«КАЗАНСКИЙ ПРИВОЛЖСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

ИНСТИТУТ ВЫЧИСЛИТЕЛЬНОЙ МАТЕМАТИКИ И ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

КАФЕДРА СИСТЕМНОГО АНАЛИЗА И ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

Отчет по дисциплине «Алгоритмы и анализ сложности»

на тему «Экспериментальный анализ различных методов сортировки»

Выполнил:

студент 3 курса

группы 09-641

Десятов Александр Геннадьевич

Преподаватель:

Васильев Александр Валерьевич

Казань
2019

Оглавление

Введение.

В ходе выполнения практического задания на языке программирования C# были реализованы различные методы сортировок:

1) глупая сортировка,
2) сортировка пузырьком,
3) сортировка вставками,
4) сортировка Шелла,
5) быстрая сортировка,
6) сортировка слиянием,
7) сортировка кучей,
8) поразрядная сортировка.

А также для сравнения использовалась (9) сортировка, встроенная в язык программирования.

Для сортировок использовались массивы различных типов данных:

1) однобайтовый тип данных byte (массивы, состоящие из цифр из цифр от 0 до 9),
2) целочисленный тип данных int,
3) строковый тип данных string,
4) тип данных, представляющий собой дату и время, DateTime.

Для анализа сортировок были реализованы различные генерации массивов:

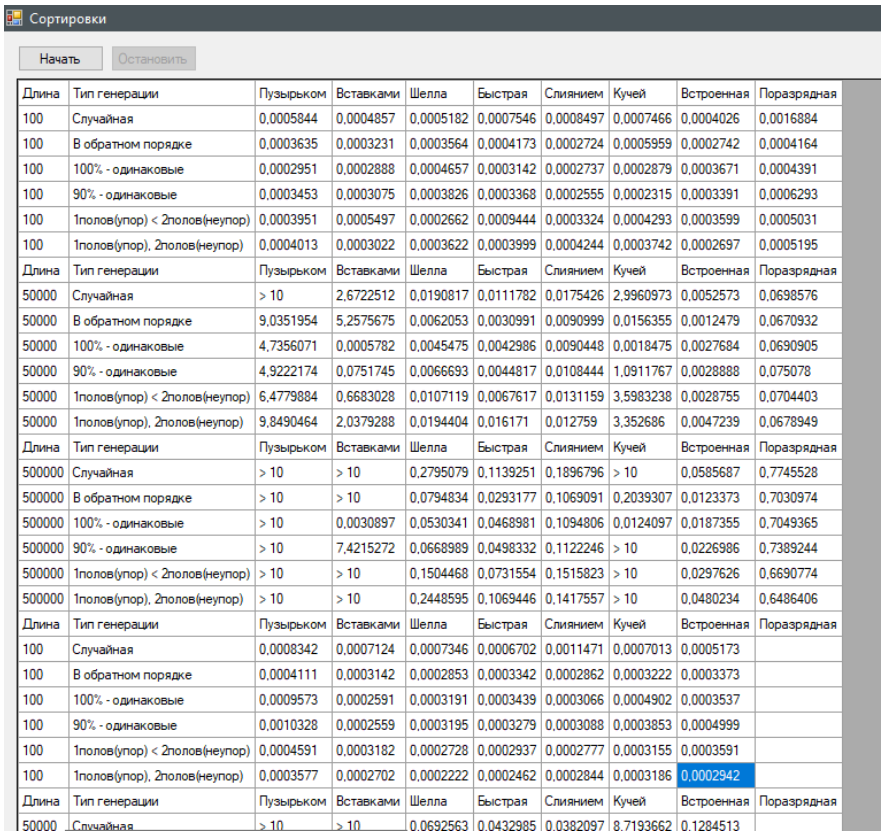
1) случайная генерация,
2) генерация массивов, упорядоченных в обратном порядке,
3) генерация массивов, на 100% состоящих из одинаковых элементов,
4) генерация массивов, на 90% состоящих из одинаковых элементов,
5) генерация массивов, у которых первая половина элементов упорядочена и любой элемент второй половины больше самого большого элемента из первой половины.
6) генерация массивов, у которых первая половина элементов упорядочена, а вторая половина случайно сгенерирована из элементов, которые могут быть как больше, так и меньше элементов первой половины.

Кроме того, сортировки анализировались на массивах различной длины:

1) 100 элементов,
2) 50000 элементов,
3) 500000 элементов.

Получение результатов.

Чтобы получить наиболее точные результаты, одна и та же сортировка проводилась несколько раз на массиве каждой длины каждого типа данных каждой генерации. Затем в таблицу программно записывалось среднее значение времени сортировки в секундах.



Длина	Тип генерации	Пузырьком	Вставками	Шелла	Быстрая	Слиянием	Кучей	Встроенная	Поразрядная
100	Случайная	0,0005844	0,0004857	0,0005182	0,0007546	0,0008497	0,0007466	0,0004026	0,0016884
100	В обратном порядке	0,0003635	0,0003231	0,0003564	0,0004173	0,0002724	0,0005959	0,0002742	0,0004164
100	100% - одинаковые	0,0002951	0,0002888	0,0004657	0,0003142	0,0002737	0,0002879	0,0003671	0,0004391
100	90% - одинаковые	0,0003453	0,0003075	0,0003826	0,0003368	0,0002555	0,0002315	0,0003391	0,0006293
100	1полов(упор) < 2полов(неупор)	0,0003951	0,0005497	0,0002662	0,0009444	0,0003324	0,0004293	0,0003599	0,0005031
100	1полов(упор), 2полов(неупор)	0,0004013	0,0003022	0,0003622	0,0003999	0,0004244	0,0003742	0,0002697	0,0005195
Длина	Тип генерации	Пузырьком	Вставками	Шелла	Быстрая	Слиянием	Кучей	Встроенная	Поразрядная
50000	Случайная	> 10	2,6722512	0,0190817	0,0111782	0,0175426	2,9960973	0,0052573	0,0698576
50000	В обратном порядке	9,0351954	5,2575675	0,0062053	0,0030991	0,0090999	0,0156355	0,0012479	0,0670932
50000	100% - одинаковые	4,7356071	0,0005782	0,0045475	0,0042986	0,0090448	0,0018475	0,0027684	0,0690905
50000	90% - одинаковые	4,9222174	0,0751745	0,0066693	0,0044817	0,0108444	1,0911767	0,0028888	0,075078
50000	1полов(упор) < 2полов(неупор)	6,4779884	0,6683028	0,0107119	0,0067617	0,0131159	3,5983238	0,0028755	0,0704403
50000	1полов(упор), 2полов(неупор)	9,8490464	2,0379288	0,0194404	0,016171	0,012759	3,352686	0,0047239	0,0678949
Длина	Тип генерации	Пузырьком	Вставками	Шелла	Быстрая	Слиянием	Кучей	Встроенная	Поразрядная
500000	Случайная	> 10	> 10	0,2795079	0,1139251	0,1896796	> 10	0,0585687	0,7745528
500000	В обратном порядке	> 10	> 10	0,0794834	0,0293177	0,1069091	0,2039307	0,0123373	0,7030974
500000	100% - одинаковые	> 10	0,0030897	0,0530341	0,0468981	0,1094806	0,0124097	0,0187355	0,7049365
500000	90% - одинаковые	> 10	7,4215272	0,0668989	0,0498332	0,1122246	> 10	0,0226986	0,7389244
500000	1полов(упор) < 2полов(неупор)	> 10	> 10	0,1504468	0,0731554	0,1515823	> 10	0,0297626	0,6690774
500000	1полов(упор), 2полов(неупор)	> 10	> 10	0,2448595	0,1069446	0,1417557	> 10	0,0480234	0,6486406
Длина	Тип генерации	Пузырьком	Вставками	Шелла	Быстрая	Слиянием	Кучей	Встроенная	Поразрядная
100	Случайная	0,0008342	0,0007124	0,0007346	0,0006702	0,0011471	0,0007013	0,0005173	
100	В обратном порядке	0,0004111	0,0003142	0,0002853	0,0003342	0,0002862	0,0003222	0,0003373	
100	100% - одинаковые	0,0009573	0,0002591	0,0003191	0,0003439	0,0003066	0,0004902	0,0003537	
100	90% - одинаковые	0,0010328	0,0002559	0,0003195	0,0003279	0,0003088	0,0003853	0,0004999	
100	1полов(упор) < 2полов(неупор)	0,0004591	0,0003182	0,0002728	0,0002937	0,0002777	0,0003155	0,0003591	
100	1полов(упор), 2полов(неупор)	0,0003577	0,0002702	0,0002222	0,0002462	0,0002844	0,0003186	0,0002942	
Длина	Тип генерации	Пузырьком	Вставками	Шелла	Быстрая	Слиянием	Кучей	Встроенная	Поразрядная
50000	Случайная	> 10	> 10	0,0692563	0,0432985	0,0382097	8,7193662	0,1284513	

Рисунок 1 Оконное приложение с полученными результатами

Таблицы из оконного приложения копировались в другую программу (Excel), где по данным таблиц строились диаграммы.

Для результатов с большим диапазоном значений для наглядности строились не диаграммы самих значений, а диаграммы натурального логарифма от этих значений.

В связи с тем, что глупая сортировка работает долго на массивах большой длины и значительно отстает в производительности от других сортировок, в отчете представлена лишь ее реализация (в приложении), но для анализа она не используется.

Поразрядная сортировка анализируется только в массивах с целочисленным типом данных, так как для других исследуемых типов данных она не реализована.

Методика сравнения

Для анализа сортировок сначала нужно классифицировать массивы. В данной работе, в первую очередь, все массивы делятся на группы относительно типа данных. Затем каждая группа делится на подгруппы в зависимости от количества элементов в массиве. И каждая подгруппа подразделяется на маленькие группы в зависимости от того, каким образом сгенерированы массивы. Самая маленькая группа массивов сортируется разными способами.

В разделе «Результаты» представлены средние полученные значения и по ним построены диаграммы. Если заранее известна какая-нибудь информация о массиве, то с помощью диаграмм можно подобрать самую подходящую сортировку.

Например, необходимо отсортировать массив, про который известно, что он однобайтового типа данных и что он состоит из небольшого числа элементов. Пусть ранее уже проводилась сортировка этого массива, и после этого в его конец были добавлены новые элементы. На диаграмме есть колонки, соответствующие массивам, у которых первая половина отсортирована, а вторая нет. На диаграмме видно, что такой массив быстрее всего упорядочится с помощью сортировки Шелла.

По диаграмме можно также определить, какой алгоритм сортировки для массивов, по-разному сгенерированных, часто оказывается самым медленным или, наоборот, самым быстрым, что подсказывает, стоит ли его использовать, если заранее не известна информация о массиве.

Также с помощью диаграмм можно сравнить скорости сортировок массивов, по-разному сгенерированных. К примеру, заметно, что случайно сгенерированные массивы, как правило, сортируются дольше остальных массивов.

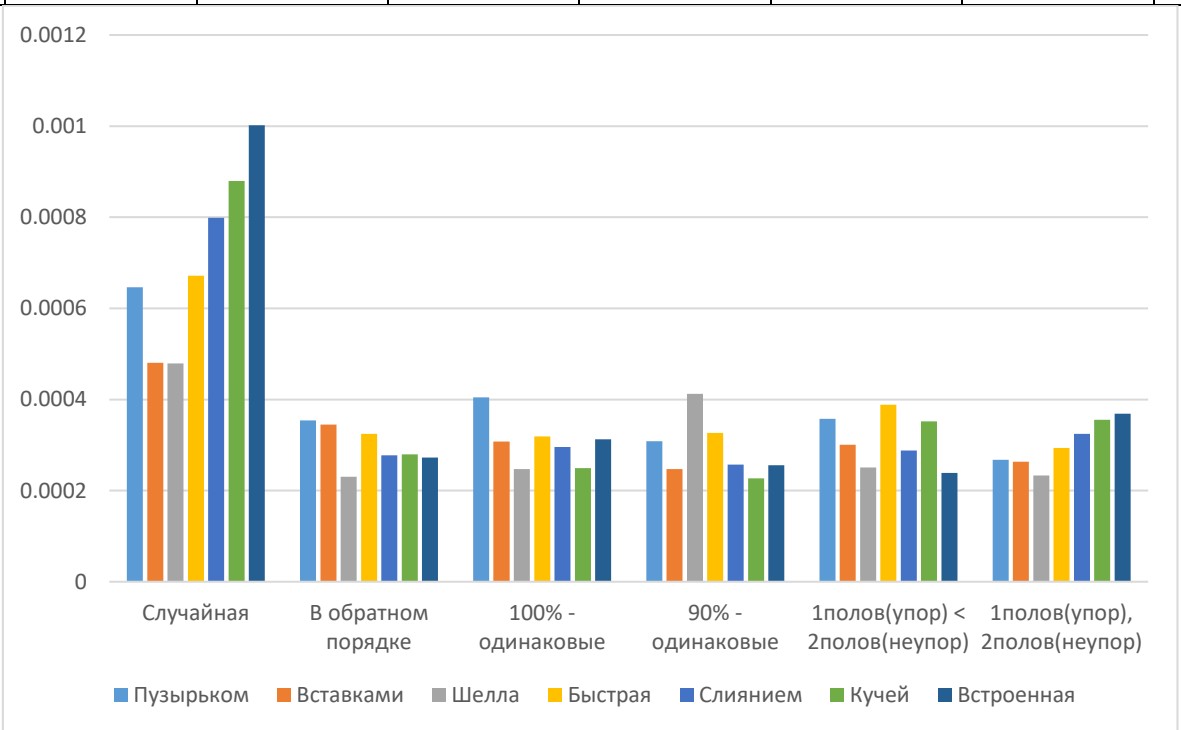
Иногда массив, состоящий из одинаковых элементов, то есть который изначально отсортирован, проходит весь алгоритм сортировки и тратит больше времени, чем массивы, которые не являлись изначально отсортированными и которые проходят другие алгоритмы.

