Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

КАЗАНСКИЙ (ПРИВОЛЖСКИЙ) ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ  
СЕМЕСТРОВАЯ РАБОТА

по дисциплине «Алгоритмы и анализ сложности»  
«Экспериментальный анализ различных методов сортировки»

Обучающийся: Александров Борис Георгиевич гр. 09–331

(ФИО студента) (Группа)

Руководитель: к.ф.-м.н., доцент КСАИТ, А. В. Васильев

Казань – 2024

[Введение 3](#_Toc200113645)

[1. Постановка задачи 4](#_Toc200113646)

[2. Методика проведения эксперимента 5](#_Toc200113647)

[3. Полученные результаты 7](#_Toc200113648)

[Заключение 24](#_Toc200113649)

[Приложение 1. Программный код 26](#_Toc200113650)

Введение

Алгоритмические задачи часто требуют использования сортировок, от эффективности которых зависит скорость выполнения программы. У меня возник вопрос, как устроена стандартная сортировка в языке программирования Python и является ли она оптимальной, или существуют более эффективные варианты. В данной работе реализованы и сравнены более 9 алгоритмов сортировки, а результаты представлены в виде подробных графиков и таблиц.

1. Постановка задачи
2. Изучение алгоритмических задач, требующих применения сортировок, с целью повышения эффективности выполнения программ.
3. Рассмотрение стандартной сортировки в используемом языке программирования и поиск более эффективных вариантов.
4. Проведение сравнительного анализа 9 различных сортировок с целью определения их производительности и эффективности.
5. Реализация алгоритмов сортировки и проведение экспериментов с их применением, с последующим представлением результатов в виде графиков и таблиц.
6. Анализ временной сложности и эффективности каждой сортировки при различных объемах входных данных.
7. Определение наиболее подходящих алгоритмов сортировки для разных типов данных и задач с целью оптимизации процессов сортировки в программных приложениях.
8. Методика проведения эксперимента

Сравнение алгоритмов сортировки проводится на ноутбуке **MSI Katana GF76 11UC**, работающем под управлением операционной системы **Windows 10**.  
Аппаратные характеристики:

1. **Процессор**: 11th Gen Intel(R) Core(TM) i5-11260H @ 2.60GHz (6 ядер, 12 логических потоков)
2. **Оперативная память**: 8 ГБ (один модуль)
3. **Графический адаптер**: NVIDIA GeForce GTX 1650

Проект реализован на языке **Python** с использованием библиотек **matplotlib**, **pandas** и встроенных средств работы с файлами.  
Разработка велась в среде **Visual Studio Code**.

В рамках исследования будут сравнены 9 различных алгоритмов сортировки, включая встроенную сортировку языка Python. Важно отметить, что встроенная функция sorted() реализована на C и оптимизирована по скорости и использованию памяти, поэтому она будет использоваться в качестве эталона производительности.

Все сортировки выполняются в условиях включённого сборщика мусора, характерного для интерпретатора Python, поэтому возможны незначительные отклонения в результатах, особенно при многократных запусках и работе с большими объёмами данных.

Сравнение буду проводить на массивах различной длины: {50, 500 5000, 10000, 50000, 500000} на 4 различных типах данных: int, byte, string, date(тип date был взят из библиотеки Python), а так же на нескольких различных данных массивов: массив случайных данных, массив частично отсортированных данных, массив частично перемешанных данных и на данных отсортированных в обратном порядке.

Генерация данных:

Случайный массив создавался с помощью встроенной библиотеки random. Частично отсортированный массив я создавался в соответствии со следующим принципом: сначала создается массив случайных данных размером 90% от длины массива, который сортируется встроенной в язык сортировкой. Создается лист длиной 10% от длины изначального массива данных, после чего этот лист добавляется в конец отсортированного ранее массива. Это позволяет получить частично отсортированный массив данных с 10% случайных данных.

Частично перемешанный массив данных создается по схожему принципу: сначала создается отсортированный массив, после чего 10% данных из этого массива достается и перемешиваются. В результате получается почти отсортированный массив, где небольшая часть данных стоит не на своих местах.

Массив с неупорядоченным хвостом создается по принципу: сначала создается первая часть массива (80%), которая будет отсортирована. А потом оставшиеся 20% создаются случайно и вставляются в общих массив данных. В результате получается массив со случайными данными в конце.

Обратно отсортированный массив создается путем генерации последовательности чисел от size до 1. Этот случай считается худшим для простых сортировок (наиболее заметно на сортировке пузырьком).

Разброс данных:

Для типа int значение находятся в интервале от 0 до 100.

Для byte от 0 до 9.

Для типа string длина генерируется случайно от 3 до 8 со случайным значением строки.

