**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра** **ВТ**

**ОТЧЕТ**

**по лабораторной работе №3**

**по дисциплине «Алгоритмы и структуры данных»**

**Тема: «Алгоритмы сортировки»**

**Вариант 5**

|  |  |  |
| --- | --- | --- |
| Студент гр. 0322 |  | Кренев И.Ф. |
| Преподаватель |  | Пестерев Д.О. |

Санкт-Петербург

2022

**Постановка задачи**

**Цель работы**

Получение практических навыков сортировки массивов с использованием различных алгоритмов. Практическое определение наиболее быстрого алгоритма сортировки для различных случаев.

**Формулировка задания**

Для каждого алгоритма сортировки указать временную асимптотическую сложность для лучшего, худшего случая и среднего случая, а также пространственную сложность, подкрепив это логическими построениями (в меру своих сил). Свести получившиеся результаты в таблицу. Построить график зависимости времени выполнения от размера входных данных и определить временную асимптотическую сложность для лучшего, худшего, среднего случая практически. Определить наиболее быстрый алгоритм сортировки. Сравнить скорость его выполнения с одним из алгоритмов сортировки из базовых библиотек языка.

**Описание программы**

Написанная программа реализует 2 класса – ArraySorts и Main. Класс ArraySorts отвечает за реализацию алгоритмов сортировок и хранение оригинального и отсортированного массивов. Класс Main реализует логику отображения и взаимодействия с элементами на веб-странице.

Методы класса ArraySorts: insertionSort, selectionSort, bubbleSort, mergeSort, shellSort, quickSort, defaultSort отвечают за реализацию алгоритмов сортировок. Метод timeWrapper – это метод-обертка, позволяющий определить время выполнения функции, переданной в неё в качестве параметра.

**Асимптотическая сложность алгоритмов**

**Сортировка вставками**

В этом алгоритме и во всех последующих поля класса, отвечающие за хранение оригинального и отсортированного массивов не будут участвовать в подсчете пространственной сложности, поскольку самими алгоритмами они не используются.

Наихудшим случаем будет являться массив, изначально отсортированный по убыванию. В этом случае алгоритм пройдет по всем i-ым и j-ым ячейкам массива (O(N2)). Лучшим случаем является уже отсортированный по возрастанию массив. В этом случае алгоритму достаточно одного прохода по массиву (O(N)). Сложность среднего случая алгоритма соответствует его худшему случаю.

Пространственная сложность составляет O(1) - дополнительная переменная.

**Сортировка выбором**

В данном алгоритме сложность для всех случаев составляет O(N2), так как вне зависимости от начальных условий алгоритм проходит по всем i-ым и j-ым ячейкам массива.

Пространственная сложность совпадает со сложностью сортировки вставками.

**Пузырьковая сортировка**

Асимптотическая и пространственная сложность данного алгоритма совпадает с сортировкой вставками по аналогичным причинам.

**Сортировка слиянием**

Асимптотическая сложность для всех случаев равна O(N\*logN), поскольку используется метод «Разделяй и властвуй».

Поскольку в алгоритме используется дополнительный массив с размерностью оригинального – пространственная сложность равняется O(N).

**Сортировка Шелла**

Наихудшим случаем будет являться массив, изначально отсортированный по убыванию. В этом случае алгоритм пройдет по всем i-ым и j-ым ячейкам массива (O(N2)).

Поскольку сортировка Шелла является доработанной версией сортировки вставками, к коду сортировки вставками добавляется еще один цикл со значением сдвига, уменьшающимся каждую итерацию, а сортировка начинается от элемента с индексом, равным этому значению. Исходя из этого, лучшим случаем является O(N\*log2N).

Сложность среднего случая зависит от выбранного расстояния между сравниваемыми элементами. Мной был выбран способ задания расстояния, где первое значение — это половина длины сортируемого массива, второе — половина от предыдущего и так далее, каждый раз округляя значение до целого числа O(N2).

Пространственная сложность составляет O(1) - дополнительная переменная.

**Быстрая сортировка**

Худший случай может возникнуть в ситуации, когда на каждом этапе выбирается наибольший или наименьший элемент из обрабатываемых. В этом случае каждое разделение даёт два подмассива размерами 1 и N, что в общем дает сложность O(N2).

В лучшем случае каждую операция разделяет массив на две одинаковые части. В этом случае глубина рекурсии составит log2n (сложность - O(N\*logN)).