Результаты

Однобайтовый тип данных byte (массивы, состоящие из цифр от 0 до 9).

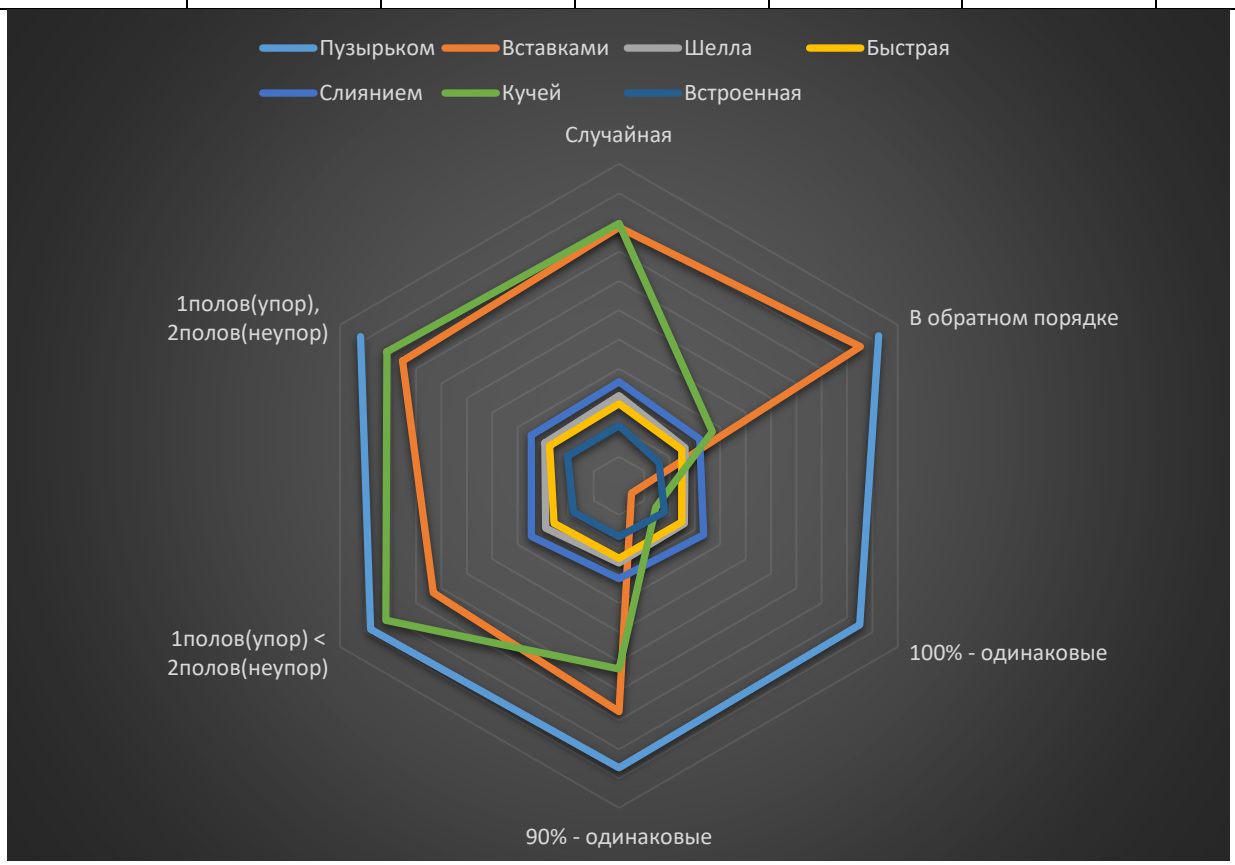
100 элементов

Тип генерации	Пузырьком	Вставками	Шелла	Быстрая	Слиянием	Кучей	Встроенная
Случайная	0,000647	0,00048	0,00048	0,000672	0,000799	0,00088	0,001002
В обратном порядке	0,000354	0,000345	0,00023	0,000325	0,000278	0,00028	0,000272
100% - одинаковые	0,000404	0,000308	0,000248	0,000319	0,000296	0,000249	0,000312
90% - одинаковые	0,000308	0,000248	0,000412	0,000327	0,000257	0,000227	0,000256
1полов(упор) < 2полов(неупор)	0,000358	0,0003	0,000251	0,000388	0,000288	0,000352	0,000239
1полов(упор), 2полов(неупор)	0,000268	0,000264	0,000233	0,000294	0,000325	0,000356	0,000369



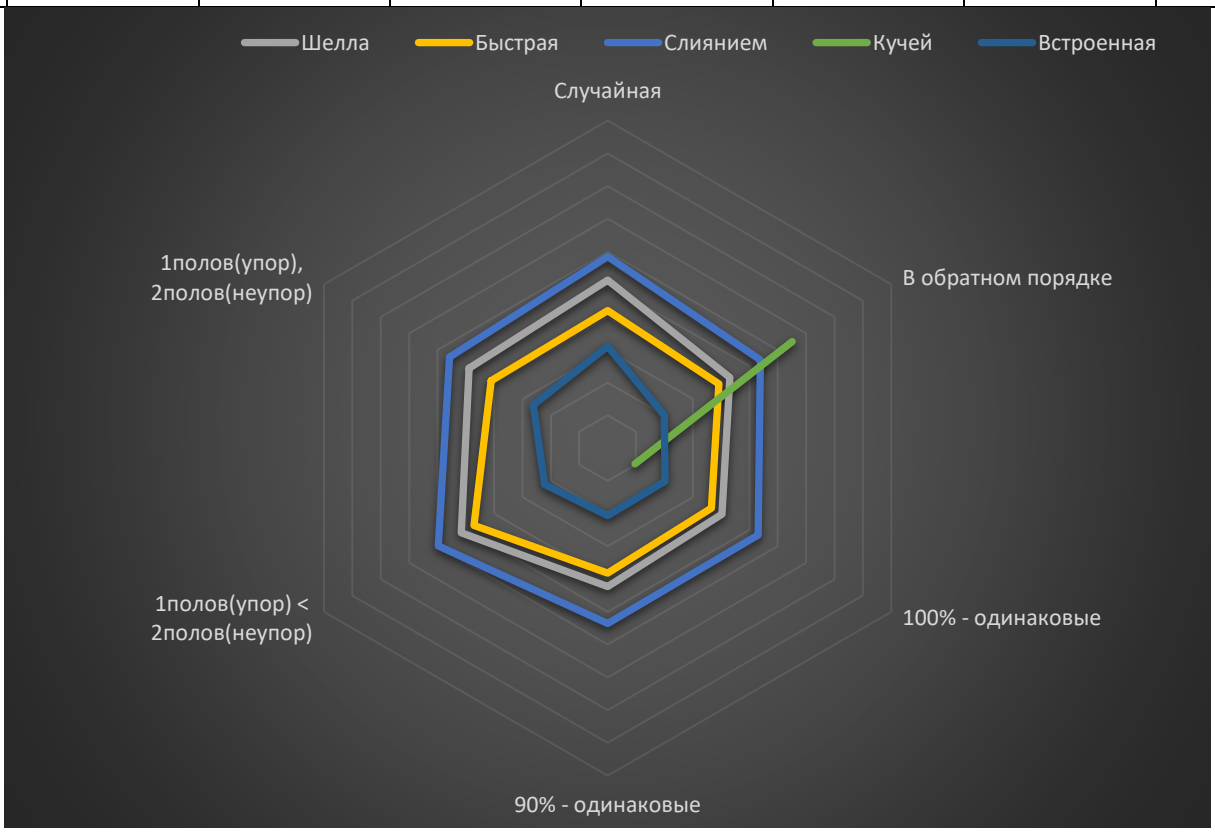
50000 элементов

Тип генерации	Пузырьком	Вставками	Шелла	Быстрая	Слиянием	Кучей	Встроенная
Случайная	> 10	2,288304	0,007298	0,005547	0,011724	2,626026	0,002624
В обратном порядке	9,458954	4,591218	0,004556	0,004014	0,008088	0,013384	0,00164
100% - одинаковые	4,454092	0,000553	0,004424	0,003993	0,009492	0,001438	0,00202
90% - одинаковые	5,111102	0,748229	0,004676	0,003989	0,007988	0,174594	0,001899
1полов(упор) < 2полов(неупор)	5,986918	0,503836	0,006022	0,004318	0,010613	3,297882	0,002004
1полов(упор), 2полов(неупор)	8,957735	1,706818	0,006232	0,005167	0,010605	3,17643	0,002543



500000 элементов

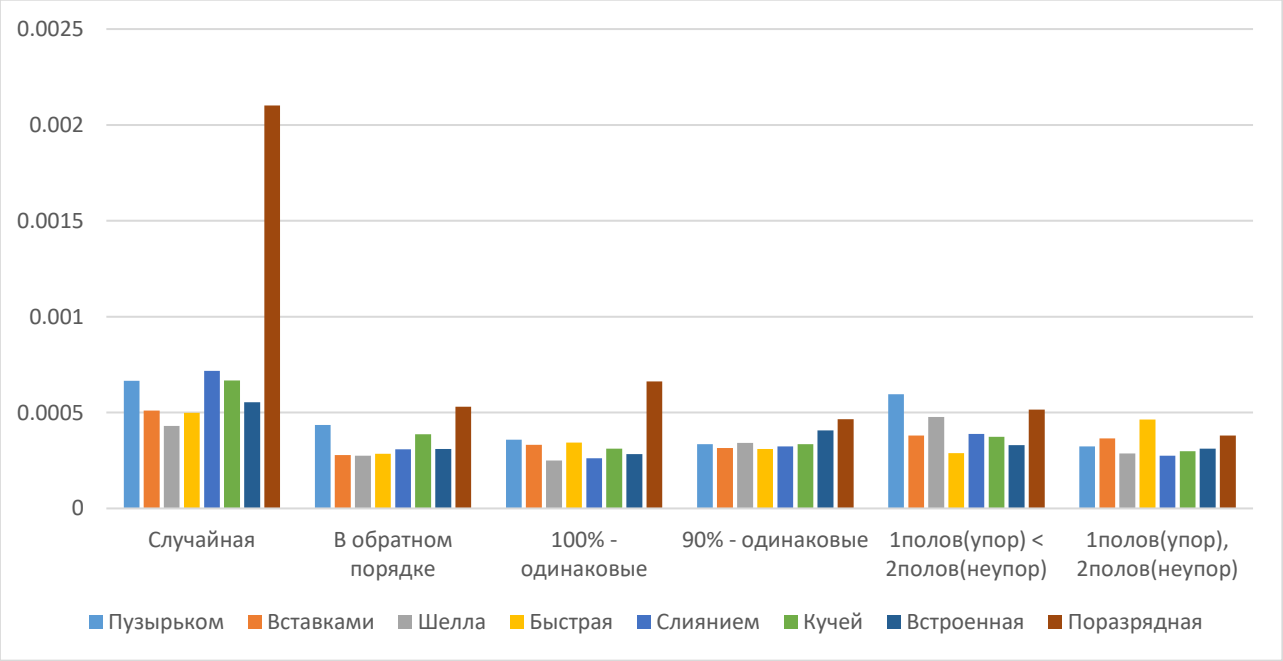
Тип генерации	Пузырьком	Вставками	Шелла	Быстрая	Слиянием	Кучей	Встроенная
Случайная	> 10	> 10	0,087569	0,054984	0,125493	> 10	0,031891
В обратном порядке	> 10	> 10	0,058139	0,047899	0,099545	0,174452	0,018348
100% - одинаковые	> 10	0,002925	0,050662	0,041989	0,095417	0,010908	0,018519
90% - одинаковые	> 10	> 10	0,055782	0,045421	0,097997	> 10	0,018876
1полов(упор) < 2полов(неупор)	> 10	> 10	0,089036	0,071102	0,133449	> 10	0,020417
1полов(упор), 2полов(неупор)	> 10	> 10	0,077577	0,052505	0,109218	> 10	0,024997



Целочисленный тип данных int.