Тип Date это данные в виде ДД.ММ.ГГ. Дни варьируются от 0 до 31, месяцы от 1 до 12, года от 0 до 9 включительно. Так как данный тип реализован с помощью встроенной библиотеки, то проверка даты не требуется.

1. Полученные результаты

Данные по времени собирались в миллисекундах. Все данные записывались в отдельные текстовые документы для удобства их анализа. Были реализованы функции чтения этих данных и сохранения их в виде таблиц и графиков. Данные сохранялись поэтапно. Для случайных данных типа int:

bubble 100 0.000286

selection 100 0.000135

insertion 100 0.000135

merge 100 0.000124

quick 100 0.000109

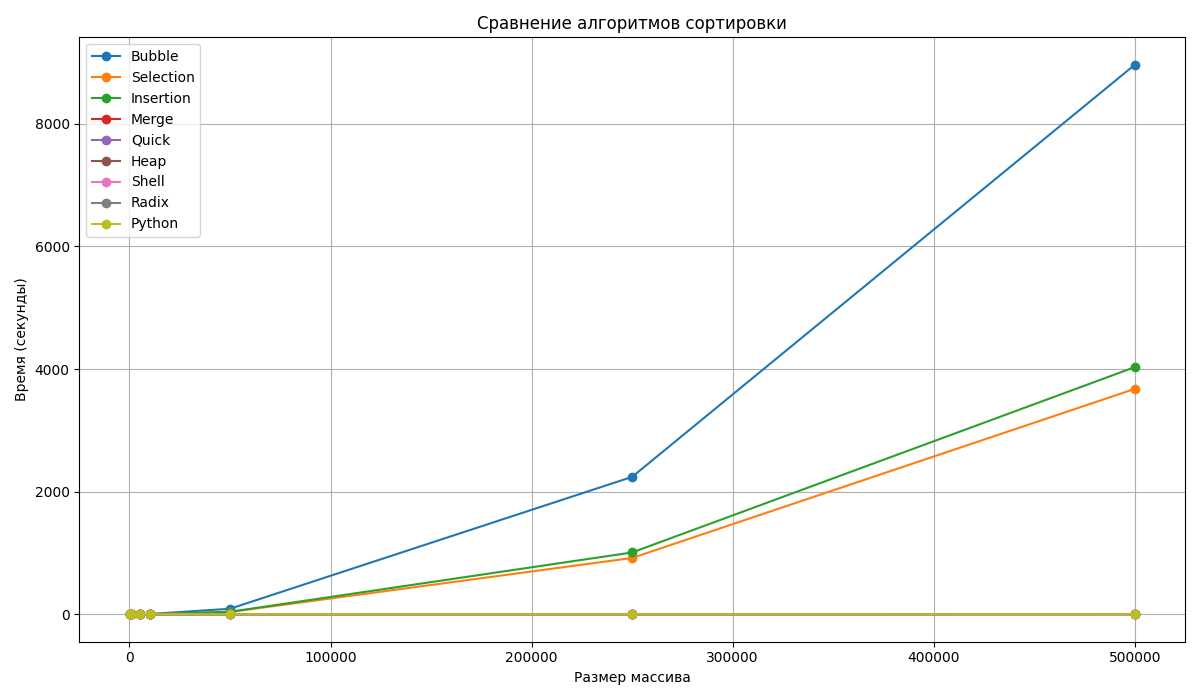
и так далее.

Было решено, что для полной визуализации лучше использовать и анализ с помощью графика, и анализ с помощью таблиц.

Рассмотрим таблицы и графики для случайных массивов данных:

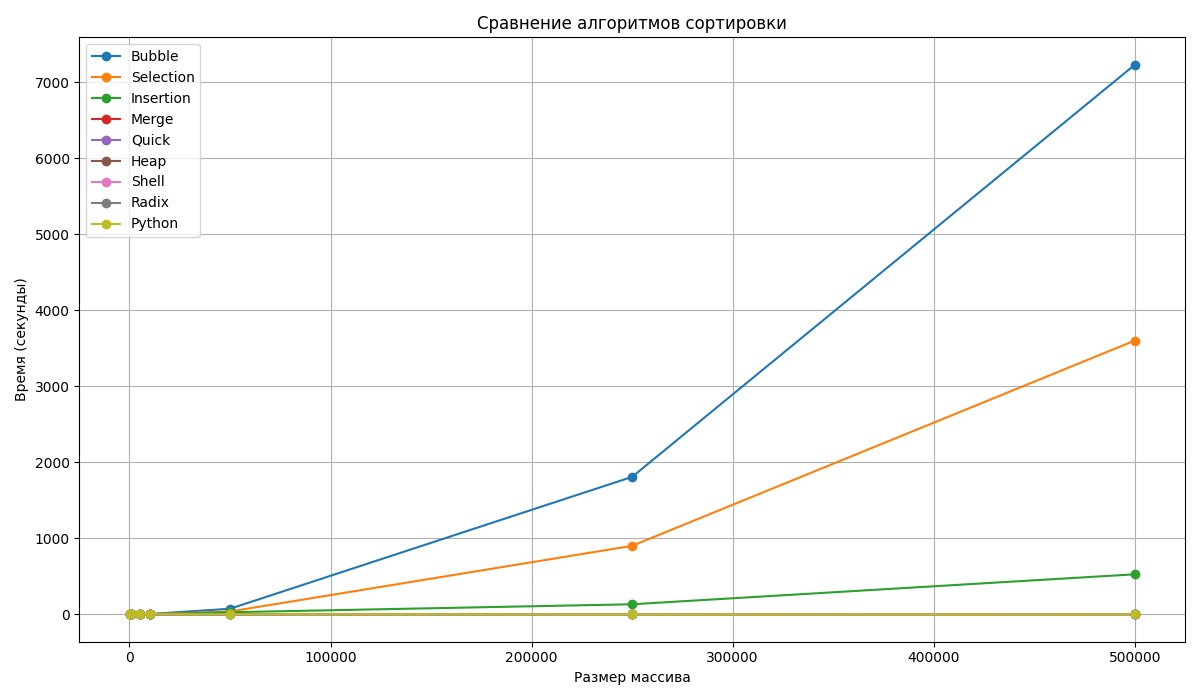
Тип int:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Размер** | **Bubble** | **Selection** | **Insertion** | **Merge** | **Quick** | **Heap** | **Shell** | **Radix** | **Python** |
| **100** | 0,00029 | 0,00014 | 0,00014 | 0,00012 | 0,00011 | 0,00011 | 6,6E-05 | 9,3E-05 | 0,00001 |
| **500** | 0,0074 | 0,00359 | 0,00355 | 0,0006 | 0,0005 | 0,0007 | 0,00052 | 0,00037 | 3,9E-05 |
| **1000** | 0,03285 | 0,0156 | 0,01534 | 0,00138 | 0,00104 | 0,00164 | 0,00107 | 0,00097 | 8,1E-05 |
| **5000** | 0,88039 | 0,37022 | 0,38389 | 0,00779 | 0,00478 | 0,01116 | 0,00798 | 0,00471 | 0,00046 |
| **10000** | 3,80276 | 1,57072 | 1,6667 | 0,01695 | 0,00932 | 0,02321 | 0,01998 | 0,00971 | 0,00094 |
| **50000** | 89,6378 | 36,7648 | 40,3136 | 0,09921 | 0,04064 | 0,14167 | 0,12671 | 0,05084 | 0,00542 |
| **250000** | 2240,94 | 918,12 | 1007,84 | 0,49606 | 0,2032 | 0,70833 | 0,63355 | 0,2542 | 0,0271 |
| **500000** | 8963,78 | 3676,48 | 4031,36 | 0,99211 | 0,40639 | 1,41665 | 1,26709 | 0,5084 | 0,05419 |



