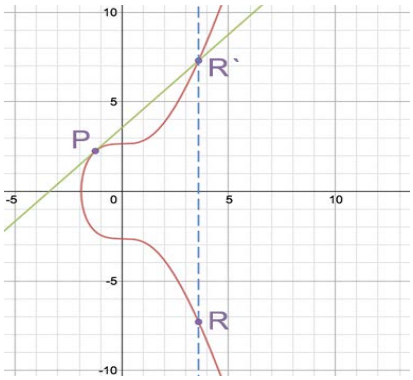

Конечным полем называется конечное множество элементов, замкнутое по отношению к двум заданным в нем операциям комбинирования элементов [5]. Под замкнутостью понимается тот факт, что результаты операций не выходят за пределы конечного множества введенных элементов.

Чтобы использовать ECDSA, такой протокол биткоина должен зафиксировать набор параметров для эллиптической кривой и ее конечного поля, чтобы эти параметры знали и применяли все пользователи протокола. Иначе каждый будет решать свои собственные уравнения, которые не будут сходиться друг с другом, и они никогда ни о чем не договорятся.

Эти зафиксированные параметры включают в себя *уравнение кривой*, значение *модуля поля* и *базовую точку*, которая лежит на кривой. Последним параметром является *порядок* базовой точки, который в графическом виде можно представить себе как количество раз, которое базовая точка может быть прибавлена к себе до тех пор, пока ее касательная кривая не станет вертикальной. Этот параметр подбирается таким образом, чтобы он являлся очень большим простым числом.

Заключение. В наши дни криптовалюты продолжают свое развитие, число пользователей новыми платежными средствами постоянно растет. Популярность биткоина стимулирует создание других криптовалют, которые развиваются параллельно, но их возможности и популярность пока еще намного меньше. Однако вместе с тем следует отметить, что каждая из них основана на различных математических принципах, хотя идеология блокчейна (выстроенной по определённым правилам непрерывной последовательной цепочки блоков, содержащих всю полную информацию, копии которых хранятся независимо друг от друга на множестве разных компьютеров) присутствует в любой из них.

1. Свон, М. Блокчейн. Схема новой экономики / М. Свон – М.: Олимп-Бизнес, 2017. – 240с.
2. Филиппов, Е. Криптовалюта от А до Я / Е. Филиппов. – 2017. – 50с.
3. Алгоритм ECDSA / Википедия [Электронный ресурс]. – Режим доступа: http://ru.wikipedia.org/?oldid=90242139 (дата обращения: 12.01.2018).
4. Эллиптическая кривая / Википедия [Электронный ресурс]. – Режим доступа http://ru.wikipedia.org/?oldid=90075838 (дата обращения: 05.01.2018).
5. Биткоин / Википедия [Электронный ресурс]. – Режим доступа: http://ru.wikipedia.org/?oldid=90310153 (дата обращения: 15.01.2018).

ВЫБОР И ОПТИМИЗАЦИЯ АЛГОРИТМА СОРТИРОВКИ ДЛЯ ДОСТИЖЕНИЯ МАКСИМАЛЬНОГО БЫСТРОДЕЙСТВИЯ

Д.Ю. Романцов
Орша, Оршанский колледж ВГУ имени П.М. Машерова

Сортировка данных одна из самых распространённых задач. На данный момент существует около 40 видов сортировок [1]. Практически для всех определена временная сложность O, на основе которой можно определить их скорость. Однако из-за того, что это достаточно обобщённая характеристика, то сказать однозначно, какой алгоритм быстрее нельзя.

Целью исследования является выявление наиболее оптимального алгоритма сортировки массива данных эмпирическим способом.

Материал и методы. Проведём исследование в этом проблемном поле на базе наиболее известных представителей: глупая, пузырьковая, быстрая, шейкерная, гребешковая, слиянием, клоуна Бозо, простыми вставками, пирамидальная, интроспективная сортировки. В тесте участвуют множества с дробными случайно сгенерированными числами. Обозначим одно такое множество через A, его элемент как Ai, а длину n = |A|. Для улучшения статистических показателей, проведём тесты для 100 разных A одной n. При этом n = {10, 100, 1000, 5000, 10000, 50000, 100000, 300000, 500000, 1000000}, что даст лучшую иллюстрацию быстродействия.