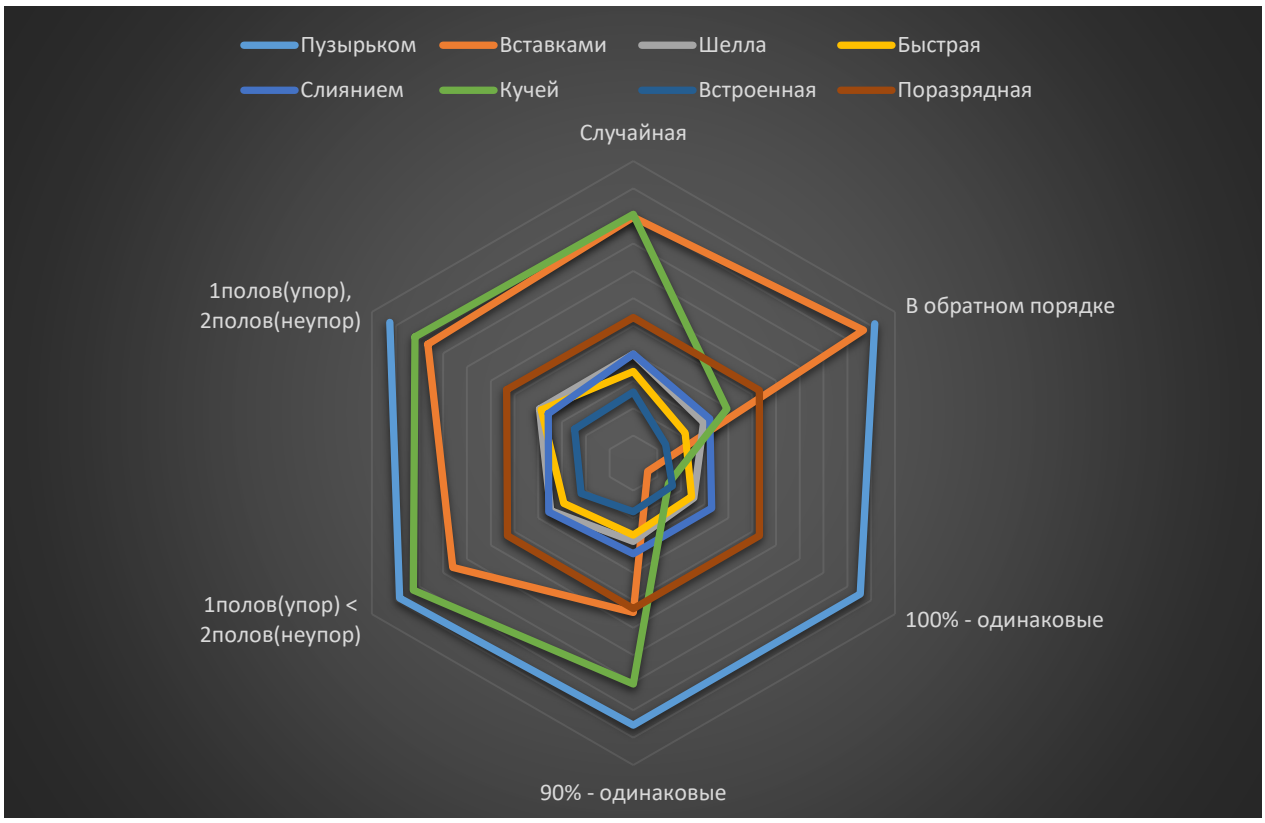
100 элементов

Тип генерации	Пузырьком	Вставками	Шелла	Быстрая	Слиянием	Кучей	Встроенная	Поразрядная
Случайная	0,000665	0,00051	0,00043	0,0005	0,000718	0,000667	0,000554	0,002102
В обратном порядке	0,000435	0,000279	0,000275	0,000284	0,000308	0,000387	0,00031	0,000531
100% - одинаковые	0,000359	0,000332	0,00025	0,000344	0,000262	0,000312	0,000284	0,000663
90% - одинаковые	0,000335	0,000315	0,000341	0,00031	0,000323	0,000336	0,000408	0,000465
1полов(упор) < 2полов(неупор)	0,000596	0,00038	0,000477	0,000289	0,000389	0,000373	0,000331	0,000515
1полов(упор), 2полов(неупор)	0,000324	0,000365	0,000286	0,000464	0,000276	0,000299	0,000312	0,00038



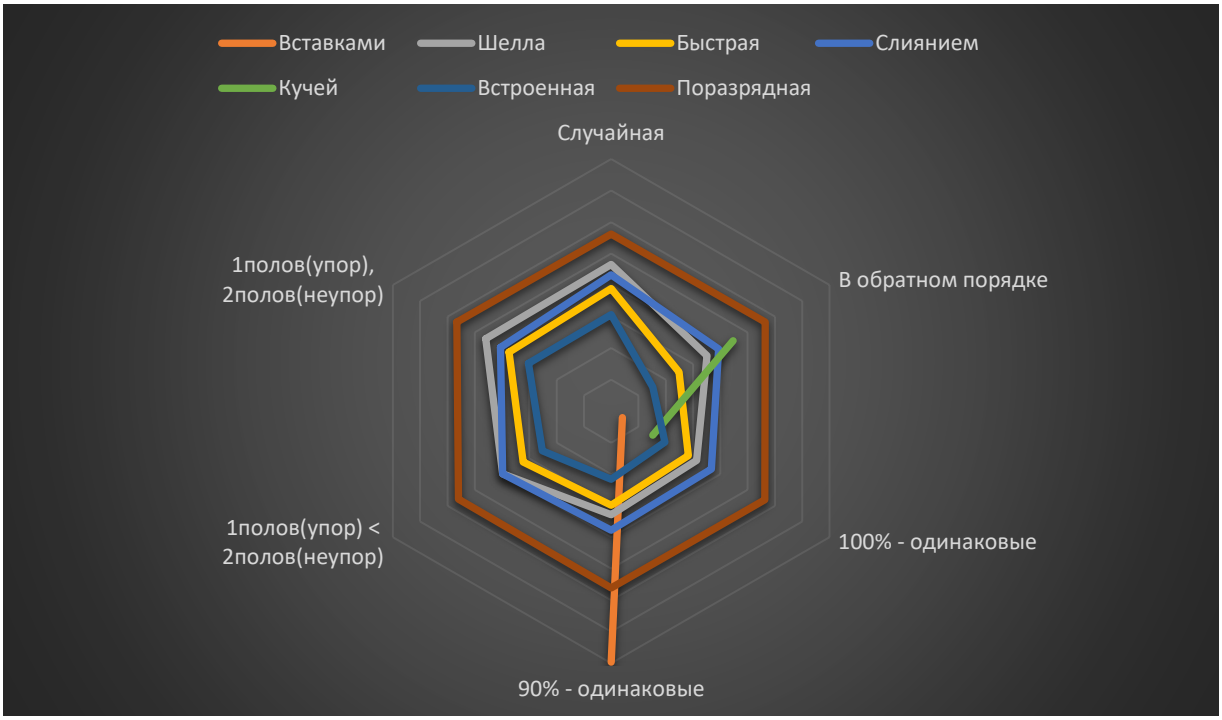
50000 элементов

Тип генерации	Пузырьком	Вставками	Шелла	Быстрая	Слиянием	Кучей	Встроенная	Поразрядная
Случайная	> 10	2,596086	0,017681	0,009421	0,017775	2,883515	0,004458	0,067154
В обратном порядке	8,604134	5,356558	0,00643	0,002975	0,008376	0,017172	0,001294	0,067194
100% - одинаковые	4,688975	0,000608	0,004276	0,003881	0,009021	0,001478	0,001736	0,066992
90% - одинаковые	4,695378	0,07744	0,005817	0,00462	0,009147	1,049407	0,001976	0,067468
1полов(упор) < 2полов(неупор)	6,26724	0,671085	0,010849	0,006278	0,011836	3,5111	0,00304	0,067223
1полов(упор), 2полов(неупор)	9,439527	1,90823	0,017398	0,016147	0,012051	3,265605	0,00398	0,069258



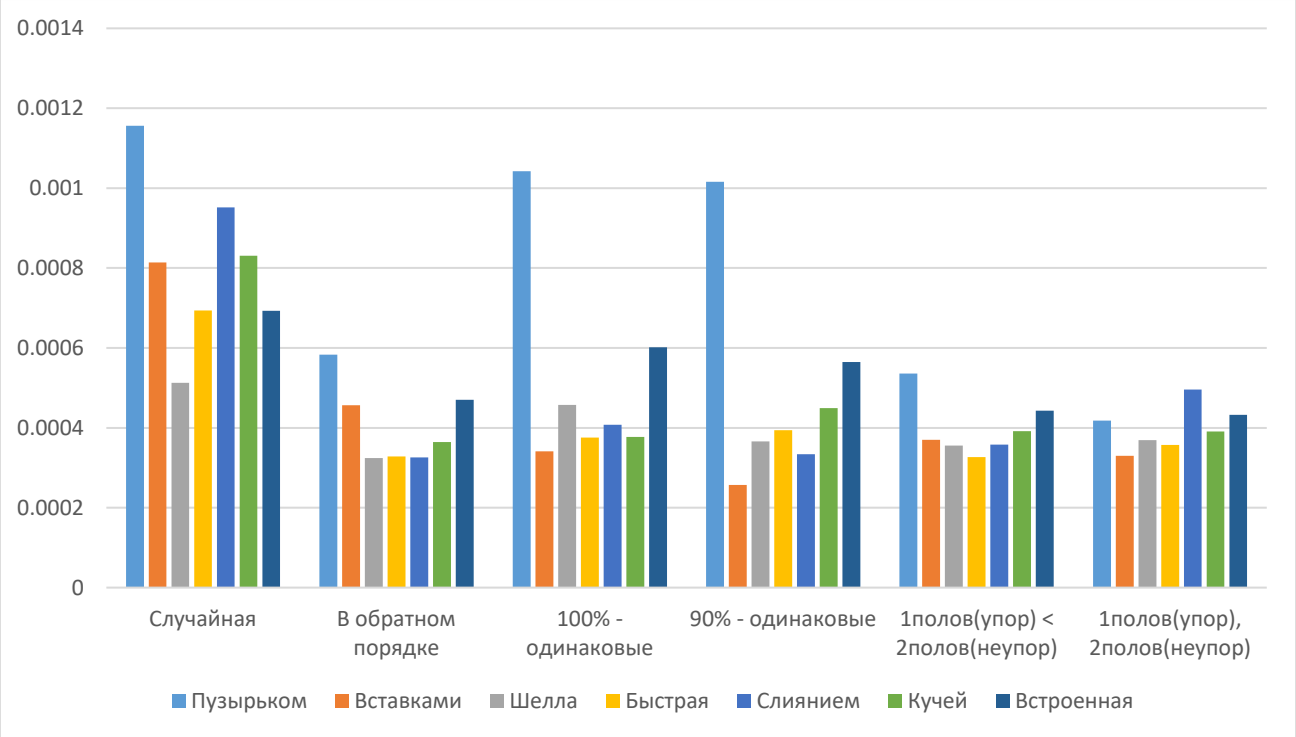
500000 элементов

Тип генерации	Пузырьком	Вставками	Шелла	Быстрая	Слиянием	Кучей	Встроенная	Поразрядная
Случайная	> 10	> 10	0,260435	0,121105	0,187284	> 10	0,053007	0,682856
В обратном порядке	> 10	> 10	0,083041	0,029619	0,127134	0,216214	0,011404	0,700434
100% - одинаковые	> 10	0,003738	0,05687	0,041914	0,097081	0,011387	0,017733	0,691038
90% - одинаковые	> 10	7,122573	0,066259	0,049068	0,108305	> 10	0,0215	0,676756
1полов(упор) < 2полов(неупор)	> 10	> 10	0,131662	0,062509	0,131567	> 10	0,030747	0,664917
1полов(упор), 2полов(неупор)	> 10	> 10	0,245738	0,104684	0,143162	> 10	0,051407	0,708847



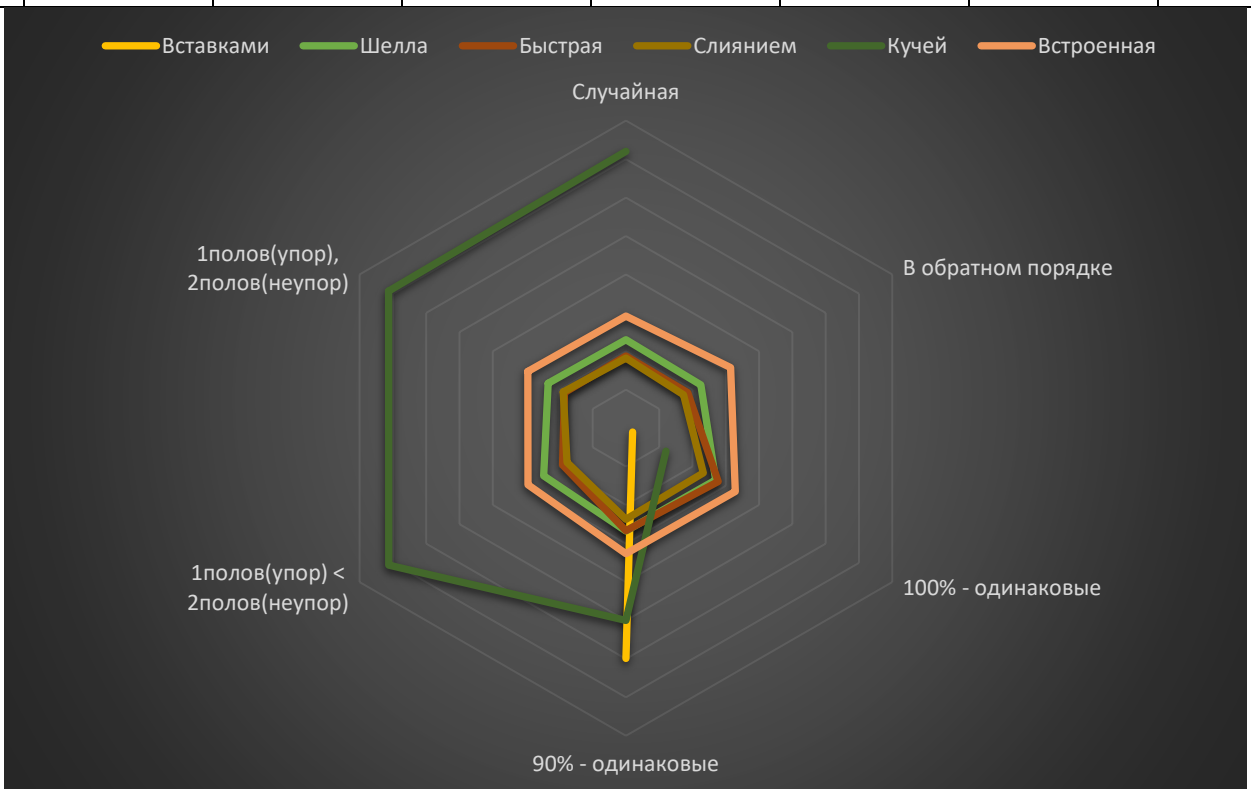
Строковый тип данных string,
100 элементов