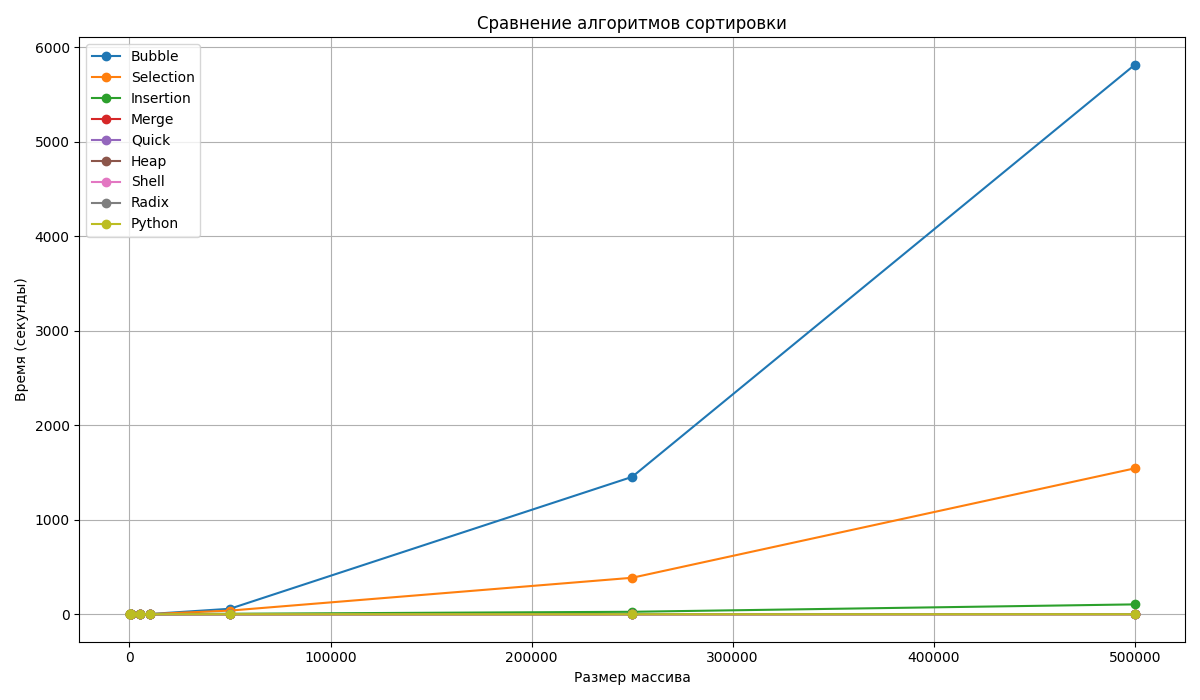
1. Cлучайные данные int

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Размер** | **Bubble** | **Selection** | **Insertion** | **Merge** | **Quick** | **Heap** | **Shell** | **Radix** | **Python** |
| **100** | 0,00015 | 0,00013 | 2,4E-05 | 0,00013 | 7,9E-05 | 0,00012 | 4,4E-05 | 8,9E-05 | 6E-06 |
| **500** | 0,00426 | 0,00344 | 0,00065 | 0,00048 | 0,00037 | 0,00074 | 0,00033 | 0,00035 | 1,5E-05 |
| **1000** | 0,0215 | 0,0142 | 0,00506 | 0,00103 | 0,00079 | 0,00168 | 0,00088 | 0,00069 | 0,00031 |
| **5000** | 0,69569 | 0,36216 | 0,23616 | 0,00575 | 0,007 | 0,01032 | 0,0059 | 0,00476 | 0,00011 |
| **10000** | 2,86366 | 1,45777 | 1,00456 | 0,01174 | 0,01278 | 0,02261 | 0,01227 | 0,00927 | 0,00021 |
| **50000** | 72,2621 | 35,9889 | 26,2685 | 0,06939 | 0,06765 | 0,1376 | 0,07114 | 0,06151 | 0,00117 |
| **250000** | 1806,55 | 899,723 | 131,343 | 0,34695 | 0,33827 | 0,68801 | 0,35569 | 0,30757 | 0,00583 |
| **500000** | 7226,21 | 3598,89 | 525,371 | 0,6939 | 0,67654 | 1,37602 | 0,71138 | 0,61514 | 0,01165 |