Время – основной параметр, характеризующий быстродействие алгоритма. Называется также вычислительной сложностью. Для упорядочения важны худшее, среднее и лучшее поведение алгоритма в терминах мощности входного множества A. Для типичного алгоритма хорошее поведение – это O(n*log n) и плохое поведение – это $O(n^2)$, идеальное – O(n) [2].

В связи с тем, что алгоритмы являются достаточно простыми и используют несложные математические выражения, то стоит предположить, что время выполнения может быть достаточно небольшим. Поэтому использовать стандартные средства операционной системы (время отклика системного таймера около 10,0144 мс) не представляется возможным.

Чтобы получить точность порядка 10^{-6}с или по-другому 1 нс, воспользуемся счетчиком меток реального времени процессора (Time Stamp Counter). Начиная с Pentium III, процессоры обычно содержат его в доступной программистам зоне. TSC представляет собой 64-разрядный регистр, содержимое которого с каждым тактом процессора инкрементируется. Счет начинается с нуля каждый раз при старте ЭВМ [3].

Из этого следует, что время выполнения алгоритма напрямую зависит от частоты процессора. Для соблюдения равенства условий эксперимента, все замеры будут производиться на одном компьютере с процессором Intel Pentium G620 2.6 ГГц. Для тестирования написана программа на Delphi.

Результаты и их обсуждение. В результате проведённых измерений получена следующая таблица:

Таблица 1 – Время работы в мс алгоритма сортировки в зависимости от объёма массива

	10	100	1k	10 k	50 k	100 k	300 k	500 k	1 M
Глупая	0,0008	0,0442	2,8009	344,48	9521,73	38180,7	>40c	>40c	>40c
Пузырьковая	0,0005	0,064	5,78	466,27	11621,31	46685,3	>40c	>40c	>40c
Шейкерная	0,0005	0,0172	0,8584	68,752	1860,28	7760,9	81998	>40c	>40c
Гребешковая	0,0008	0,011	0,14	1,73	9,5	18	52	82	155
Быстрая	0,0010	0,0140	0,1266	1,0465	5,15284	10,186	30,898	51,7092	107,106
Слиянием	0,0022	0,0145	0,1627	1,7411	8,04589	16,220	50,975	86,4585	205,061
Простыми вставками	0,0003	0,0058	0,4628	43,701	1095,38	4306,7	39390	>40c	>40c
Клоуна Бозо	113,55	12000	>40c	>40c	>40c	>40c	>40c	>40c	>40c
Пирамидальная	0,0009	0,0129	0,1707	1,7915	9,32412	19,0624	60,281	102,920	218,26
Интроспективная	0,0009	0,0116	0,1576	1,0838	5,12777	10,195	30,853	52,2175	107,170

Как видно из таблицы алгоритмы разделились на две группы: существенно и не очень зависящие от n. В отношении первой группы можно сказать, что эти алгоритмы не очень эффективны, т.к. при 50000 элементах на *глупую* сортировку или *пузырьковую* требуется около 10 секунд. *Шейкерная* и с *простыми вставками* сортировки показывают такое же время на множестве в два раза большем. Пузырьковая и глупая сортировки примерно в 5 раз менее производительны и применимы для массивов до 6-7 тыс. элементов. Сортировка *Клоуна Бозо* является «шуточной» и достаточно быстро работает для массивов длиной не более 10 элементов. Остальные сортировки показывают достаточно близкий результат и вполне применимы для массивов в 1 млн. элементов. Естественно они могут сортировать и большие массивы. Например, быстрая сортировка справляется с 10 млн элементов за 1,3 с.

