**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»**

**Отчёт по лабораторной работе № 5**

**«Работа с сортировками»**

**Выполнил работу**

**Кулинич Павел Васильевич**

**Академическая группа №3111**

**Принято**

**Ментор, Владислав Вершинин**

**Санкт-Петербург**

**2024**

**Подсчет памяти для трех алгоритмов :**

Пузырьковая сортировка

Дополнительная память:

Используется только несколько переменных (например, swapped, i, j), которые занимают фиксированное количество памяти. Это O(1).

Общая память:

Память для хранения входного массива: O(N), где N — количество элементов в массиве.

Итог:

Общая память: O(N)

Быстрая сортировка

Дополнительная память:

Быстрая сортировка использует рекурсию, и глубина рекурсии в худшем случае может достигать O(N) (например, если массив уже отсортирован). Однако, в среднем случае глубина рекурсии составляет O (log N).

В дополнение к этому, используется несколько переменных для хранения индексов, что также занимает O(1).

Общая память:

Память для хранения входного массива: O(N).

Итог:

Общая память: O(N) (из-за хранения массива) + O( log N) (для стека вызовов) = O(N)

Сортировка подсчетом

Дополнительная память:

Создается дополнительный массив count размером O(k + 1), где k — максимальное значение в массиве. Это необходимо для хранения частоты каждого элемента.

Общая память:

Память для хранения входного массива: O(N).

Итог:

Общая память: O(N) + O(k) = O(N + k)

**Подсчет асимптотики :**

1) Пузырьковая сортировка (Bubble Sort)

Временная сложность:

Лучший случай: O(N) — когда массив уже отсортирован. Алгоритм проходит один раз по массиву и не делает ни одной замены.

Средний случай: O(N²) — в среднем, алгоритм выполняет O(N) итераций, и на каждой итерации в худшем случае требуется O(N) сравнений.

Худший случай: O(N²) — когда массив отсортирован в обратном порядке. Алгоритм выполняет максимальное количество сравнений и обменов.

Пространственная сложность: O(1) — используется только фиксированное количество дополнительных переменных.

2) Быстрая сортировка (Quick Sort)

Временная сложность:

Лучший случай: O(N log N) — когда массив делится на две равные части на каждом шаге.

Средний случай: O(N log N) — в среднем, алгоритм также делит массив на две части, что приводит к логарифмическому числу уровней рекурсии.

Худший случай: O(N²) — когда массив уже отсортирован или содержит много одинаковых элементов, и алгоритм выбирает наихудший опорный элемент (например, самый левый или самый правый).

Пространственная сложность: O(log N) — в среднем случае, из-за стека рекурсии. В худшем случае может достигать O(N).

3) Сортировка подсчетом (Counting Sort)

Временная сложность:

Лучший случай: O(N + k) — где N — количество элементов в массиве, а k — максимальное значение в массиве.

Средний случай: O(N + k) — аналогично, поскольку алгоритм всегда проходит по всем элементам массива и массиву частот.

Худший случай: O(N + k) — время выполнения не зависит от порядка элементов.

Пространственная сложность: O(N + k) — для хранения входного массива и массива частот.

**Тесты :**

Пузырьковая сортировка :

Функция bubbleSort: Реализует алгоритм пузырьковой сортировки, как и в вашем исходном коде.

Функция testBubbleSort: Содержит три теста:

Лучший случай: Проверяет, что уже отсортированный массив остается без изменений.

Средний случай: Проверяет, что случайный массив сортируется правильно.

Худший случай: Проверяет, что массив, отсортированный в обратном порядке, сортируется в правильном порядке.

assert: Используется для проверки, что результат сортировки соответствует ожидаемому результату. Если условие assert не выполняется, программа завершится с ошибкой.

Быстрая сортировка :

Функция partition: Реализует разбиение массива на основе выбранного опорного элемента.

Функция quickSort: Реализует алгоритм быстрой сортировки.

Функция testQuickSort: Содержит три теста:

Лучший случай: Проверяет, что уже отсортированный массив остается без изменений.