Тип генерации	Пузырьком	Вставками	Шелла	Быстрая	Слиянием	Кучей	Встроенная
Случайная	0,001156	0,000814	0,000513	0,000694	0,000952	0,000831	0,000693
В обратном порядке	0,000584	0,000456	0,000324	0,000328	0,000326	0,000364	0,00047
100% - одинаковые	0,001043	0,000341	0,000458	0,000376	0,000408	0,000377	0,000601
90% - одинаковые	0,001016	0,000257	0,000366	0,000394	0,000334	0,00045	0,000564
1полов(упор) < 2полов(неупор)	0,000536	0,00037	0,000356	0,000327	0,000358	0,000392	0,000443
1полов(упор), 2полов(неупор)	0,000418	0,00033	0,000369	0,000357	0,000496	0,000391	0,000432



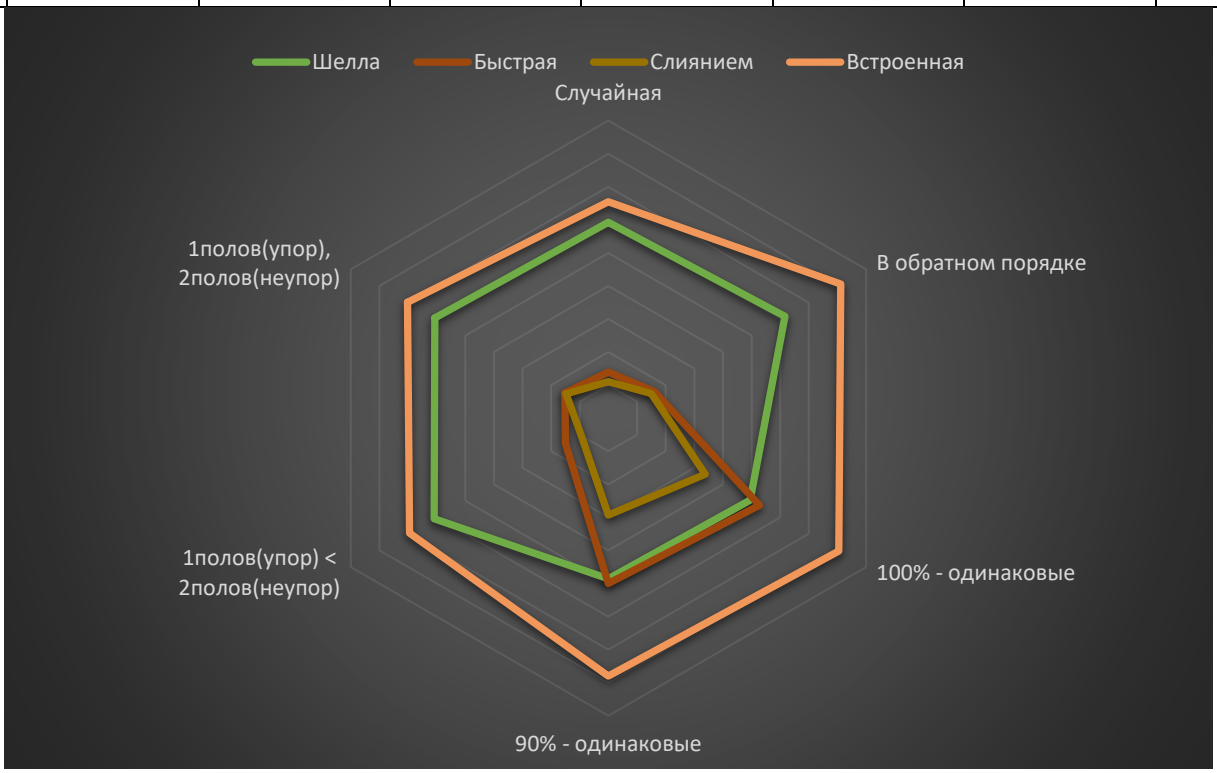
50000 элементов

Тип генерации	Пузырьком	Вставками	Шелла	Быстрая	Слиянием	Кучей	Встроенная
Случайная	> 10	> 10	0,067254	0,044038	0,041688	9,031475	0,124278
В обратном порядке	> 10	> 10	0,064205	0,043172	0,038199	> 10	0,156506
100% - одинаковые	> 10	0,008238	0,099911	0,107688	0,068817	0,022424	0,180025
90% - одинаковые	> 10	2,687485	0,097675	0,096885	0,072093	> 10	0,17617
1полов(упор) < 2полов(неупор)	> 10	> 10	0,079703	0,045184	0,03959	8,359784	0,127249
1полов(упор), 2полов(неупор)	> 10	> 10	0,06965	0,04273	0,044279	8,394726	0,128291



500000 элементов

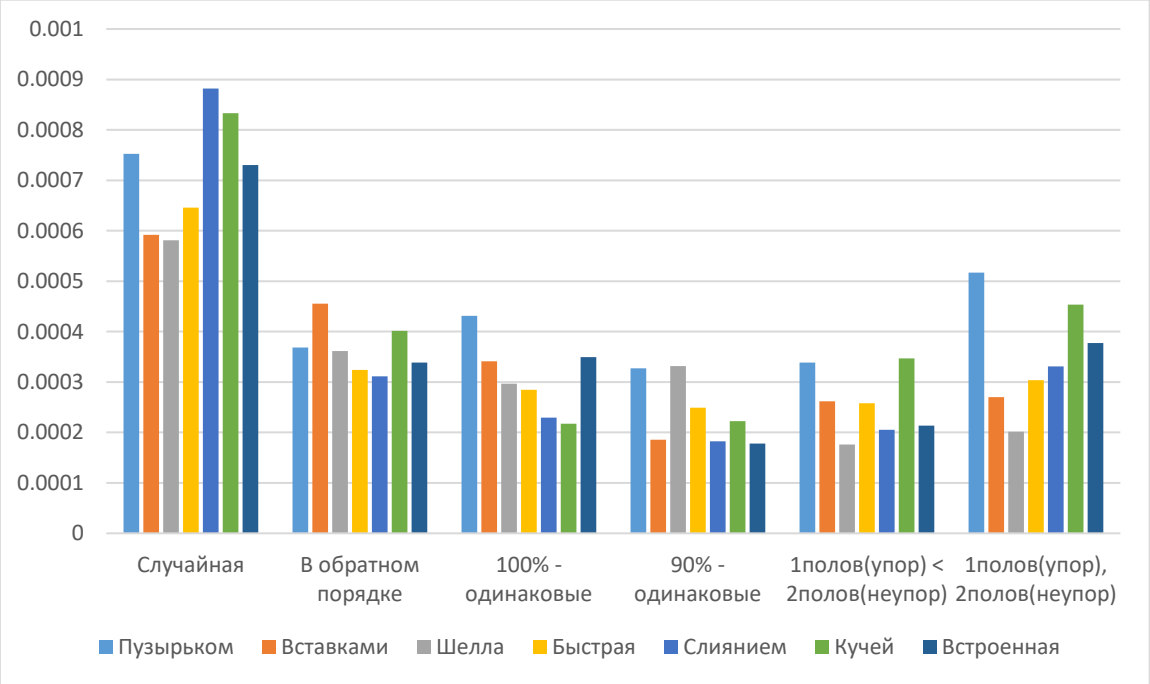
Тип генерации	Пузырьком	Вставками	Шелла	Быстрая	Слиянием	Кучей	Встроенная
Случайная	> 10	> 10	1,469568	0,59387	0,560387	> 10	1,663771
В обратном порядке	> 10	> 10	1,542093	0,619036	0,606115	> 10	2,27692
100% - одинаковые	> 10	0,073785	1,209842	1,290663	0,883688	0,220472	2,245151
90% - одинаковые	> 10	> 10	1,1873	1,221058	0,807323	> 10	2,138132
1полов(упор) < 2полов(неупор)	> 10	> 10	1,51821	0,606865	0,544755	> 10	1,803215
1полов(упор), 2полов(неупор)	> 10	> 10	1,512551	0,608967	0,603967	> 10	1,828814



Тип данных, представляющий собой дату и время, DateTime.

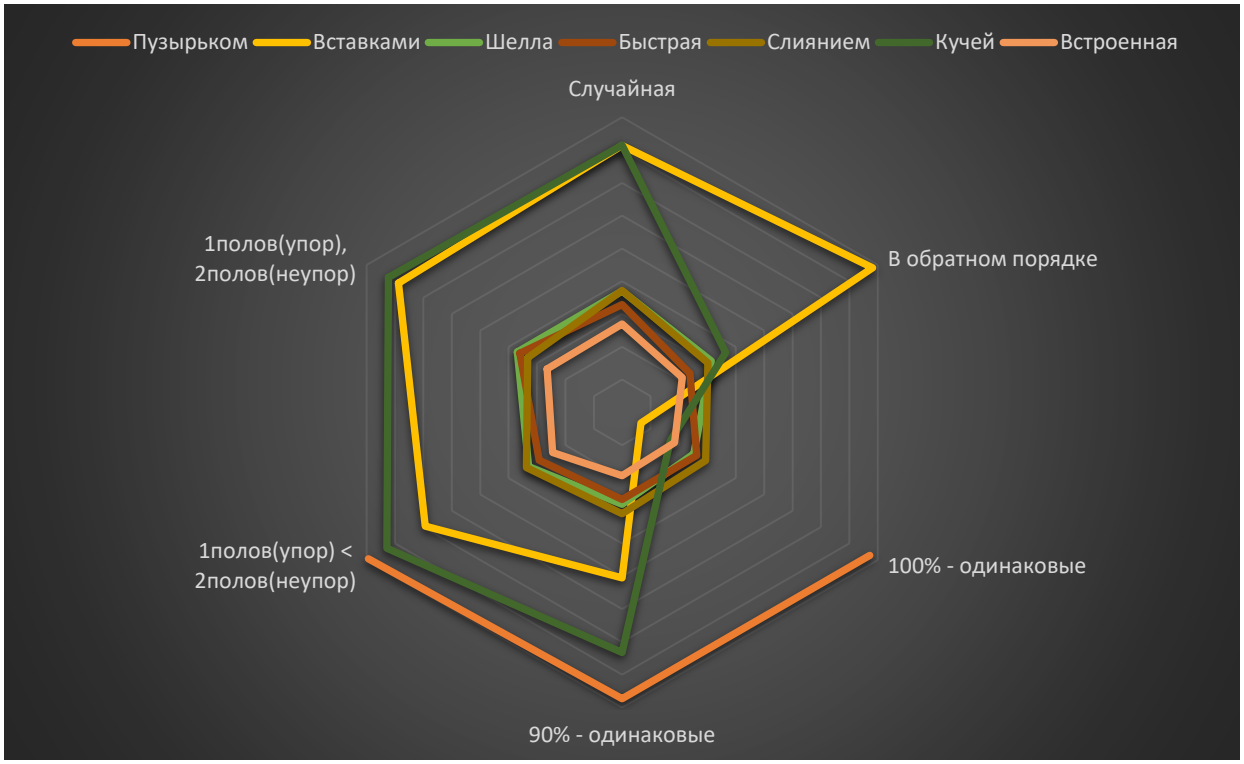
100 элементов

Тип генерации	Пузырьком	Вставками	Шелла	Быстрая	Слиянием	Кучей	Встроенная
Случайная	0,000753	0,000592	0,000581	0,000646	0,000882	0,000833	0,000731
В обратном порядке	0,000368	0,000456	0,000362	0,000324	0,000311	0,000401	0,000339
100% - одинаковые	0,000431	0,000341	0,000297	0,000285	0,000229	0,000217	0,000349
90% - одинаковые	0,000328	0,000186	0,000332	0,000249	0,000182	0,000222	0,000178
1полов(упор) < 2полов(неупор)	0,000339	0,000262	0,000176	0,000258	0,000205	0,000347	0,000214
1полов(упор), 2полов(неупор)	0,000517	0,00027	0,000202	0,000304	0,000331	0,000454	0,000377



