Лабораторная работа #1

Создание компонента, реализующего алгоритм Быстрой сортировки

Выполнил студент 21ПИ3 Кутузов Всеволод

**Содержание**

**Быстрая сортировка (46)**

Описание алгоритма ------------------------------------------------------------------ 2

Сложность алгоритма ----------------------------------------------------------------- 3

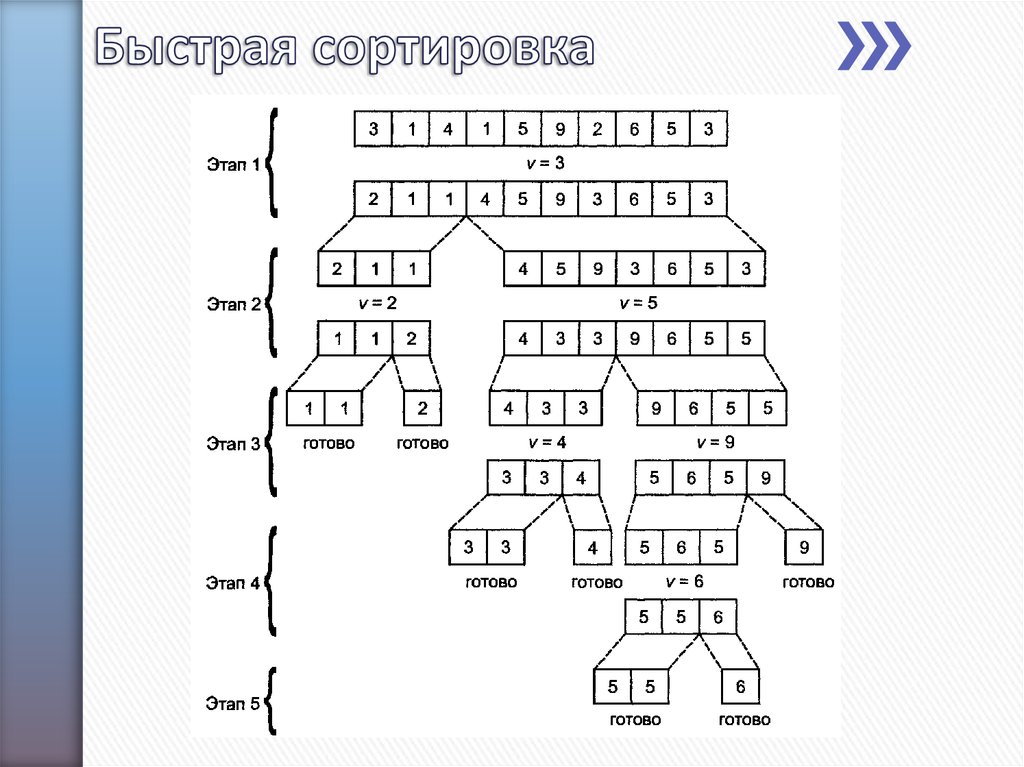
Сравнение и результат тестов ------------------------------------------------------- 3

**Алгоритм быстрой сортировки**

Алгоритм быстрой сортировки, также известный как алгоритм Хоара, является одним из самых эффективных алгоритмов сортировки. Он использует подход "разделяй и властвуй", который заключается в разбиении массива на подмассивы, сортировке каждого подмассива и объединении их в один отсортированный массив.

Основная идея быстрой сортировки заключается в выборе элемента, называемого опорным (), из массива и разбиении массива на две части: элементы, меньшие опорного, и элементы, большие опорного. Затем рекурсивно применяется алгоритм к обеим частям.

Шаги алгоритма быстрой сортировки:

1. **Выбор опорного элемента**: Выбирается опорный элемент из массива. Этот элемент может быть выбран различными способами, например, первый элемент, последний элемент или случайный элемент.
2. **Разделение массива**: Массив разделяется на две части: элементы, меньшие опорного, и элементы, большие опорного. Это выполняется так, что все элементы слева от опорного меньше него, а все элементы справа от опорного больше него.
3. **Рекурсивная сортировка**: Рекурсивно применяется алгоритм к обеим частям массива.
4. **Объединение подмассивов**: После завершения сортировки подмассивов они объединяются в один отсортированный массив.

**Сложность алгоритма**

Алгоритм быстрой сортировки (Quick Sort) является одним из самых эффективных алгоритмов сортировки в среднем случае. Его асимптотическая сложность зависит от выбора опорного элемента и наличия повторяющихся элементов в массиве. Обычно алгоритм быстрой сортировки имеет среднюю временную сложность O(n\*log n) n - количество элементов в массиве.

В лучшем случае, когда опорный элемент разбивает массив на две примерно равные части, сложность алгоритма быстрой сортировки составляет O(n\*log n). Это происходит, когда на каждом шаге деления массива опорный элемент попадает ровно в середину массива.

В худшем случае, когда опорный элемент является наименьшим или наибольшим элементом в массиве, и он не разбивает массив на две примерно равные части, сложность алгоритма быстрой сортировки составляет O(n\*\*2) Это происходит, когда массив уже отсортирован по возрастанию или убыванию, а опорный элемент выбирается как первый или последний элемент массива. Однако, вероятность такого худшего случая крайне мала при случайном выборе опорного элемента или при использовании случайного выбора опорного элемента.

Средняя временная сложность быстрой сортировки составляет O(n\*log n), что делает ее одним из самых эффективных алгоритмов сортировки в среднем случае.

**Сравнение и результат тестов**

Проведено тестирование алгоритма QuickSort на разнообразных типах данных (int, double, char, float). В качестве сравнения моей реализации по времени выполнения, была взята стандартная функция qsort из библиотеки stdlib.h

Проведенное тестирование двух реализаций алгоритма QuickSort - стандартной и qsort из stdlib.h, показало, что моя собственная реализация в среднем немного превосходит по скорости выполнения стандартную реализацию. Однако разница в производительности не является значительной и скорее всего не будет иметь существенного влияния на общую производительность системы или приложения.