Средний случай: Проверяет, что случайный массив сортируется правильно.

Худший случай: Проверяет, что массив, отсортированный в обратном порядке, сортируется в правильном порядке.

assert: Используется для проверки, что результат сортировки соответствует ожидаемому результату. Если условие assert не выполняется, программа завершится с ошибкой.

Сортировка подсчёта :

Функция countingSort: Реализует алгоритм сортировки подсчетом.

Функция testCountingSort: Содержит три теста:

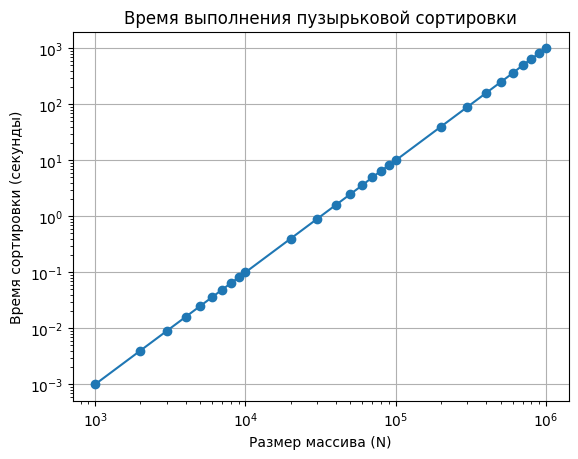
Лучший случай: Проверяет, что уже отсортированный массив остается без изменений.

Средний случай: Проверяет, что случайный массив сортируется правильно.

Худший случай: Проверяет, что массив, состоящий только из одного значения, сортируется правильно.

assert: Используется для проверки, что результат сортировки соответствует ожидаемому результату. Если условие assert не выполняется, программа завершится с ошибкой.

**Линейный график**



**Выводы**

**В данной программе реализованы два алгоритма сортировки: пузырьковая сортировка и сортировка подсчетом. Каждый из этих алгоритмов имеет свои особенности, преимущества и недостатки, которые влияют на их применение в различных ситуациях.**

**1. Пузырьковая сортировка**

**Асимптотика:**

* **Лучший случай: O(n) — когда массив уже отсортирован.**
* **Средний и худший случай: O(n²) — когда массив отсортирован в обратном порядке или случайно.**

**Практика: Пузырьковая сортировка является простым и интуитивно понятным алгоритмом, но из-за своей квадратичной сложности она неэффективна для больших массивов. В реальных приложениях ее можно использовать для небольших массивов или в образовательных целях для демонстрации принципов сортировки.**

**Выбросы: Выбросы могут возникать в случае, если массив содержит много одинаковых элементов, что может привести к большому количеству сравнений, но не к обменам. Это может замедлить алгоритм, хотя в некоторых случаях (например, если массив уже отсортирован) он может работать быстрее.**

**2. Сортировка подсчетом**

**Асимптотика:**

* **Лучший, средний и худший случай: O(n + k), где n — количество элементов в массиве, а k — максимальное значение элемента.**

**Практика: Сортировка подсчетом эффективна для сортировки массивов, элементы которых находятся в ограниченном диапазоне (например, от 0 до k). Она работает быстрее, чем сравнение, и может быть использована в ситуациях, когда известен диапазон значений. Это делает ее подходящей для сортировки целых чисел, например, в задачах, связанных с подсчетом частоты элементов.**

**Выбросы: Выбросы могут возникать, если массив содержит элементы, выходящие за пределы заданного диапазона (0 до k). В таких случаях алгоритм может не работать корректно, так как он предполагает, что все элементы находятся в пределах от 0 до k. Также, если k значительно больше n, это может привести к неэффективному использованию памяти.**

**Заключение**

**Выбор алгоритма сортировки зависит от конкретной задачи и характеристик данных. Пузырьковая сортировка может быть полезна для небольших массивов или в образовательных целях, тогда как сортировка подсчетом подходит для больших массивов с ограниченным диапазоном значений. Важно учитывать асимптотику алгоритмов и их поведение в различных сценариях, чтобы выбрать наиболее подходящий метод для решения конкретной задачи.**