Среднюю сложность алгоритма возможно рассчитать лишь вероятностно. В общем случае, хорошим будет считаться постоянное разделение подмассивов на одинаковые части. В этом случае сложность будет равна лучшему случаю.

Поскольку в алгоритме используется дополнительный массив с размерностью оригинального – пространственная сложность равняется O(N).

Расчеты сложностей всех рассматриваемых алгоритмов были занесены в таблицу 1.

Таблица 1. Расчет сложности алгоритмов

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Название алгоритма / случай | Худший | Средний | Лучший | Пространственная  сложность |
| Сортировка вставками | O(N2) | O(N2) | O(N) | O(1) |
| Сортировка выбором | O(N2) | O(N2) | O(N2) | O(1) |
| Пузырьковая сортировка | O(N2) | O(N2) | O(N) | O(1) |
| Сортировка слиянием | O(N\*logN) | O(N\*logN) | O(N\*logN) | O(N) |
| Сортировка  Шелла | O(N2) | O(N2) | O(N\*log2N) | O(1) |
| Быстрая сортировка | O(N2) | O(N\*logN) | O(N\*logN) | O(N) |

**Зависимость времени выполнения от количества элементов**

Для расчета зависимости времени выполнения алгоритмов от количества элементов была реализована функция checkTime. Код функции представлен в приложении 1.

На рисунке 1 представлен общий график зависимости всех алгоритмов сортировок.

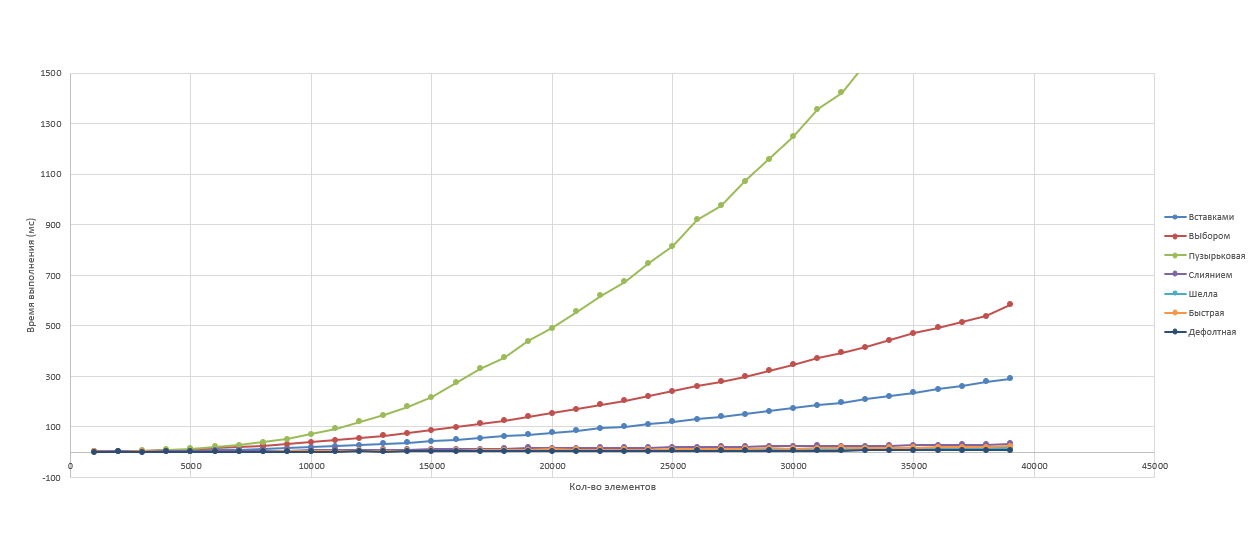


Рисунок 1. Общий график

Для определения самого быстрого алгоритма сортировки необходимо сократить рассматриваемую временную область до 50 мс. График с сокращенной временной областью представлен на рисунке 2.