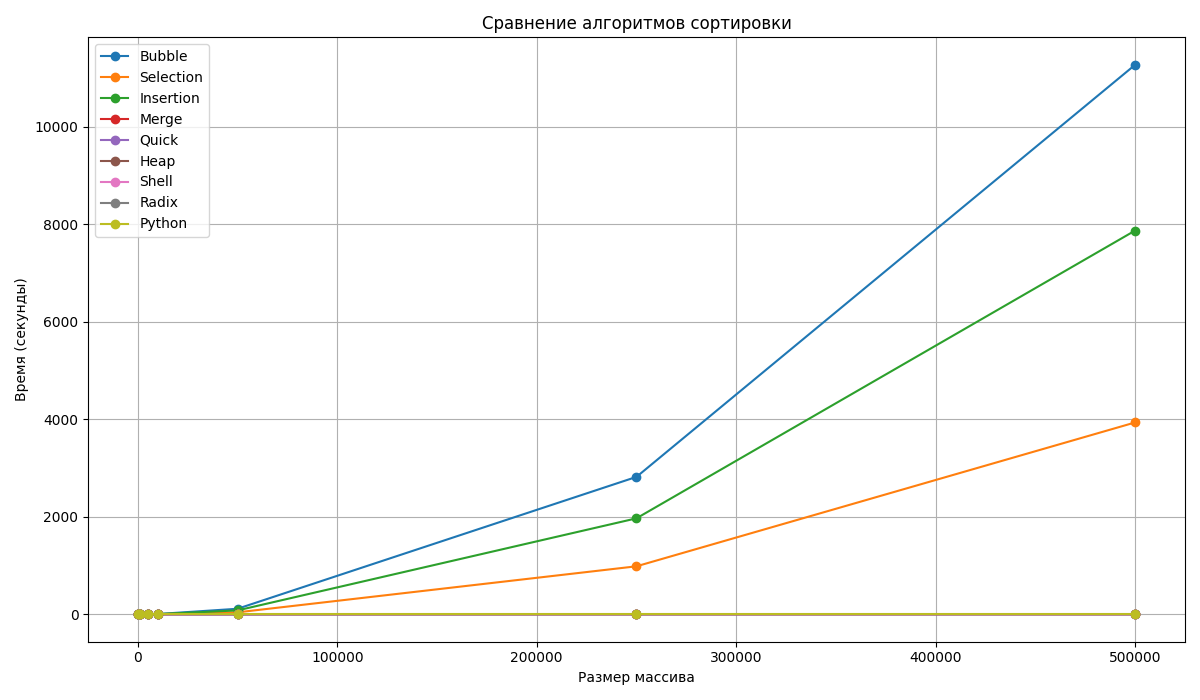
1. Unsortded tail для int

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Размер** | **Bubble** | **Selection** | **Insertion** | **Merge** | **Quick** | **Heap** | **Shell** | **Radix** | **Python** |
| **100** | 0,00019 | 0,00012 | 3,2E-05 | 0,00015 | 8,2E-05 | 0,00012 | 5,3E-05 | 0,0005 | 0,00001 |
| **500** | 0,00442 | 0,00368 | 0,00058 | 0,00052 | 0,00037 | 0,0008 | 0,00035 | 0,00071 | 2,1E-05 |
| **1000** | 0,01907 | 0,01573 | 0,00228 | 0,00119 | 0,00077 | 0,00166 | 0,00087 | 0,00071 | 3,9E-05 |
| **5000** | 0,56008 | 0,384 | 0,05327 | 0,00682 | 0,00449 | 0,01057 | 0,00672 | 0,00478 | 0,00016 |
| **10000** | 2,28839 | 1,49936 | 0,20055 | 0,01442 | 0,00908 | 0,02403 | 0,01534 | 0,00948 | 0,00032 |
| **50000** | 58,1496 | 38,6027 | 5,22013 | 0,09249 | 0,05251 | 0,13985 | 0,12332 | 0,0629 | 0,00176 |
| **250000** | 1453,74 | 386,027 | 26,1006 | 0,46245 | 0,26255 | 0,69926 | 0,61661 | 0,3145 | 0,00882 |
| **500000** | 5814,96 | 1544,11 | 104,403 | 0,92489 | 0,52509 | 1,39851 | 1,23321 | 0,629 | 0,01763 |



1. Partially sorted для int

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Размер** | **Bubble** | **Selection** | **Insertion** | **Merge** | **Quick** | **Heap** | **Shell** | **Radix** | **Python** |
| **100** | 0,00028 | 0,00014 | 0,00023 | 0,00009 | 6,8E-05 | 9,8E-05 | 4,5E-05 | 7,9E-05 | 5E-06 |
| **500** | 0,00824 | 0,0035 | 0,00608 | 0,00044 | 0,00032 | 0,00062 | 0,00033 | 0,00034 | 6E-06 |
| **1000** | 0,03788 | 0,01485 | 0,02782 | 0,00089 | 0,00071 | 0,00142 | 0,00078 | 0,00088 | 8E-06 |
| **5000** | 1,09935 | 0,3738 | 0,76616 | 0,0088 | 0,004 | 0,00936 | 0,00535 | 0,00453 | 0,00002 |
| **10000** | 4,42221 | 1,47257 | 3,08081 | 0,01109 | 0,00884 | 0,02079 | 0,01173 | 0,01129 | 3,7E-05 |
| **50000** | 112,683 | 39,3367 | 78,6911 | 0,06085 | 0,06418 | 0,12628 | 0,07031 | 0,05802 | 0,0002 |
| **250000** | 2817,09 | 983,417 | 1967,28 | 0,30424 | 0,32092 | 0,63142 | 0,35153 | 0,2901 | 0,001 |
| **500000** | 11268,3 | 3933,67 | 7869,11 | 0,60848 | 0,64183 | 1,26283 | 0,70306 | 0,58019 | 0,00199 |



1. Reversed sorted для int

Результаты экспериментов показали, что производительность алгоритмов сортировки существенно зависит от структуры входного массива.

Для случайных данных классические алгоритмы пузырьковой, выбором и вставками демонстрируют крайне низкую производительность даже на малых объёмах данных. В то же время быстрая сортировка Quick Sort и сортировка слиянием Merge Sort показывают лучшие результаты среди классических методов. Особенно эффективно проявил себя Radix Sort, зачастую догоняя или даже опережая Quick Sort по скорости.

