Сортировки

**Анализ эффективности четырех наиболее популярных алгоритмов сортировки.**

Работа студента группы МФ-12   
Наймытенко Глеба

Оглавление

[Сравнение четырех видов сортировок: 2](#_Toc419669530)

[Результаты проведенного исследования 3](#_Toc419669531)

[Анализ результатов 5](#_Toc419669532)

## Сравнение четырех видов сортировок:

1. [Пузырьковая сортировка](BubbleSort)
2. [Сортировка вставками](InsertionSort)
3. [Сортировка Шелла](ShellSort)
4. [Быстрая сортировка](Quicksort)

Переход по гиперссылкам приведет в папку с исходным кодом реализации каждой сортировки на С.

Данные сортировки будут сравниваться по трем параметрам:

* Количество перестановок
* Количество сравнений
* Время выполнения программы

# Результаты проведенного исследования

Все точные значения содержатся в [таблице](ResultTable.xlsx), здесь же приведены лишь диаграммы, наглядно показывающие основные результаты.

Таблица Количество сравнений

Таблица Количество Перестановок

Таблица Время выполнения в сек (на одном компьютере)

**Примечание:** Время работы программ не зависело от человека и замерялось программно.   
Все программы унифицированы, и различаются лишь функциями для сортировки массива**.**

### Анализ результатов

Во всех экспериментах, кроме наилучшего набора входящих данных, быстрейшей по времени исполнения и оптимальной по количеству сравнений и перестановок была Быстрая сортировка, чем полностью оправдала свое название, а так же популярность как наиболее используемого алгоритма сортировки.

Быстрая сортировка была менее эффективной, чем другие сортировки лишь на практически отсортированном массиве. Она уступила только сортировке пузырьком, которая, не смотря на наихудшие показатели в других случаях, оказалась быстрейшей на лучшем наборе. Причем как по времени работы, так и по оптимальности (количеству перестановок и сравнений). Так же Быструю сортировку по количеству сравнений на наилучшем наборе опередила сортировка вставками, что, тем не менее, не повлияло положительно на реальное время работы, и быстрая сортировка все равно оказалась быстрее (Таблица 3 Время выполнения в сек (на одном компьютере)).

Быстрая сортировка показала свою максимальную эффективность на больших массивах из случайных элементов, показав гигантскую (Таблица 1 Количество сравнений) разницу по времени работы и оптимальности с наивными сортировками (пузырьком и вставками). А так же показав свое превосходство над Сортировкой Шелла. Так же эта сортировка была быстрейшей на худшем наборе. Количество перестановок в наивных сортировках на массиве из 50 тыс. элементов, отсортированном в обратном порядке, превысило 1,2 млрд ,а быстрой сортировке потребовалось менее миллиона перестановок на том же наборе.

Сортировка Шелла (при выборе опорного элемента по методу Шелла  ~d_1 = N/2,  d_i = d_{i-1} / 2,  d_k = 1. В худшем случае, сложность алгоритма составит O( N^2 )) показывала стабильно хорошие результаты, но в среднем была медленнее быстрой. Тем не менее, она имеет преимущества перед ней:

* 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 ( на модифицированных алгоритмах выбора опорного элемента).

Эффективность данной сортировки очень зависит от выбора опорного элемента.

Существует несколько известных методов выбора опорного элемента, которые повышают эффективность сортировки Шелла:

* предложенная Хиббардом последовательность: все значения ~2^i-1 \le N, i \in \mathbb N; такая последовательность шагов приводит к алгоритму сложностью O(N^{3/2});
* предложенная [Седжвиком](https://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D0%B4%D0%B6%D0%B2%D0%B8%D0%BA,_%D0%A0%D0%BE%D0%B1%D0%B5%D1%80%D1%82) последовательность: ~d_i = 9\cdot2^i - 9\cdot2^{i/2} + 1, если i четное и ~d_i = 8\cdot2^i - 6\cdot2^{(i+1)/2} + 1, если i нечетное. При использовании таких приращений средняя сложность алгоритма составляет: O(n^{7/6}), а в худшем случае порядка O(n^{4/3}).
* предложенная Праттом последовательность: все значения ~2^i\cdot3^j \le N/2, i, j \in \mathbb N; в таком случае сложность алгоритма составляет O(N (log N)^2);
* эмпирическая последовательность Марцина Циура ~d \in \left\{1, 4, 10, 23, 57, 132, 301, 701, 1750\right\}; является одной из лучших для сортировки массива ёмкостью приблизительно до 4000 элементов.[[2]](https://ru.wikipedia.org/wiki/%D0%A1%D0%BE%D1%80%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%BA%D0%B0_%D0%A8%D0%B5%D0%BB%D0%BB%D0%B0#cite_note-mciura-2);
* эмпирическая последовательность, основанная на  [числах Фибоначчи](https://ru.wikipedia.org/wiki/%D0%A7%D0%B8%D1%81%D0%BB%D0%B0_%D0%A4%D0%B8%D0%B1%D0%BE%D0%BD%D0%B0%D1%87%D1%87%D0%B8):  ~d \in \left\{F_n\right\};
* все значения ~(3^j-1) \le N, j \in \mathbb N; такая последовательность шагов приводит к алгоритму сложностью O(N^{3/2}).