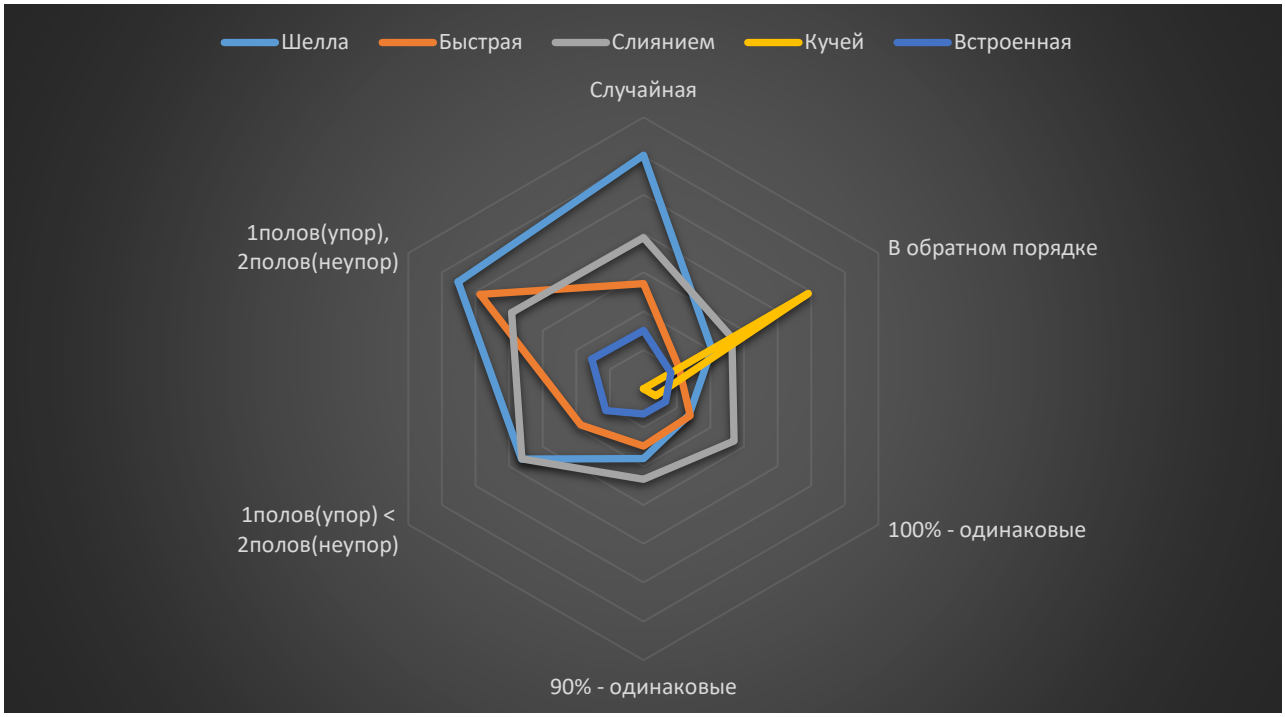
50000 элементов

Тип генерации	Пузырьком	Вставками	Шелла	Быстрая	Слиянием	Кучей	Встроенная
Случайная	> 10	3,716042	0,022424	0,013905	0,022804	3,755178	0,00704
В обратном порядке	> 10	8,130796	0,011724	0,004928	0,010228	0,021032	0,003619
100% - одинаковые	7,245271	0,000674	0,005801	0,006504	0,009372	0,00223	0,002641
90% - одинаковые	7,343324	0,105502	0,007972	0,006724	0,011097	1,449721	0,002917
1полов(упор) < 2полов(неупор)	9,31479	0,931929	0,014052	0,009149	0,015358	4,400994	0,005286
1полов(упор), 2полов(неупор)	> 10	2,754447	0,022228	0,02008	0,014624	4,139858	0,00664



500000 элементов

Тип генерации	Пузырьком	Вставками	Шелла	Быстрая	Слиянием	Кучей	Встроенная
Случайная	> 10	> 10	0,301092	0,135543	0,194774	> 10	0,075251
В обратном порядке	> 10	> 10	0,101871	0,053206	0,131986	0,245629	0,041071
100% - одинаковые	> 10	0,004126	0,067844	0,069687	0,135078	0,018354	0,033203
90% - одинаковые	> 10	> 10	0,090146	0,073963	0,11677	> 10	0,032536
1полов(упор) < 2полов(неупор)	> 10	> 10	0,181432	0,09322	0,181102	> 10	0,056064
1полов(упор), 2полов(неупор)	> 10	> 10	0,275899	0,243068	0,196196	> 10	0,076727



Заключение (или собственная гибридная сортировка)

Выводы

Сортировки пузырьком и вставками могут конкурировать с остальными алгоритмами только на массивах малой длины, но они все равно заметно уступают. Это подтверждает то, что эти алгоритмы имеют сложность $O(n^2)$, что больше сложности других алгоритмов. Алгоритм вставками стоит похвалить за хорошую скорость на всех изначально упорядоченных массивах.

Быстрая сортировка, несмотря на свое название, уступает в скорости некоторым алгоритмам на массивах малой длины. Однако на массивах большой длины оказывается очень эффективной, что соответствует теории, о том, что в среднем её сложность $O(n \log n)$. Худший случай быстрой сортировки со сложностью $O(n^2)$ в ходе данной работы найден не был.

Алгоритм сортировки Шелла, напротив, хорошо смотрится на массивах малой длины.

Неплохие результаты порой показывают сортировки слиянием и кучей. Однако они не всегда отличаются стабильностью, то есть на разных массивах они могут показывать как хорошие, так и плохие результаты.

Поразрядная сортировка выглядит очень слабо на массивах малой длины. Но на массивах средней и большой длины держится ровно по середине между быстрыми и медленными алгоритмами.

Даже встроенная в язык программирования функция сортировки иногда уступает в производительности. Особенно это заметно на строковых массивах. Этот факт подтверждает целесообразность создания собственной гибридной сортировки.

Чтобы составить свою гибридную сортировку, необходимо во всех таблицах данной работы по вертикальным столбцам посчитать средние значения и сортировки с лучшими показателями вставить в гибридную сортировку.

Примерный план гибридной сортировки.
Если на входе массив малой длины, значит упорядочиваем его сортировкой Шелла.

Если массив средней или большой длины, то нужно узнать, какой тип данных он содержит.

Итак, необходимо выбрать лучшую сортировку для строкового типа данных. Очень занимательно наблюдать за противостоянием между быстрой сортировкой и сортировкой слиянием в массивах строкового типа данных

большой и средней длины. По показателям данных экспериментов побеждает сортировка слиянием.

Для средних и длинных массивов дат, целых чисел и однобайтовых типов данных нужно оставить встроенную функцию. В экспериментах от нее всегда совсем на чуть-чуть отстает быстрая сортировка. То есть доработав быструю сортировку, можно получить свою собственную гибридную сортировку, которая в производительности не уступит даже сортировке, встроенной в язык программирования C#.