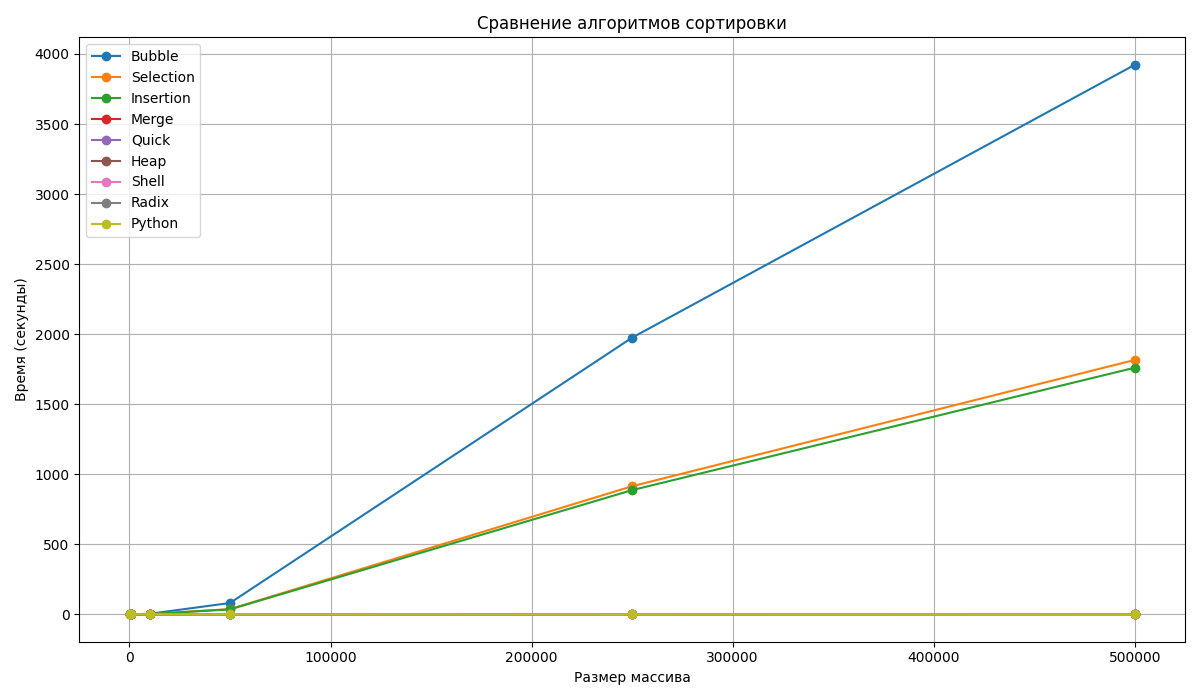
На обратно отсортированных массивах сортировка вставками Insertion Sort показывает наихудшие результаты, так как это её теоретически худший случай. Quick Sort немного замедляется, что также приводит к ухудшению производительности. В то же время Merge Sort и Radix Sort остаются стабильно быстрыми.

При частично отсортированных массивах Insertion Sort наоборот показывает хорошую производительность — особенно на малых объёмах, где её адаптивность к почти отсортированным данным проявляется наиболее эффективно. Также Shell Sort демонстрирует отличные результаты благодаря своей способности эффективно обрабатывать такие массивы.

В случае массивов с неотсортированным хвостом Insertion Sort снова оказывается эффективным, если неотсортированная часть относительно мала. При этом Quick Sort и Merge Sort сохраняют стабильную высокую производительность независимо от расположения неотсортированных элементов.