Судя по результатам, Быстрая сортировка и сортировка Шелла показали полное превосходство над наивными сортировками. Пузырьковая сортировка в реальной жизни может быть использована лишь, если известно что массив для сортировки будет практически отсортирован. А повода для использования сортировки вставками ,помимо учебного, я не вижу.

Сортировку же Шелла стоит использовать при малом доступном количестве стековой памяти и больших оперируемых массивах, в которых возможен худший для быстрой сортировки случай разбиения, когда глубина рекурсии достигнет n.

В остальном же алгоритм Быстрой Сортировки имеет значительное количество преимуществ:

* Один из самых быстродействующих (на практике) из алгоритмов внутренней сортировки общего назначения.
* Прост в реализации.
* Требует лишь O(\lg n) дополнительной памяти для своей работы. (Не улучшенный рекурсивный алгоритм в худшем случае O(n \log n) памяти).
* Допускает естественное распараллеливание (сортировка выделенных подмассивов в параллельно выполняющихся подпроцессах).
* Работает на связных списках и других структурах с последовательным доступом, допускающих эффективный проход как от начала к концу, так и от конца к началу.

*Что дает мне сделать вывод о том, что быстрая сортировка – наилучший сортировочный алгоритм****.***

В том числе лучшим он является по тому, что легко модифицируется для придания алгоритму устойчивости и «защиты от худшего случая» — устранения деградации производительности и переполнения стека вызовов из-за большой глубины рекурсии при неудачных входных данных.

Что касается первой проблемы, то она решается путём расширения ключа исходным индексом элемента в массиве. В случае равенства основных ключей сравнение производится по индексу, исключая, таким образом, возможность изменения взаимного положения равных элементов. Эта модификация не бесплатна — она требует дополнительно O(n) памяти и одного полного прохода по массиву для сохранения исходных индексов.

Вторая проблема, в свою очередь, решается по двум разным направлениям: снижение вероятности возникновения худшего случая путём специального выбора опорного элемента и применение различных технических приёмов, обеспечивающих устойчивую работу на неудачных входных данных. Для первого направления:

* Выбор среднего элемента. Устраняет деградацию для предварительно отсортированных данных, но оставляет возможность случайного появления или намеренного подбора «плохого» массива.
* Выбор медианы из трёх элементов: первого, среднего и последнего. Снижает вероятность возникновения худшего случая, по сравнению с выбором среднего элемента.
* Случайный выбор. Вероятность случайного возникновения худшего случая становится исчезающе малой, а намеренный подбор — практически неосуществимым. Ожидаемое время выполнения алгоритма сортировки составляет O(*n* lg *n*).

Недостаток всех усложнённых методов выбора опорного элемента — дополнительные накладные расходы; впрочем, они не так велики.

Во избежание отказа программы из-за большой глубины рекурсии могут применяться следующие методы:

* При достижении нежелательной глубины рекурсии переходить на сортировку другими методами, не требующими рекурсии. Можно заметить, что алгоритм очень хорошо подходит для такого рода модификаций, так как на каждом этапе позволяет выделить непрерывный отрезок исходного массива, предназначенный для сортировки, и то, каким методом будет отсортирован этот отрезок, никак не влияет на обработку остальных частей массива.
* Модификация алгоритма, устраняющая одну ветвь рекурсии. Вместо того, чтобы после разделения массива вызывать рекурсивно процедуру разделения для обоих найденных подмассивов, рекурсивный вызов делается только для меньшего подмассива, а больший обрабатывается в цикле в пределах этой же процедуры.
* Разбивать массив не на две, а на три части[[2]](https://ru.wikipedia.org/wiki/%D0%91%D1%8B%D1%81%D1%82%D1%80%D0%B0%D1%8F_%D1%81%D0%BE%D1%80%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%BA%D0%B0#cite_note-2).