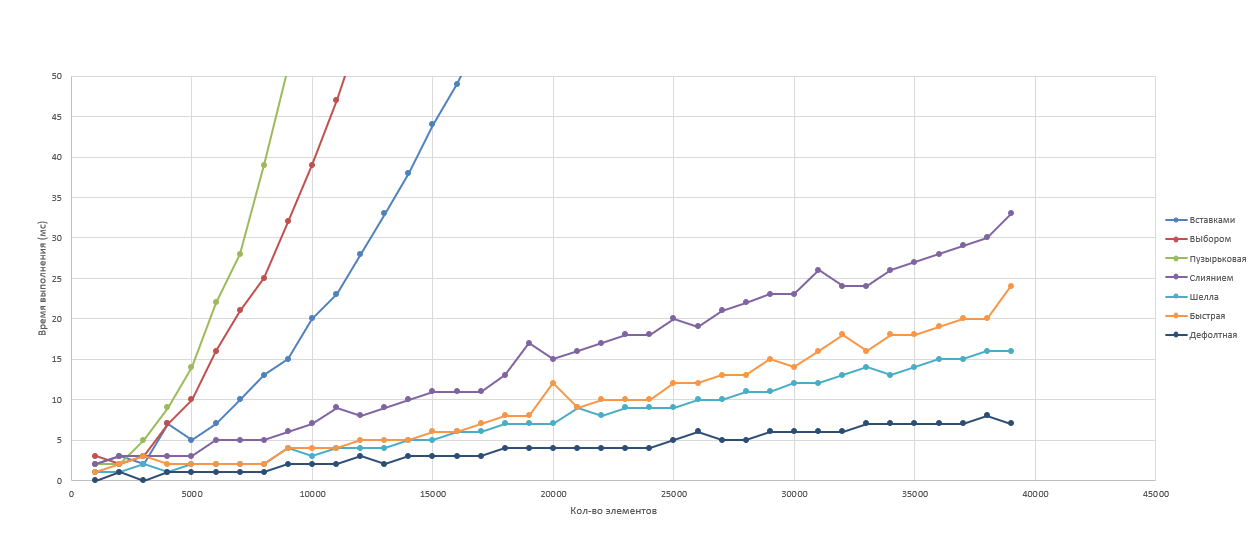


Рисунок 2. График с сокращенной временной областью

Исходя из рисунка 2 мы можем судить, что наиболее быстрым алгоритмом сортировки является базовая (дефолтная) сортировка языка программирования JavaScript. Однако, нужно отметить, что используемый внутри неё алгоритм может отличаться в зависимости от используемого браузера и количества элементов.

Самым быстрым из изучаемых в лабораторной работе алгоритмом является сортировка Шелла, даже с учетом того, что мною был выбран не самый оптимальный способ выбора расстояния между сравниваемыми элементами.

**Практическое определение сложности**

Для расчета сложности была реализована функция checkDifficult, в которой происходит генерация трех массивов для всех случаев. Логика генерации соответствует теоретическим данным из раздела «Асимптотическая сложность алгоритмов». Код функции расположен в дополнении 1.

Скриншоты результатов выполнения функции представлены ниже.

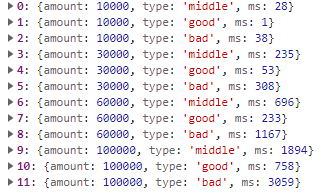


Рисунок 3. Сортировка вставками

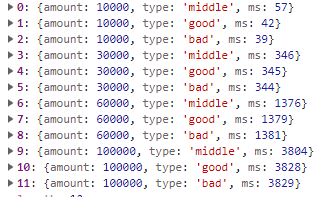


Рисунок 4. Сортировка выбором

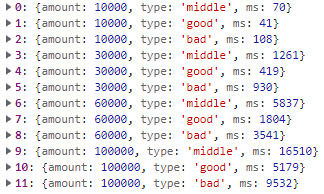


Рисунок 5. Пузырьковая сортировка

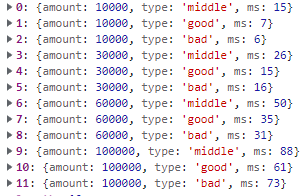


Рисунок 6. Сортировка слиянием

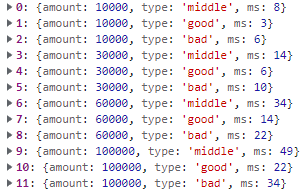


Рисунок 7. Сортировка Шелла

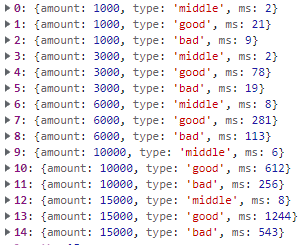


Рисунок 8. Быстрая сортировка

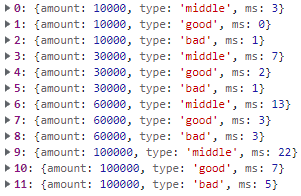


Рисунок 9. Дефолтная сортировка

**Пример работы программы**

Интерфейс веб-страницы имеет интуитивно понятный интерфейс с подсказками для полей ввода. Имеется возможность ввести исходный массив вручную либо сгенерировать его рандомно. Начальный интерфейс представлен на рисунке 10.