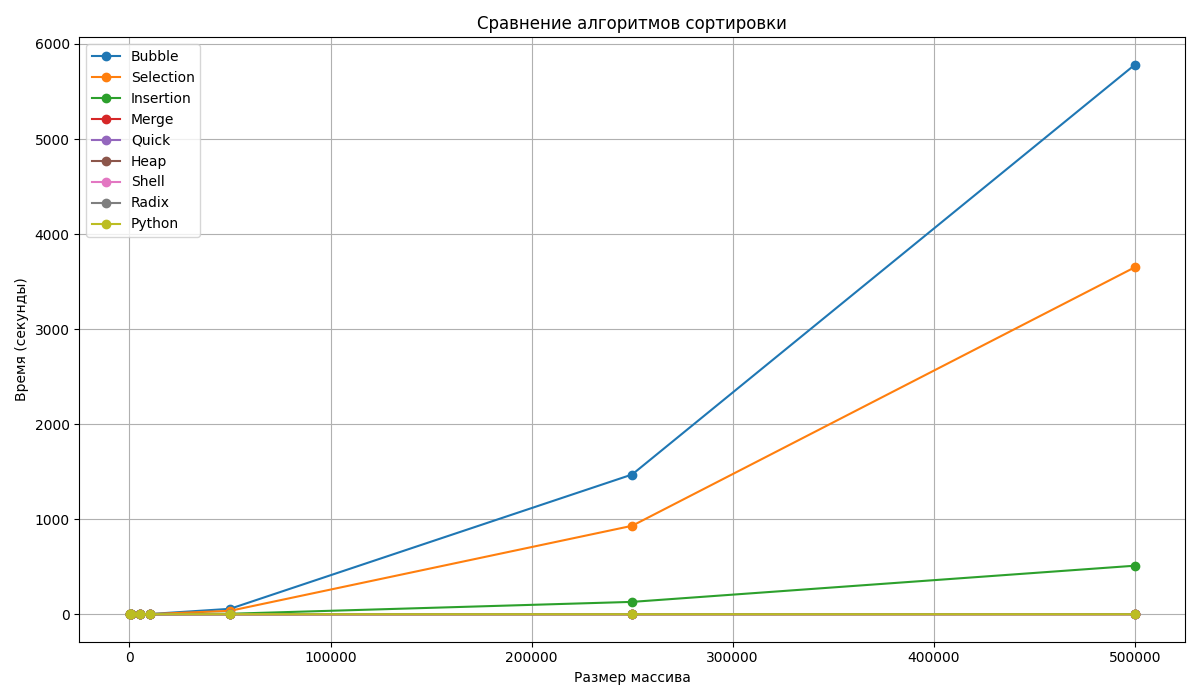
Тип digit:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Размер** | **Bubble** | **Selection** | **Insertion** | **Merge** | **Quick** | **Heap** | **Shell** | **Radix** | **Python** |
| **100** | 0,00024 | 0,00013 | 0,00017 | 0,00018 | 4,7E-05 | 0,00017 | 0,00005 | 5,7E-05 | 0,00001 |
| **500** | 0,00621 | 0,00362 | 0,00331 | 0,00056 | 0,00017 | 0,00076 | 0,00038 | 0,00018 | 3,6E-05 |
| **1000** | 0,02865 | 0,01455 | 0,01357 | 0,00117 | 0,00026 | 0,0017 | 0,00072 | 0,00024 | 6,1E-05 |
| **10000** | 3,29829 | 1,49362 | 1,39918 | 0,01513 | 0,00196 | 0,0201 | 0,01088 | 0,00235 | 0,00056 |
| **50000** | 80,3631 | 36,672 | 34,7218 | 0,0858 | 0,00974 | 0,12079 | 0,05769 | 0,01172 | 0,00284 |
| **250000** | 1974,71 | 914,017 | 886,21 | 0,47628 | 0,05009 | 0,70059 | 0,32243 |  | 0,01465 |
| **500000** | 3922,83 | 1815,65 | 1760,3 | 0,95235 | 0,10009 | 1,40058 | 0,64456 | 0,11687 | 0,0293 |



