализации алгоритмов


Рис. 4. Графический пользовательский интерфейс программного средства AlgoExplorer

На рис. 4 видно, что основными элементами управления программного средства являются:

1. Выпадающий список, обозначенный буквой *A*.

Позволяет пользователю выбрать алгоритм сортировки: пузырьковая и быстрая сортировки, сортировки вставками, выбором, кучей и слиянием, а также сортировка Шелла.

2. Строка для ввода, обозначенная буквой *B*.

Служит для пользовательского ввода массива целых чисел. Поддерживается как простое перечисление элементов массива через знак пробела или запятую, так и запись интервалов через знак "-". Например, ввод "1 2 3 4 5" эквивалентен строке "1-5", а ввод "1, 2, 3, 4, 5, 10" – строке "1-5, 10". Присутствует также защита от нечислового ввода и некорректных интервальных значений.

3. Выпадающий список, обозначенный буквой *C*.

Необходим для выбора способа реорганизации элементов массива. Введенный массив можно оставить без изменений, перемешать его элементы случайным образом, вывести элементы в обратном порядке или отсортировать таким образом, чтобы каждый элемент случайным образом сменил свое положение в массиве, но не более чем на две позиции, что удобно для приведения заранее упорядоченных последовательностей в состояние близкое к отсортированному.

4. Кнопка "Shuffle Array", обозначенная буквой *D*.

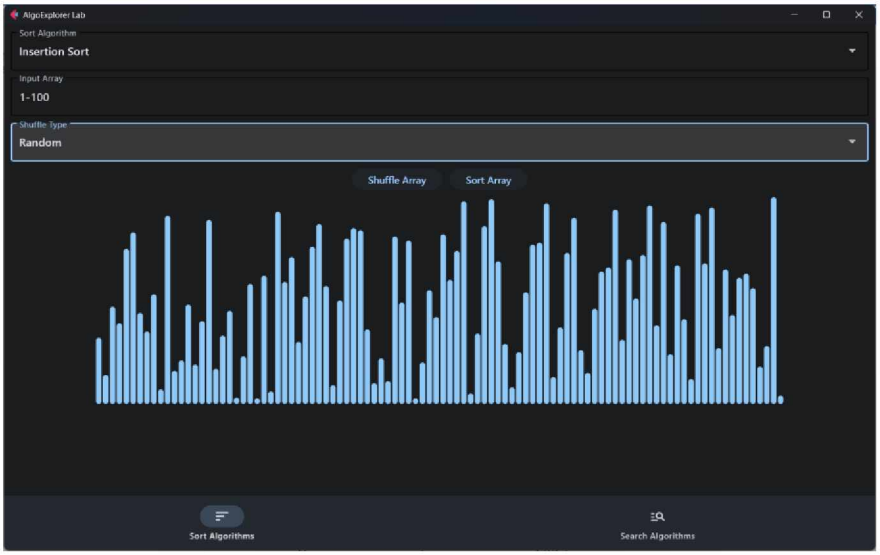
Применяет к введенной в строку *B* последовательности выбранный в списке *C* способ реорганизации элементов и выводит полученный результат на экран в виде столбчатой диаграммы.

5. Кнопка "Sort Array", обозначенная буквой *E*.

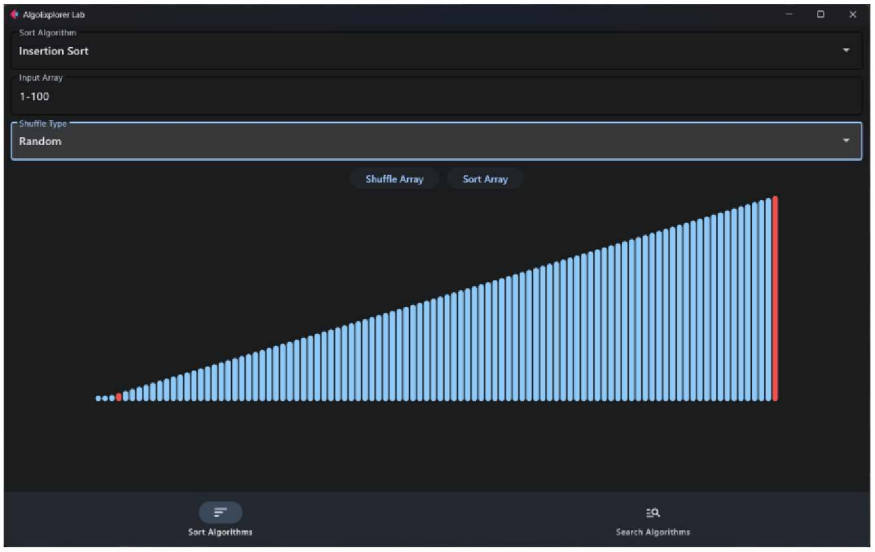
Запускает пошаговую сортировку для введенного пользователем массива чисел.