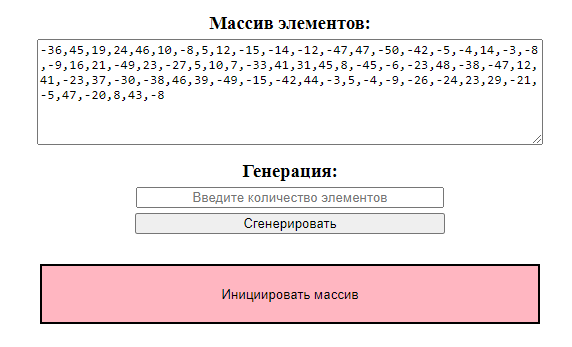


Рисунок 10. Исходное состояние программы

По нажатию кнопки «Инициировать массив» становится доступным выбор алгоритма сортировки (рисунок 11). После изменения исходного массива программа возвращается к исходному состоянию, отображенному на рисунке.

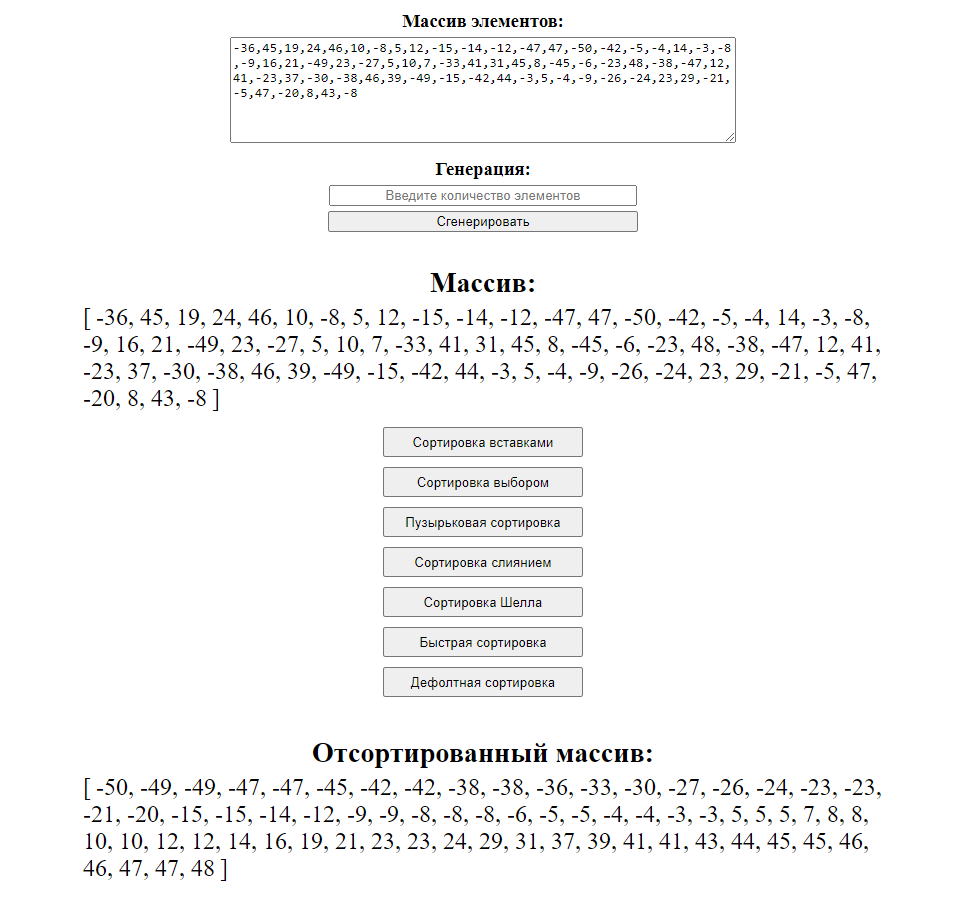


Рисунок 11. Выбор алгоритма сортировки