Приложение

Код программы:

```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
using System.Threading;
using System.Windows.Forms;

namespace Сортировки
{
	public partial class Sort_Form : Form
	{
		#region функции Windows Forms
		public Sort_Form()
		{
			InitializeComponent();
		}

		private void Sort_Form_Load(object sender, EventArgs e)
		{

		}

		private void btn_start_Click(object sender, EventArgs e)
		{
			btn_start.Enabled = false;
			btn_stop.Enabled = true;

			thread_test = new Thread(Testing);

			thread_test.Start();
		}

		private void btn_stop_Click(object sender, EventArgs e)
		{
			btn_stop.Enabled = false;
			if (thread_test != null)
				thread_test.Abort();
			if (thread_sort != null)
				thread_sort.Abort();
			btn_start.Enabled = true;
		}

		private void Sort_Form_FormClosing(object sender, FormClosingEventArgs e)
		{
			// При закрытии формы, закрываем потоки
			if (thread_test != null)
				thread_test.Abort();
			if (thread_sort != null)
				thread_sort.Abort();
		}
		#endregion

		#region Глобальные переменные
		public delegate int [ ] deleg_gener_integers(int numeric_for_random, int array_length);
		public delegate void deleg_sort_integers(int[] array);

		public delegate byte [ ] deleg_gener_digits(int numeric_for_random, int array_length);
		public delegate void deleg_sort_digits(byte[] array);

		public delegate string [ ] deleg_gener_strings(int numeric_for_random, int array_length);
		public delegate void deleg_sort_strings(string [ ] array);

		public delegate DateTime [ ] deleg_gener_dates(int numeric_for_random, int array_length);
		public delegate void deleg_sort_dates(DateTime [ ] array);

		int max_waiting_sec = 10;
		int amount_attempts = 3;

		Stopwatch stopwatch; // секундомер
		Thread thread_sort; // Поток для каждой сортировки
		Thread thread_test; // Поток для общего теста
```

```
		#endregion

		#region Самая главная функция
		private void Testing()
		{
			int [] length_array = {100, 50000, 500000};
			int stroka = -1;
			#region Цифры
			deleg_gener_digits [ ] all_func_gener_dig = { get_random_digits_array, get_reverse_sort_digits_array,
get_identical_100_digits_array, get_identical_90_digits_array, get_sort1_and_2more_then1_digits_array, get_sort1_and_unsort_digits_array
};
			// Без глупой
			deleg_sort_digits [ ] all_func_sort_dig = { sort_bubble_digits, sort_insert_digits, sort_shell_digits,
sort_quick_digits, sort_merge_digits, sort_heap_digits, sort_built_in_digits };

			// по длиннам массивов
			foreach (int len in length_array)
			{
				// Прописать названия столбцов
				stroka++;
				int stolbec = 0;
				dataGridView.Rows.Add();
				dataGridView.Rows [stroka].Cells [stolbec++].Value = "Длина";
				dataGridView.Rows [stroka].Cells [stolbec++].Value = "Тип генерации";
				for (; stolbec < all_func_gener_dig.Length + 4; stolbec++)
					dataGridView.Rows [stroka].Cells [stolbec].Value = get_name_sort( stolbec - 2 );
				// по функциям генераций
				int index_func = 0;
				foreach (deleg_gener_digits func_gen in all_func_gener_dig)
				{
					stroka++;
					stolbec = 0;
					dataGridView.Rows.Add();
					dataGridView.Rows [stroka].Cells [stolbec++].Value = len;
					dataGridView.Rows [stroka].Cells [stolbec++].Value =
get_name_generation(index_func++ % all_func_gener_dig.Length);
					byte [ ] array = func_gen(10, len);

					// по сортировкам
					foreach (deleg_sort_digits func_sort in all_func_sort_dig)
					{
						byte [ ] array_for_sort = new byte [array.Length];
						Array.Copy(array, array_for_sort, array.Length);

						double sum = 0.0;
						// по количеству испытаний
						for (int i = 0; i < amount_attempts; i++)
						{
							thread_sort = new Thread(delegate () { func_sort(array_for_sort);
});
							stopwatch = new Stopwatch(); // секундомер
							stopwatch.Start(); // Запуск секундомера
							thread_sort.Start();
							thread_sort.Join(new TimeSpan(0, 0, max_waiting_sec));// Ждем
не дольше __ секунд
							thread_sort.Abort();
							stopwatch.Stop(); // Останавливаем секундомер

							if (stopwatch.Elapsed.TotalSeconds > max_waiting_sec + 1) // Если
намного дольше, чем максимальная граница, то больше 1 попытки не делаем
							{
								sum = amount_attempts *
stopwatch.Elapsed.TotalSeconds;
								break;
							}
							else
								sum += stopwatch.Elapsed.TotalSeconds;
						}
						double mean_time = sum / amount_attempts; // Среднее время
						if (mean_time > max_waiting_sec)
							dataGridView.Rows [stroka].Cells [stolbec++].Value = "> " +
max_waiting_sec;
						else
							dataGridView.Rows [stroka].Cells [stolbec++].Value = mean_time;
```

```
                                        }
                                }
                        }
                        #endregion
                        #region Числа

                        deleg_gener_integers [ ] all_func_gener_int = {get_random_integers_array, get_reverse_sort_integers_array,
get_identical_100_integers_array, get_identical_90_integers_array, get_sort1_and_2more_then1_integers_array,
get_sort1_and_unsort_integers_array };

                        // Без глупой
                        deleg_sort_integers[] all_func_sort_int = {sort_bubble_integers, sort_insert_integers, sort_shell_integers,
sort_quick_integers, sort_merge_integers, sort_heap_integers, sort_built_in_integers, sort_bitwise_integers };

                        // по длиннам массивов
                        foreach (int len in length_array)
                        {
                                // Прописать названия столбцов
                                stroka++;
                                int stolbec = 0;
                                dataGridView.Rows.Add();
                                dataGridView.Rows [stroka].Cells [stolbec++].Value = "Длина";
                                dataGridView.Rows [stroka].Cells [stolbec++].Value = "Тип генерации";
                                for (; stolbec < all_func_gener_int.Length + 4; stolbec++)
                                        dataGridView.Rows [stroka].Cells [stolbec].Value = get_name_sort( stolbec - 2 );
                                // по функциям генераций
                                int index_func = 0;
                                foreach(deleg_gener_integers func_gen in all_func_gener_int)
                                {
                                        stroka++;
                                        stolbec=0;
                                        dataGridView.Rows.Add();
                                        dataGridView.Rows [stroka].Cells [stolbec++].Value = len;
                                        dataGridView.Rows [stroka].Cells [stolbec++].Value =
get_name_generation(index_func++ % all_func_gener_int.Length);
                                        int [ ] array = func_gen(10, len);

                                        // по сортировкам
                                        foreach (deleg_sort_integers func_sort in all_func_sort_int)
                                        {
                                                int [ ] array_for_sort = new int[array.Length];
                                                Array.Copy(array, array_for_sort, array.Length);

                                                double sum = 0.0;
                                                // по количеству испытаний
                                                for (int i = 0; i < amount_attempts; i++)
                                                {
                                                        thread_sort = new Thread(delegate () { func_sort(array_for_sort);
});
                                                        stopwatch = new Stopwatch(); // секундомер
                                                        stopwatch.Start(); // Запуск секундомера
                                                        thread_sort.Start();
                                                        thread_sort.Join(new TimeSpan(0, 0, max_waiting_sec));// Ждем
не дольше __ секунд
                                                        thread_sort.Abort();
                                                        stopwatch.Stop(); // Останавливаем секундомер

                                                        if (stopwatch.Elapsed.TotalSeconds > max_waiting_sec + 1) // Если
намного дольше, чем максимальная граница, то больше 1 попытки не делаем
                                                        {
                                                                sum = amount_attempts *
stopwatch.Elapsed.TotalSeconds;
                                                                break;
                                                        }
                                                        else
                                                                sum += stopwatch.Elapsed.TotalSeconds;
                                                }
                                                double mean_time = sum / amount_attempts; // Среднее время
                                                if (mean_time > max_waiting_sec)
                                                        dataGridView.Rows [stroka].Cells [stolbec++].Value = "> " +
max_waiting_sec;
                                                else
                                                        dataGridView.Rows [stroka].Cells [stolbec++].Value = mean_time;
                                        }
```

```
                }
            }
            #endregion
            #region Строки
            deleg_gener_strings [ ] all_func_gener_string = { get_random_string_array, get_reverse_sort_string_array,
get_identical_100_string_array, get_identical_90_string_array, get_sort1_and_2more_then1_string_array, get_sort1_and_unsort_string_array
};

            // Без глупой
            deleg_sort_strings [ ] all_func_sort_string = { sort_bubble_strings, sort_insert_strings, sort_shell_strings,
sort_quick_strings, sort_merge_strings, sort_heap_strings, sort_built_in_strings};

            // по длиннам массивов
            foreach (int len in length_array)
            {
                // Прописать названия столбцов
                stroka++;
                int stolbec = 0;
                dataGridView.Rows.Add();
                dataGridView.Rows [stroka].Cells [stolbec++].Value = "Длина";
                dataGridView.Rows [stroka].Cells [stolbec++].Value = "Тип генерации";
                for (; stolbec < all_func_gener_string.Length + 4; stolbec++)
                    dataGridView.Rows [stroka].Cells [stolbec].Value = get_name_sort( stolbec - 2 );
                // по функциям генераций
                int index_func = 0;
                foreach (deleg_gener_strings func_gen in all_func_gener_string)
                {
                    stroka++;
                    stolbec = 0;
                    dataGridView.Rows.Add();
                    dataGridView.Rows [stroka].Cells [stolbec++].Value = len;
                    dataGridView.Rows [stroka].Cells [stolbec++].Value =
get_name_generation(index_func++ % all_func_gener_string.Length);
                    string [ ] array = func_gen(10, len);

                    // по сортировкам
                    foreach (deleg_sort_strings func_sort in all_func_sort_string)
                    {

                        string [ ] array_for_sort = new string [array.Length];
                        Array.Copy(array, array_for_sort, array.Length);

                        double sum = 0.0;
                        // по количеству испытаний
                        for (int i = 0; i < amount_attempts; i++)
                        {
                            thread_sort = new Thread(delegate () { func_sort(array_for_sort);
});
                            stopwatch = new Stopwatch(); // секундомер
                            stopwatch.Start(); // Запуск секундомера
                            thread_sort.Start();
                            thread_sort.Join(new TimeSpan(0, 0, max_waiting_sec));// Ждем
не дольше __ секунд
                            thread_sort.Abort();
                            stopwatch.Stop(); // Останавливаем секундомер

                            if (stopwatch.Elapsed.TotalSeconds > max_waiting_sec + 1) // Если
намного дольше, чем максимальная граница, то больше 1 попытки не делаем
                            {
                                sum = amount_attempts *
stopwatch.Elapsed.TotalSeconds;
                                break;
                            }
                            else
                                sum += stopwatch.Elapsed.TotalSeconds;

                        }
                        double mean_time = sum / amount_attempts; // Среднее время
                        if (mean_time > max_waiting_sec)
                            dataGridView.Rows [stroka].Cells [stolbec++].Value = "> " +
max_waiting_sec;
                        else
                            dataGridView.Rows [stroka].Cells [stolbec++].Value = mean_time;
                    }
```

```
                }
            }
            #endregion
            #region Даты
            deleg_gener_dates [ ] all_func_gener_dates = { get_random_date_array, get_reverse_sort_date_array,
get_identical_100_date_array, get_identical_90_date_array, get_sort1_and_2more_then1_date_array, get_sort1_and_unsort_date_array };
            // Без глупой
            deleg_sort_dates [ ] all_func_sort_dates = { sort_bubble_dates, sort_insert_dates, sort_shell_dates,
sort_quick_dates, sort_merge_dates, sort_heap_dates, sort_built_in_dates };

            // по длиннам массивов
            foreach (int len in length_array)
            {
                // Прописать названия столбцов
                stroka++;
                int stolbec = 0;
                dataGridView.Rows.Add();
                dataGridView.Rows [stroka].Cells [stolbec++].Value = "Длина";
                dataGridView.Rows [stroka].Cells [stolbec++].Value = "Тип генерации";
                for (; stolbec < all_func_gener_dates.Length + 4; stolbec++)
                    dataGridView.Rows [stroka].Cells [stolbec].Value = get_name_sort(stolbec - 2);
                // по функциям генераций
                int index_func = 0;
                foreach (deleg_gener_dates func_gen in all_func_gener_dates)
                {
                    stroka++;
                    stolbec = 0;
                    dataGridView.Rows.Add();
                    dataGridView.Rows [stroka].Cells [stolbec++].Value = len;
                    dataGridView.Rows [stroka].Cells [stolbec++].Value =
get_name_generation(index_func++ % all_func_gener_dates.Length);
                    DateTime [ ] array = func_gen(10, len);

                    // по сортировкам
                    foreach (deleg_sort_dates func_sort in all_func_sort_dates)
                    {
                        DateTime [ ] array_for_sort = new DateTime [array.Length];
                        Array.Copy(array, array_for_sort, array.Length);

                        double sum = 0.0;
                        // по количеству испытаний
                        for (int i = 0; i < amount_attempts; i++)
                        {
                            thread_sort = new Thread(delegate () { func_sort(array_for_sort);
});
                            stopwatch = new Stopwatch(); // секундомер
                            stopwatch.Start(); // Запуск секундомера
                            thread_sort.Start();
                            thread_sort.Join(new TimeSpan(0, 0, max_waiting_sec));// Ждем
не дольше __ секунд
                            thread_sort.Abort();
                            stopwatch.Stop(); // Останавливаем секундомер

                            if (stopwatch.Elapsed.TotalSeconds > max_waiting_sec + 1) // Если
намного дольше, чем максимальная граница, то больше 1 попытки не делаем
                            {
                                sum = amount_attempts *
stopwatch.Elapsed.TotalSeconds;
                                break;
                            }
                            else
                                sum += stopwatch.Elapsed.TotalSeconds;
                        }
                        double mean_time = sum / amount_attempts; // Среднее время
                        if (mean_time > max_waiting_sec)
                            dataGridView.Rows [stroka].Cells [stolbec++].Value = "> " +
max_waiting_sec;
                        else
                            dataGridView.Rows [stroka].Cells [stolbec++].Value = mean_time;
                    }
                }
            }
            #endregion
            btn_start.Enabled = true;
```

```
		btn_stop.Enabled = false;
}
#endregion

#region Сортировки
#region Глупая сортировка
private static void sort_stupid_integers(int [ ] array)
{
		for (int i = 0; i < array.Length - 1;)
		{
				if (array [i + 1] < array [i])
				{
						var temp = array [i];
						array [i] = array [i + 1];
						array [i + 1] = temp;
						i = 0;
				}
				else
						i++;
		}
}
private static void sort_stupid_digits(byte [ ] array)
{
		for (int i = 0; i < array.Length - 1;)
		{
				if (array [i + 1] < array [i])
				{
						var temp = array [i];
						array [i] = array [i + 1];
						array [i + 1] = temp;
						i = 0;
				}
				else
						i++;
		}
}
private static void sort_stupid_strings(string [ ] array)
{
		for (int i = 0; i < array.Length - 1;)
		{
				if (e1_less_then_e2(array [i + 1], array [i]))
				{
						var temp = array [i];
						array [i] = array [i + 1];
						array [i + 1] = temp;
						i = 0;
				}
				else
						i++;
		}
}
private static void sort_stupid_dates(DateTime [ ] array)
{
		for (int i = 0; i < array.Length - 1;)
		{
				if (array [i + 1] < array [i])
				{
						var temp = array [i];
						array [i] = array [i + 1];
						array [i + 1] = temp;
						i = 0;
				}
				else
						i++;
		}
}
#endregion
		#region Пузырьком
private static void sort_bubble_integers(int [ ] array)
{
		for (int i = 0; i < array.Length - 1; i++)
				for (int j = 0; j < array.Length - i - 1; j++)
						if (array [j + 1] < array [j])
						{
```

```