6. Кнопки *F* и *G*, позволяют пользователю переключаться между режимами анализа алгоритмов сортировки и поиска.

Пример визуализации входных данных и результата работы алгоритма сортировки приведен на рис. 5. В ходе работы разработанное программное средство демонстрирует каждый шаг работы выбранного алгоритма. Для этого текущий обрабатываемый элемент массива, ключевой элемент, подсвечивается красным цветом. В зависимости от алгоритма ключевыми считаются один или сразу несколько столбцов. Наведение мыши на любой из столбцов диаграммы покажет его численное значение. До окончания процесса сортировки кнопки *D* и *E* блокируются. Для обеспечения наглядности для каждой сортировки было подобрано значение задержки между итерациями, таким образом программа не отражает реальной длительности выполнения того или иного алгоритма, однако позволяет оценить количество операций.




*а*




*б*

Рис. 6. Пример работы сортировки вставками для пользовательского ввода "1-100":
*а* – элементы массива перемешаны случайным образом, *б* – массив был упорядочен с помощью сортировки вставками

В режиме работы с алгоритмами поиска к описанным выше полям добавляется строка для ввода целевого значения, позицию которого необходимо обнаружить в заданном пользователем массиве. На выбор предлагаются алгоритмы линейного и бинарного поиска, а также хеширование. В ходе поиска на диаграмме также подсвечиваются ключевые столбцы.

## Заключение

Выбор алгоритмов поиска и сортировки данных является важной задачей при разработке ПО. Понимание, как алгоритм будет вести себя при различных условиях, помогает выбирать наилучшие подходы для решения конкретных задач и минимизировать использование ресурсов, что особенно важно в контексте разнообразных платформ, от устройств с ограниченными ресурсами до серверных систем с высокой производительностью.

Анализ сложности в разработке программного обеспечения играет ключевую роль, поскольку он позволяет создавать эффективные и оптимизированные алгоритмы. Однако лучшее понимание принципов работы алгоритмов поиска и сортировки дает их визуальное представление, что также позволяет оценить их зависимость от входных данных.

Разаработанное программное средство пошагового анализа алгоритмов поиска и сортировки данных визуализирует не только полный цикл работы выбранного алгоритма, но и позволяет детально рассмотреть каждый его шаг. Это обеспечивает лучшее понимание особенности работы каждого алгоритма и их эффективность в различных сценариях.

## SOFTWARE FOR STEP-BY-STEP ANALYSIS OF DATA SEARCH AND SORTING ALGORITHMS

E.V. DOVGULEVICH, V.S. SHUKA, O.G. SHEVCHUK

**Abstract.** A software for step-by-step analysis of data search and sorting algorithms has been developed. The main capabilities and operating principles of the software are described. A description of the search and sorting algorithms used is provided. Methods for estimating the computational complexity of algorithms are considered.

*Keywords:* search algorithms, sorting algorithms, computational complexity of algorithms.

## Список литературы

1.Клейнберг Дж., Тардос Е. Алгоритмы: разработка и применение. Классика Computer Science, СПб., 2016.
2.Левитин А. В. Алгоритмы. Введение в разработку и анализ, М., 2006.
3.Стивен С.С. Алгоритмы. Руководство по разработке., СПб., 2022.
4.Рафгарден Т. Совершенный алгоритм. Основы., СПб., 2019.
5.Как определить теоретическую сложность алгоритма. Интернет-источник: https://uchet-jkh.ru/i/kak-opredelit-teoreticeskuyu-sloznost-algoritma
6.Лафоре Р. Структуры данных и алгоритмы в Java. Классика Computers Science. 2-е изд., СПб., 2013.