Таким образом, из рассмотренных сортировок наибольшим потенциалом обладают *быстрая* и *интроспективная*. Это не случайно, поскольку интроспективная является быстрой сортировкой с контролем глубины рекурсии. Если глубина превышает установленный лимит, то быстрая сортировка останавливается и вызывается другой алгоритм, выполняющий досортировку. Выполним оптимизацию её работы по следующим направлениям: выбор лимита глубины рекурсии; уменьшение количества математических операций; выбор запасного алгоритма досортировки.

Лимит глубины рекурсии H можно установить по-разному. Автор алгоритма Дэвид Мюссер предложил ориентироваться на величину H=log(n) [1]. Почему предложен именно десятичный логарифм не совсем понятно, т.к. в процессе выполнения рекурсивно вызываются 2 функции для левого и правого подмножеств, поэтому получается бинарное дерево, глубина которого определяется как $H=\log_2(n)$. В этом случае, например, для массива n=1млн. H=20. Здесь следует осознать тот факт, что между n и H есть прямая связь. Чем больше n, тем больше H, следовательно, вычислять текущую глубину с помощью $\log_2(n)$ на каждом шаге не будем, так как эта операция не очень быстрая. Вместо этого заведём счётчик числа вызовов функции сортировки подмножества. Когда значение счётчика превысит объём массива, то будем считать, что произошло переполнение и вызовем второй алгоритм. Для увеличения значения счётчика используем операцию +, вместо инкремента inc(). Измерения проводились при n = 500 тыс.

Таблица 2 – Замер времени при оптимизации

	Начальная реализация	Переход на + вместо inc()	Переход от $\log_2$ к объёму	Динамическое измен. границы
Время, мс	64,449	64,377	63,402	52,218

Девид Мюссер в своей работе указал, что запасным алгоритмом следует использовать *пирамидальную* сортировку, т.к. она всегда сортирует за n*log(n). Однако в тесте выше было показано, что гребешковая и сортировка слиянием в среднем работают быстрее, поэтому проверим все 3 вида сортировок. Худший результат оказался именно у пирамидальной.

Таблица 3 – Результаты теста работы с запасным алгоритмом

	Быстрая без ограничения	Быстрая + пирамидальная	Быстрая + гребешковая	Быстрая + слиянием
Время, мс	10,452	23,471	14,223	18,099

Заключение. Проведённые измерения показывают, что наиболее быстродействующими алгоритмами без использования распараллеливания являются быстрая и интроспективная сортировки. В качестве запасного алгоритма для интроспективной выбираем гребешковую сортировку.

1. Описание алгоритмов сортировок [Электронный ресурс]: Коллекция сортировок / Валерий Макаров. – РФ, 2015. – Режим доступа: http://sorting.valemak.com. – Дата доступа: 15.12.2017.
2. Седжвик, Р. Фундаментальные алгоритмы на С. Ч. 1–4. Анализ. Структуры данных. Сортировка. Поиск [Текст] : Пер. с англ. / Роберт Седжвик. – СПб. : ООО «ДиаСофтЮП», 2003. – 672 с.
3. Точное время: измеряем, применяем [Электронный ресурс]: Социальное Интернет СМИ об IT / begin_end. – РФ, 2009. – Режим доступа: http://habrahabr.ru/post/75234/. – Дата доступа: 15.12.2017.
4. Musser, D. Introspective Sorting and Selection Algorithms / D. Musser // Software: Practice and Experience. – 1997. – August, volume 27, issue 8. – Pages 983–999.

О ИЗМЕРЕНИЯХ ВРЕМЕННЫХ ЗАТРАТ АЛГОРИТМОВ, РЕАЛИЗОВАННЫХ НА ЯЗЫКАХ ВЫСОКОГО УРОВНЯ

М.Г. Семёнов
Витебск, ВГУ имени П.М. Машерова

Сравнение производительности различных алгоритмов является важной задачей при изучении таких дисциплин, как «Методы алгоритмизации и программирования», «Алгоритмы и структуры данных», «Вычислительные методы», а также при разработке программного обеспечения. Одной из наиболее важных характеристик производительности являются