                                        var temp = array [j + 1];
                                        array [j + 1] = array [j];
                                        array [j] = temp;
                                }
}
private static void sort_bubble_digits(byte [ ] array)
{
        for (int i = 0; i < array.Length - 1; i++)
                for (int j = 0; j < array.Length - i - 1; j++)
                        if (array [j + 1] < array [j])
                        {
                                var temp = array [j + 1];
                                array [j + 1] = array [j];
                                array [j] = temp;
                        }
}
private static void sort_bubble_strings(string [ ] array)
{
        for (int i = 0; i < array.Length - 1; i++)
                for (int j = 0; j < array.Length - i - 1; j++)
                        if (e1_less_then_e2(array [j + 1], array [j]))
                        {
                                var temp = array [j + 1];
                                array [j + 1] = array [j];
                                array [j] = temp;
                        }
}
private static void sort_bubble_dates(DateTime [ ] array)
{
        for (int i = 0; i < array.Length - 1; i++)
                for (int j = 0; j < array.Length - i - 1; j++)
                        if (array [j + 1] < array [j])
                        {
                                var temp = array [j + 1];
                                array [j + 1] = array [j];
                                array [j] = temp;
                        }
}
#endregion
        #region Вставками
        private static void sort_insert_integers(int [ ] array)
        {
                for (int i = 1; i < array.Length; i++)
                {
                        var x = array [i];
                        int j = i;
                        for (; j >= 1 && array [j - 1] > x; j--)
                                array [j] = array [j - 1];
                        array [j] = x;
                }
        }
        private static void sort_insert_digits(byte [ ] array)
        {
                for (int i = 1; i < array.Length; i++)
                {
                        var x = array [i];
                        int j = i;
                        for (; j >= 1 && array [j - 1] > x; j--)
                                array [j] = array [j - 1];
                        array [j] = x;
                }
        }
        private static void sort_insert_strings(string [ ] array)
        {
                for (int i = 1; i < array.Length; i++)
                {
                        var x = array [i];
                        int j = i;
                        for (; j >= 1 && e1_less_then_e2(x, array [j - 1]); j--)
                                array [j] = array [j - 1];
                        array [j] = x;
                }
        }
        private static void sort_insert_dates(DateTime [ ] array)
```

```
		{
			for (int i = 1; i < array.Length; i++)
			{
				var x = array [i];
				int j = i;
				for (; j >= 1 && array [j - 1] > x; j--)
					array [j] = array [j - 1];
				array [j] = x;
			}
		}
		#endregion
		#region Шелла
private static void sort_shell_integers(int [ ] array)
{
		for (int k = array.Length / 2; k > 0; k /= 2)
			for (int i = k; i < array.Length; i++)
			{
				var t = array [i];
				int j;
				for (j = i; j >= k; j -= k)
				{
					if (t < array [j - k])
						array [j] = array [j - k];
					else
						break;
				}
				array [j] = t;
			}
}
private static void sort_shell_digits(byte [ ] array)
{
		for (int k = array.Length / 2; k > 0; k /= 2)
			for (int i = k; i < array.Length; i++)
			{
				var t = array [i];
				int j;
				for (j = i; j >= k; j -= k)
				{
					if (t < array [j - k])
						array [j] = array [j - k];
					else
						break;
				}
				array [j] = t;
			}
}
private static void sort_shell_strings(string [ ] array)
{
		for (int k = array.Length / 2; k > 0; k /= 2)
			for (int i = k; i < array.Length; i++)
			{
				var t = array [i];
				int j;
				for (j = i; j >= k; j -= k)
				{
					if (e1_less_then_e2(t, array [j - k]))
						array [j] = array [j - k];
					else
						break;
				}
				array [j] = t;
			}
}
private static void sort_shell_dates(DateTime [ ] array)
{
		for (int k = array.Length / 2; k > 0; k /= 2)
			for (int i = k; i < array.Length; i++)
			{
				var t = array [i];
				int j;
				for (j = i; j >= k; j -= k)
				{
					if (t < array [j - k])
						array [j] = array [j - k];
```

```
                                        else
                                                break;
                                }
                                array [j] = t;
                        }
}
#endregion
        #region Быстрая сортировка
private static void sort_quick_integers(int [ ] array)
{
        sort_quick_integers(array, 0, array.Length - 1);
}
private static void sort_quick_integers(int [ ] array, int first, int last)
{
        if (first < last)
        {
                int left = first, right = last;
                var middle = array [( right + left ) / 2];
                do
                {
                        while (array [left] < middle)
                                left++;
                        while (array [right] > middle)
                                right--;
                        if (left <= right)
                        {
                                var temp = array [left];
                                array [left] = array [right];
                                array [right] = temp;
                                left++;
                                right--;
                        }
                } while (left <= right);
                sort_quick_integers(array, first, right);
                sort_quick_integers(array, left, last);
        }
}

private static void sort_quick_digits(byte [ ] array)
{
        sort_quick_digits(array, 0, array.Length - 1);
}
private static void sort_quick_digits(byte [ ] array, int first, int last)
{
        if (first < last)
        {
                int left = first, right = last;
                var middle = array [( right + left ) / 2];
                do
                {
                        while (array [left] < middle)
                                left++;
                        while (array [right] > middle)
                                right--;
                        if (left <= right)
                        {
                                var temp = array [left];
                                array [left] = array [right];
                                array [right] = temp;
                                left++;
                                right--;
                        }
                } while (left <= right);
                sort_quick_digits(array, first, right);
                sort_quick_digits(array, left, last);
        }
}

private static void sort_quick_strings(string [ ] array)
{
        sort_quick_strings(array, 0, array.Length - 1);
}
private static void sort_quick_strings(string [ ] array, int first, int last)
{
```

```
		if (first < last)
		{
			int left = first, right = last;
			var middle = array [( right + left ) / 2];
			do
			{
				while (e1_less_then_e2(array [left], middle))
					left++;
				while (e1_less_then_e2(middle, array [right]))
				{
					right--;
				}
				if (left <= right)
				{
					var temp = array [left];
					array [left] = array [right];
					array [right] = temp;
					left++;
					right--;
				}
			} while (left <= right);
			sort_quick_strings(array, first, right);
			sort_quick_strings(array, left, last);
		}
}

private static void sort_quick_dates(DateTime [ ] array)
{
		sort_quick_dates(array, 0, array.Length - 1);
}
private static void sort_quick_dates(DateTime [ ] array, int first, int last)
{
		if (first < last)
		{
			int left = first, right = last;
			var middle = array [( right + left ) / 2];
			do
			{
				while (array [left] < middle)
					left++;
				while (array [right] > middle)
					right--;
				if (left <= right)
				{
					var temp = array [left];
					array [left] = array [right];
					array [right] = temp;
					left++;
					right--;
				}
			} while (left <= right);
			sort_quick_dates(array, first, right);
			sort_quick_dates(array, left, last);
		}
}
#endregion
		#region Сортировка слиянием
private static void sort_merge_integers(int [ ] array)
{
		sort_merge_integers(array, 0, array.Length - 1);
}
private static void sort_merge_integers(int [ ] array, int lowIndex, int highIndex)
{
		if (lowIndex < highIndex)
		{
			var middleIndex = ( lowIndex + highIndex ) / 2;
			sort_merge_integers(array, lowIndex, middleIndex);
			sort_merge_integers(array, middleIndex + 1, highIndex);
			Merge(array, lowIndex, middleIndex, highIndex);
		}
}
private static void Merge(int [ ] array, int lowIndex, int middleIndex, int highIndex)
{
		var left = lowIndex;
```

```
		var right = middleIndex + 1;
		var tempArray = new int [highIndex - lowIndex + 1];
		var index = 0;

		while (( left <= middleIndex ) && ( right <= highIndex ))
		{
			if (array [left] < array [right])
			{
				tempArray [index] = array [left];
				left++;
			}
			else
			{
				tempArray [index] = array [right];
				right++;
			}

			index++;
		}

		for (var i = left; i <= middleIndex; i++)
		{
			tempArray [index] = array [i];
			index++;
		}

		for (var i = right; i <= highIndex; i++)
		{
			tempArray [index] = array [i];
			index++;
		}

		for (var i = 0; i < tempArray.Length; i++)
			array [lowIndex + i] = tempArray [i];
}

private static void sort_merge_digits(byte [ ] array)
{
		sort_merge_digits(array, 0, array.Length - 1);
}
private static void sort_merge_digits(byte [ ] array, int lowIndex, int highIndex)
{
		if (lowIndex < highIndex)
		{
			var middleIndex = ( lowIndex + highIndex ) / 2;
			sort_merge_digits(array, lowIndex, middleIndex);
			sort_merge_digits(array, middleIndex + 1, highIndex);
			Merge(array, lowIndex, middleIndex, highIndex);
		}
}
private static void Merge(byte [ ] array, int lowIndex, int middleIndex, int highIndex)
{
		var left = lowIndex;
		var right = middleIndex + 1;
		var tempArray = new byte [highIndex - lowIndex + 1];
		var index = 0;

		while (( left <= middleIndex ) && ( right <= highIndex ))
		{
			if (array [left] < array [right])
			{
				tempArray [index] = array [left];
				left++;
			}
			else
			{
				tempArray [index] = array [right];
				right++;
			}

			index++;
		}

		for (var i = left; i <= middleIndex; i++)
```

```
		{
			tempArray [index] = array [i];
			index++;
		}

		for (var i = right; i <= highIndex; i++)
		{
			tempArray [index] = array [i];
			index++;
		}

		for (var i = 0; i < tempArray.Length; i++)
			array [lowIndex + i] = tempArray [i];
}

private static void sort_merge_strings(string [ ] array)
{
		sort_merge_strings(array, 0, array.Length - 1);
}
private static void sort_merge_strings(string [ ] array, int lowIndex, int highIndex)
{
		if (lowIndex < highIndex)
		{
			var middleIndex = ( lowIndex + highIndex ) / 2;
			sort_merge_strings(array, lowIndex, middleIndex);
			sort_merge_strings(array, middleIndex + 1, highIndex);
			Merge(array, lowIndex, middleIndex, highIndex);
		}
}
private static void Merge(string [ ] array, int lowIndex, int middleIndex, int highIndex)
{
		var left = lowIndex;
		var right = middleIndex + 1;
		var tempArray = new string [highIndex - lowIndex + 1];
		var index = 0;

		while (( left <= middleIndex ) && ( right <= highIndex ))
		{
			if (e1_less_then_e2(array [left], array [right]))
			{
				tempArray [index] = array [left];
				left++;
			}
			else
			{
				tempArray [index] = array [right];
				right++;
			}

			index++;
		}

		for (var i = left; i <= middleIndex; i++)
		{
			tempArray [index] = array [i];
			index++;
		}

		for (var i = right; i <= highIndex; i++)
		{
			tempArray [index] = array [i];
			index++;
		}

		for (var i = 0; i < tempArray.Length; i++)
			array [lowIndex + i] = tempArray [i];
}

private static void sort_merge_dates(DateTime [ ] array)
{
		sort_merge_dates(array, 0, array.Length - 1);
}
private static void sort_merge_dates(DateTime [ ] array, int lowIndex, int highIndex)
{
```

```
		if (lowIndex < highIndex)
		{
			var middleIndex = ( lowIndex + highIndex ) / 2;
			sort_merge_dates(array, lowIndex, middleIndex);
			sort_merge_dates(array, middleIndex + 1, highIndex);
			Merge(array, lowIndex, middleIndex, highIndex);
		}
}
private static void Merge(DateTime [ ] array, int lowIndex, int middleIndex, int highIndex)
{
		var left = lowIndex;
		var right = middleIndex + 1;
		var tempArray = new DateTime [highIndex - lowIndex + 1];
		var index = 0;

		while (( left <= middleIndex ) && ( right <= highIndex ))
		{
			if (array [left] < array [right])
			{
				tempArray [index] = array [left];
				left++;
			}
			else
			{
				tempArray [index] = array [right];
				right++;
			}

			index++;
		}

		for (var i = left; i <= middleIndex; i++)
		{
			tempArray [index] = array [i];
			index++;
		}

		for (var i = right; i <= highIndex; i++)
		{
			tempArray [index] = array [i];
			index++;
		}

		for (var i = 0; i < tempArray.Length; i++)
			array [lowIndex + i] = tempArray [i];
}
#endregion
		#region Сортировка кучей
private static void sort_heap_integers(int [ ] array)
{
		//step 1: building the pyramid
		for (int i = array.Length / 2 - 1; i >= 0; --i)
		{
			int prev_i = i;
			i = add2pyramid(array, i, array.Length);
			if (prev_i != i)
				++i;
		}

		//step 2: sorting
		for (int k = array.Length - 1; k > 0; --k)
		{
			var buf = array [0];
			array [0] = array [k];
			array [k] = buf;
			int i = 0, prev_i = -1;
			while (i != prev_i)
			{
				prev_i = i;
				i = add2pyramid(array, i, k);
			}
		}
}
private static int add2pyramid(int [ ] array, int i, int N)
```

```
{
	int imax;
	if (( 2 * i + 2 ) < N)
	{
		if (array [2 * i + 1] < array [2 * i + 2])
			imax = 2 * i + 2;
		else
			imax = 2 * i + 1;
	}
	else
		imax = 2 * i + 1;
	if (imax >= N)
		return i;
	if (array [i] < array [imax])
	{
		var buf = array [i];
		array [i] = array [imax];
		array [imax] = buf;
		if (imax < N / 2)
			i = imax;
	}
	return i;
}

private static void sort_heap_digits(byte [ ] array)
{
	//step 1: building the pyramid
	for (int i = array.Length / 2 - 1; i >= 0; --i)
	{
		int prev_i = i;
		i = add2pyramid(array, i, array.Length);
		if (prev_i != i)
			++i;
	}

	//step 2: sorting
	for (int k = array.Length - 1; k > 0; --k)
	{
		var buf = array [0];
		array [0] = array [k];
		array [k] = buf;
		int i = 0, prev_i = -1;
		while (i != prev_i)
		{
			prev_i = i;
			i = add2pyramid(array, i, k);
		}
	}
}
private static int add2pyramid(byte [ ] array, int i, int N)
{
	int imax;
	if (( 2 * i + 2 ) < N)
	{
		if (array [2 * i + 1] < array [2 * i + 2])
			imax = 2 * i + 2;
		else
			imax = 2 * i + 1;
	}
	else
		imax = 2 * i + 1;
	if (imax >= N)
		return i;
	if (array [i] < array [imax])
	{
		var buf = array [i];
		array [i] = array [imax];
		array [imax] = buf;
		if (imax < N / 2)
			i = imax;
	}
	return i;
}
```

```
private static void sort_heap_strings(string [ ] array)
{
	//step 1: building the pyramid
	for (int i = array.Length / 2 - 1; i >= 0; --i)
	{
		int prev_i = i;
		i = add2pyramid(array, i, array.Length);
		if (prev_i != i)
			++i;
	}

	//step 2: sorting
	for (int k = array.Length - 1; k > 0; --k)
	{
		var buf = array [0];
		array [0] = array [k];
		array [k] = buf;
		int i = 0, prev_i = -1;
		while (i != prev_i)
		{
			prev_i = i;
			i = add2pyramid(array, i, k);
		}
	}
}
private static int add2pyramid(string [ ] array, int i, int N)
{
	int imax;
	if (( 2 * i + 2 ) < N)
	{
		if (e1_less_then_e2(array [2 * i + 1], array [2 * i + 2]))
			imax = 2 * i + 2;
		else
			imax = 2 * i + 1;
	}
	else
		imax = 2 * i + 1;
	if (imax >= N)
		return i;
	if (e1_less_then_e2(array [i], array [imax]))
	{
		var buf = array [i];
		array [i] = array [imax];
		array [imax] = buf;
		if (imax < N / 2)
			i = imax;
	}
	return i;
}

private static void sort_heap_dates(DateTime [ ] array)
{
	//step 1: building the pyramid
	for (int i = array.Length / 2 - 1; i >= 0; --i)
	{
		int prev_i = i;
		i = add2pyramid(array, i, array.Length);
		if (prev_i != i)
			++i;
	}

	//step 2: sorting
	for (int k = array.Length - 1; k > 0; --k)
	{
		var buf = array [0];
		array [0] = array [k];
		array [k] = buf;
		int i = 0, prev_i = -1;
		while (i != prev_i)
		{
			prev_i = i;
			i = add2pyramid(array, i, k);
		}
	}
```

```
        }
        private static int add2pyramid(DateTime [ ] array, int i, int N)
        {
                int imax;
                if (( 2 * i + 2 ) < N)
                {
                        if (array [2 * i + 1] < array [2 * i + 2])
                                imax = 2 * i + 2;
                        else
                                imax = 2 * i + 1;
                }
                else
                        imax = 2 * i + 1;
                if (imax >= N)
                        return i;
                if (array [i] < array [imax])
                {
                        var buf = array [i];
                        array [i] = array [imax];
                        array [imax] = buf;
                        if (imax < N / 2)
                                i = imax;
                }
                return i;
        }
        #endregion
                #region Встроенная сортировка
        private static void sort_built_in_integers(int [ ] array)
        {
                Array.Sort(array);
        }
        private static void sort_built_in_digits(byte [ ] array)
        {
                Array.Sort(array);
        }
        private static void sort_built_in_strings(string [ ] array)
        {
                Array.Sort(array);
        }
        private static void sort_built_in_dates(DateTime [ ] array)
        {
                Array.Sort(array);
        }
        #endregion
                #region Поразрядная сортирвка (только для чисел)
        public static void sort_bitwise_integers(int [ ] array)
        {
                int range = 10; // 10 цифр

                // Определяем максимальное число разрядов
                int max_el = array.Max<int>();
                int length = 1;
                while (( max_el /= 10 ) > 0)
                        length++;

                List<int> [ ] lists = new List<int> [range];
                for (int i = 0; i < range; ++i)
                        lists [i] = new List<int>();

                for (int step = 0; step < length; ++step)
                {
                        //распределение по спискам
                        for (int i = 0; i < array.Length; ++i)
                        {
                                int temp = ( array [i] % (int) Math.Pow(range, step + 1) ) /
                                                                                                (int)
Math.Pow(range, step);
                                lists [temp].Add(array [i]);
                        }
                        //сборка
                        int k = 0;
                        for (int i = 0; i < range; ++i)
                        {
                                for (int j = 0; j < lists [i].Count; ++j)
```

```
                                        {
                                                array [k++] = (int) lists [i] [j];
                                        }
                                }
                                for (int i = 0; i < range; ++i)
                                        lists [i].Clear();
                        }
}
#endregion
#endregion

#region Генерация массивов
                #region Получение случайных массивов
private byte [ ] get_random_digits_array(int numeric_for_random, int array_length)
{
                Random rnd = new Random(numeric_for_random);
                byte [ ] b = new Byte [array_length];
                rnd.NextBytes(b);
                for (int i = 0; i < b.Length; i++)
                        b [i] = (byte) ( b [i] % 10 );
                return b;
}
private int [ ] get_random_integers_array(int numeric_for_random, int array_length)
{
                Random rnd = new Random(numeric_for_random);
                int [ ] b = new int [array_length];
                for (int i = 0; i < array_length; i++)
                        b [i] = rnd.Next();
                return b;
}
private string [ ] get_random_string_array(int numeric_for_random, int array_length)
{
                Random rnd = new Random(numeric_for_random);
                string [ ] b = new string [array_length];
                int max_len_word = 50;
                for (int i = 0; i < array_length; i++)
                        // get one string
                        for (int j = 0; j < rnd.Next(1, max_len_word); j++)
                                b [i] += (char) rnd.Next();
                return b;
}
private DateTime [ ] get_random_date_array(int numeric_for_random, int array_length)
{
                Random rnd = new Random(numeric_for_random);
                DateTime [ ] b = new DateTime [array_length];
                for (int i = 0; i < array_length; i++)
                        b [i] = b [i].AddYears(rnd.Next(0, 2019))
                                .AddMonths(rnd.Next(0, 12))
                                .AddDays(rnd.Next(0, 31));
                return b;
}
#endregion
                #region Получение массивов, упорядоченных в обратном порядке
private byte [ ] get_reverse_sort_digits_array(int numeric_for_random, int array_length)
{
                byte [ ] b = get_random_digits_array(numeric_for_random, array_length);
                Array.Sort(b);
                Array.Reverse(b);
                return b;
}
private int [ ] get_reverse_sort_integers_array(int numeric_for_random, int array_length)
{
                int [ ] b = get_random_integers_array(numeric_for_random, array_length);
                Array.Sort(b);
                Array.Reverse(b);
                return b;
}
private string [ ] get_reverse_sort_string_array(int numeric_for_random, int array_length)
{
                string [ ] b = get_random_string_array(numeric_for_random, array_length);
                Array.Sort(b);
                Array.Reverse(b);
                return b;
}
```

```
private DateTime [ ] get_reverse_sort_date_array(int numeric_for_random, int array_length)
{
	DateTime [ ] b = get_random_date_array(numeric_for_random, array_length);
	Array.Sort(b);
	Array.Reverse(b);
	return b;
}
#endregion
	#region Получение массивов, состоящих из одинаковых элементов
private byte [ ] get_identical_100_digits_array(int numeric_for_random, int array_length)
{
	Random rnd = new Random(numeric_for_random);
	byte identical_element = Convert.ToByte(rnd.Next() % 10);
	byte [ ] b = new Byte [array_length];
	for (int i = 0; i < b.Length; i++)
		b [i] = identical_element;
	return b;
}
private int [ ] get_identical_100_integers_array(int numeric_for_random, int array_length)
{
	Random rnd = new Random(numeric_for_random);
	int identical_element = rnd.Next();
	int [ ] b = new int [array_length];
	for (int i = 0; i < array_length; i++)
		b [i] = identical_element;
	return b;
}
private string [ ] get_identical_100_string_array(int numeric_for_random, int array_length)
{
	Random rnd = new Random(numeric_for_random);
	string identical_element = "";
	int max_len_word = 50;
	// get one string
	for (int j = 0; j < rnd.Next(1, max_len_word); j++)
		identical_element += (char) rnd.Next();
	string [ ] b = new string [array_length];
	for (int i = 0; i < array_length; i++)
		b [i] = identical_element;
	return b;
}
private DateTime [ ] get_identical_100_date_array(int numeric_for_random, int array_length)
{
	Random rnd = new Random(numeric_for_random);
	DateTime identical_element = new DateTime().AddYears(rnd.Next(0, 2019))
				.AddMonths(rnd.Next(0, 12))
				.AddDays(rnd.Next(0, 31));
	DateTime [ ] b = new DateTime [array_length];
	for (int i = 0; i < array_length; i++)
		b [i] = identical_element;
	return b;
}
#endregion
	#region Получение массивов, на 90% состоящих из одинаковых элементов
private byte [ ] get_identical_90_digits_array(int numeric_for_random, int array_length)
{
	byte [ ] b = new Byte [array_length];
	byte [ ] b_different = get_random_digits_array(numeric_for_random + 1, (int) ( 0.1 * b.Length ));
	for (int i = 0; i < b_different.Length; i++)
		b [i] = b_different [i];
	Random rnd = new Random(numeric_for_random);
	byte identical_element = Convert.ToByte(rnd.Next() % 10);
	for (int i = b_different.Length; i < b.Length; i++)
		b [i] = identical_element;
	return b;
}
private int [ ] get_identical_90_integers_array(int numeric_for_random, int array_length)
{
	int [ ] b = new int [array_length];
	int [ ] b_different = get_random_integers_array(numeric_for_random + 1, (int) ( 0.1 * b.Length ));
	for (int i = 0; i < b_different.Length; i++)
		b [i] = b_different [i];
	Random rnd = new Random(numeric_for_random);
	int identical_element = rnd.Next();
	for (int i = b_different.Length; i < b.Length; i++)
```

```
                        b [i] = identical_element;
                return b;
        }
        private string [ ] get_identical_90_string_array(int numeric_for_random, int array_length)
        {
                string [ ] b = new string [array_length];
                string [ ] b_different = get_random_string_array(numeric_for_random + 1, (int) ( 0.1 * b.Length ));
                for (int i = 0; i < b_different.Length; i++)
                        b [i] = b_different [i];
                Random rnd = new Random(numeric_for_random);
                string identical_element = "";
                int max_len_word = 50;
                // get one string
                for (int j = 0; j < rnd.Next(1, max_len_word); j++)
                        identical_element += (char) rnd.Next();
                for (int i = b_different.Length; i < b.Length; i++)
                        b [i] = identical_element;
                return b;
        }
        private DateTime [ ] get_identical_90_date_array(int numeric_for_random, int array_length)
        {
                DateTime [ ] b = new DateTime [array_length];
                DateTime [ ] b_different = get_random_date_array(numeric_for_random + 1, (int) ( 0.1 * b.Length ));
                for (int i = 0; i < b_different.Length; i++)
                        b [i] = b_different [i];
                Random rnd = new Random(numeric_for_random);
                DateTime identical_element = new DateTime().AddYears(rnd.Next(0, 2019))
                                .AddMonths(rnd.Next(0, 12))
                                .AddDays(rnd.Next(0, 31));
                for (int i = b_different.Length; i < b.Length; i++)
                        b [i] = identical_element;
                return b;
        }
        #endregion
                #region Получение массивов, у которых первая половина элементов упорядочена и любой элемент
второй половины больше самого большого элемента из первой половины.
        private byte [ ] get_sort1_and_2more_then1_digits_array(int numeric_for_random, int array_length)
        {
                int border = ( 9 + 0 ) / 2;
                byte [ ] b = new Byte [array_length];
                byte [ ] b_1part = get_random_digits_array(numeric_for_random + 1, (int) ( 0.5 * b.Length ));
                for (int i = 0; i < b_1part.Length; i++)
                        if (b_1part [i] > border)
                                b_1part [i] -= (byte) ( border + 1 );
                Array.Sort(b_1part);
                for (int i = 0; i < b_1part.Length; i++)
                        b [i] = b_1part [i];

                byte [ ] b_2part = get_random_digits_array(numeric_for_random + 2, b.Length - b_1part.Length);
                for (int i = 0; i < b_2part.Length; i++)
                {
                        if (b_2part [i] < border + 1)
                                b_2part [i] += (byte) ( border + 1 );
                        b [b_1part.Length + i] = b_2part [i];
                }
                return b;
        }
        private int [ ] get_sort1_and_2more_then1_integers_array(int numeric_for_random, int array_length)
        {
                int border = int.MaxValue / 2;
                int [ ] b = new int [array_length];
                int [ ] b_1part = get_random_integers_array(numeric_for_random + 1, (int) ( 0.5 * b.Length ));
                for (int i = 0; i < b_1part.Length; i++)
                        if (b_1part [i] > border)
                                b_1part [i] -= ( border + 1 );
                Array.Sort(b_1part);
                for (int i = 0; i < b_1part.Length; i++)
                        b [i] = b_1part [i];
                int [ ] b_2part = get_random_integers_array(numeric_for_random + 2, b.Length - b_1part.Length);
                for (int i = 0; i < b_2part.Length; i++)
                {
                        if (b_2part [i] < border + 1)
                                b_2part [i] += ( border + 1 );
                        b [b_1part.Length + i] = b_2part [i];
```

```
                }
                return b;
        }
        private string [ ] get_sort1_and_2more_then1_string_array(int numeric_for_random, int array_length)
        {
                int border = char.MaxValue / 2;
                string [ ] b = new string [array_length];
                string [ ] b_1part = get_random_string_array(numeric_for_random + 1, (int) ( 0.5 * b.Length ));
                for (int i = 0; i < b_1part.Length; i++)
                        if (b_1part [i] [0] > border)
                                b_1part [i].Insert(0, ( (char) ( b_1part [i] [0] - border + 1 ) ).ToString());
                Array.Sort(b_1part);
                for (int i = 0; i < b_1part.Length; i++)
                        b [i] = b_1part [i];
                string [ ] b_2part = get_random_string_array(numeric_for_random + 2, b.Length - b_1part.Length);
                for (int i = 0; i < b_2part.Length; i++)
                {
                        if (b_2part [i] [0] < border + 1)
                                b_2part [i].Insert(0, ( (char) ( b_1part [i] [0] + border + 1 ) ).ToString());
                        b [b_1part.Length + i] = b_2part [i];
                }
                return b;
        }
        private DateTime [ ] get_sort1_and_2more_then1_date_array(int numeric_for_random, int array_length)
        {
                Random rnd = new Random(numeric_for_random);
                DateTime [ ] b = new DateTime [array_length];
                DateTime [ ] b_1part = new DateTime [(int) ( array_length / 2 )];
                for (int i = 0; i < array_length / 2; i++)
                        b_1part [i] = b_1part [i].AddYears(rnd.Next(0, 1010))
                                .AddMonths(rnd.Next(0, 12))
                                .AddDays(rnd.Next(0, 31));
                Array.Sort(b_1part);
                for (int i = 0; i < array_length / 2; i++)
                        b [i] = b_1part [i];
                for (int i = (int) ( array_length / 2 ); i < array_length; i++)
                        b [i] = b [i].AddYears(rnd.Next(1010, 2020))
                                .AddMonths(rnd.Next(0, 12))
                                .AddDays(rnd.Next(0, 31));
                return b;
        }
        #endregion
                #region Получение массивов, у которых первая половина элементов упорядочена, а вторая половина
случайно сгенерирована из элементов, которые могут быть меньше элементов первой половины
        private byte [ ] get_sort1_and_unsort_digits_array(int numeric_for_random, int array_length)
        {
                byte [ ] b = new Byte [array_length];
                byte [ ] b_1part = get_random_digits_array(numeric_for_random + 1, (int) ( 0.5 * b.Length ));
                Array.Sort(b_1part);
                for (int i = 0; i < b_1part.Length; i++)
                        b [i] = b_1part [i];
                byte [ ] b_2part = get_random_digits_array(numeric_for_random + 2, b.Length - b_1part.Length);
                for (int i = 0; i < b_2part.Length; i++)
                        b [b_1part.Length + i] = b_2part [i];
                return b;
        }
        private int [ ] get_sort1_and_unsort_integers_array(int numeric_for_random, int array_length)
        {
                int [ ] b = new int [array_length];
                int [ ] b_1part = get_random_integers_array(numeric_for_random + 1, (int) ( 0.5 * b.Length ));
                Array.Sort(b_1part);
                for (int i = 0; i < b_1part.Length; i++)
                        b [i] = b_1part [i];
                int [ ] b_2part = get_random_integers_array(numeric_for_random + 2, b.Length - b_1part.Length);
                for (int i = 0; i < b_2part.Length; i++)
                        b [b_1part.Length + i] = b_2part [i];
                return b;
        }
        private string [ ] get_sort1_and_unsort_string_array(int numeric_for_random, int array_length)
        {
                string [ ] b = new string [array_length];
                string [ ] b_1part = get_random_string_array(numeric_for_random + 1, (int) ( 0.5 * b.Length ));
                Array.Sort(b_1part);
                for (int i = 0; i < b_1part.Length; i++)
```

```
                b [i] = b_1part [i];
        string [ ] b_2part = get_random_string_array(numeric_for_random + 2, b.Length - b_1part.Length);
        for (int i = 0; i < b_2part.Length; i++)
                b [b_1part.Length + i] = b_2part [i];
        return b;
}
private DateTime [ ] get_sort1_and_unsort_date_array(int numeric_for_random, int array_length)
{
        Random rnd = new Random(numeric_for_random);
        DateTime [ ] b = new DateTime [array_length];
        DateTime [ ] b_1part = get_random_date_array(numeric_for_random + 1, (int) ( 0.5 * b.Length ));
        Array.Sort(b_1part);
        for (int i = 0; i < array_length / 2; i++)
                b [i] = b_1part [i];
        DateTime [ ] b_2part = get_random_date_array(numeric_for_random + 2, b.Length - b_1part.Length);
        for (int i = 0; i < b_2part.Length; i++)
                b [b_1part.Length + i] = b_2part [i];
        return b;
}
#endregion
#endregion

#region Другие функции
// Функция, которая отвечает на вопрос: 1строка меньше 2строки ?
private static bool e1_less_then_e2(string str1, string str2)
{
        for (int i = 0; ; i++)
        {
                if (str1 [i] < str2 [i])
                        return true;
                else if (str1 [i] > str2 [i])
                        return false;
                else if (i + 1 == str1.Length)
                        return false; // Спорный момент
                else if (i + 1 == str2.Length)
                        return false;
        }
}
// Функция по номеру определяет тип генерации
private static string get_name_generation(int i)
{
        switch (i)
        {
                case 0:
                        return "Случайная";
                case 1:
                        return "В обратном порядке";
                case 2:
                        return "100% - одинаковые";
                case 3:
                        return "90% - одинаковые";
                case 4:
                        return "1полов(упор) < 2полов(неупор)";
                case 5:
                        return "1полов(упор), 2полов(неупор)";
                default:
                        return "";

        }
}
// Функция по номеру определяет название сортировки
private static string get_name_sort(int i)
{
        switch (i)
        {
                case 0:
                        return "Пузырьком";
                case 1:
                        return "Вставками";
                case 2:
                        return "Шелла";
                case 3:
                        return "Быстрая";
                case 4:
```

```
                                        return "Слиянием";
                                    case 5:
                                        return "Кучей";
                                    case 6:
                                        return "Встроенная";
                                    case 7:
                                        return "Поразрядная";
                                    default:
                                        return "";
                                }
                            }
                            #endregion

                        }
                    }
```