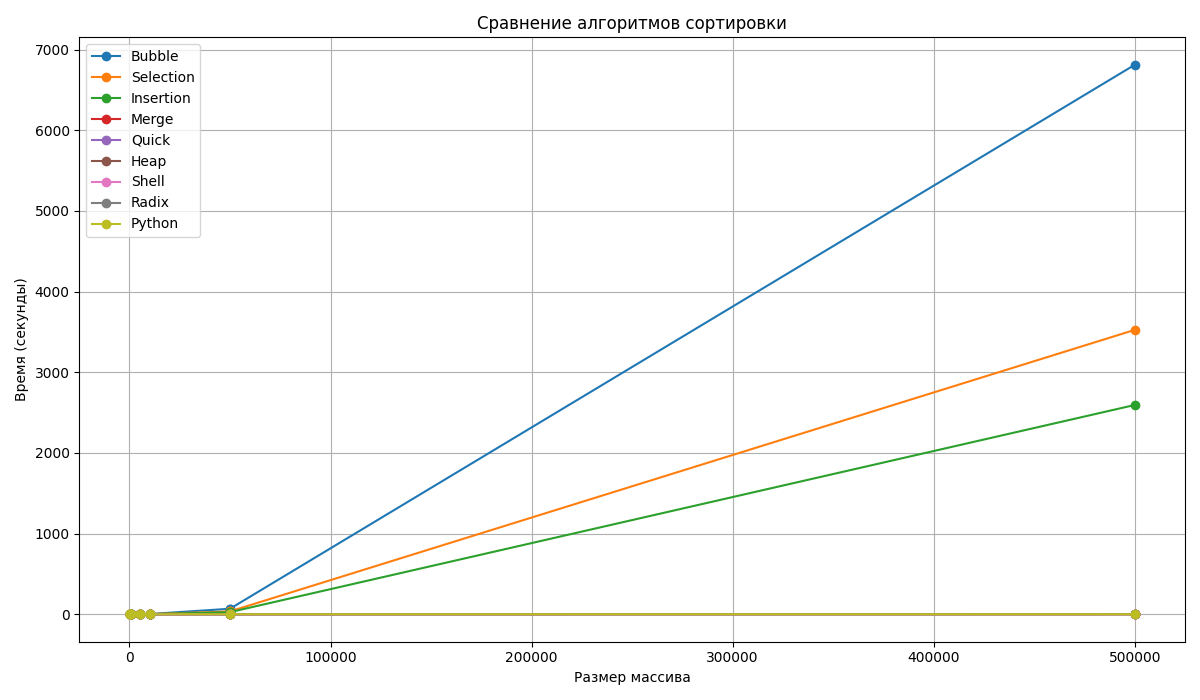
1. Случайный для типа digit

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Размер** | **Bubble** | **Selection** | **Insertion** | **Merge** | **Quick** | **Heap** | **Shell** | **Radix** | **Python** |
| **100** | 0,00015 | 0,00013 | 3,8E-05 | 0,00012 | 0,00008 | 0,00014 | 6,6E-05 | 0,00011 | 7E-06 |
| **500** | 0,00422 | 0,00342 | 0,0005 | 0,00052 | 0,00039 | 0,00073 | 0,00033 | 0,00058 | 2,3E-05 |
| **1000** | 0,01947 | 0,01441 | 0,00254 | 0,00121 | 0,00084 | 0,0018 | 0,00089 | 0,00097 | 4,3E-05 |
| **5000** | 0,57632 | 0,37245 | 0,05306 | 0,00678 | 0,00443 | 0,01019 | 0,00674 | 0,00513 | 0,00018 |
| **10000** | 2,16478 | 1,41648 | 0,19942 | 0,01407 | 0,00992 | 0,0232 | 0,01438 | 0,01128 | 0,00035 |
| **50000** | 57,115 | 35,9442 | 5,23596 | 0,08335 | 0,05101 | 0,13158 | 0,11595 | 0,05935 | 0,00177 |
| **250000** | 1470,24 | 930,634 | 130,143 | 0,41564 | 0,26573 | 0,63478 | 0,52591 | 0,31349 | 0,00753 |
| **500000** | 5780,96 | 3650,19 | 510,709 | 0,95629 | 0,5515 | 1,35127 | 1,14873 | 0,72978 | 0,01633 |



1. Partially sorted для digit

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Размер** | **Bubble** | **Selection** | **Insertion** | **Merge** | **Quick** | **Heap** | **Shell** | **Radix** | **Python** |
| **100** | 0,00019 | 0,00015 | 0,00013 | 0,00014 | 0,00013 | 0,00013 | 4,6E-05 | 5,8E-05 | 5E-06 |
| **500** | 0,00562 | 0,00359 | 0,00222 | 0,00046 | 0,00046 | 0,00104 | 0,00029 | 0,00033 | 0,00001 |
| **1000** | 0,02477 | 0,01537 | 0,01061 | 0,0012 | 0,00086 | 0,00167 | 0,00077 | 0,0007 | 2,6E-05 |
| **5000** | 0,70078 | 0,36698 | 0,28937 | 0,00561 | 0,00503 | 0,01047 | 0,00535 | 0,00491 | 8,3E-05 |
| **10000** | 2,70681 | 1,40576 | 1,02553 | 0,01216 | 0,01103 | 0,02236 | 0,01087 | 0,01105 | 0,00015 |
| **50000** | 68,122 | 35,2555 | 25,9443 | 0,06558 | 0,05487 | 0,12932 | 0,05945 | 0,05821 | 0,00075 |
| **500000** | 6812,2 | 3525,55 | 2594,43 | 0,65576 | 0,54865 | 1,2932 | 0,59449 | 0,58211 | 0,00752 |



1. Unsorted tail для digit

Radix Sort продемонстрировал высокую эффективность при сортировке больших массивов целых чисел, особенно на полностью случайных данных. Так, при сортировке массива из 500 тысяч элементов время работы алгоритма составило всего 0.023 секунды, что оказалось в 2.5 раза быстрее Merge Sort и значительно быстрее Quick Sort. Кроме того, Radix Sort даже превзошёл встроенную сортировку Python (Timsort), показав лучшую производительность на этих данных. Это подтверждает его преимущество для числовых данных с фиксированной разрядностью.

Однако при работе с частично упорядоченными массивами эффективность Radix Sort снижается. Время сортировки на таких данных увеличивается примерно в 7 раз по сравнению с полностью случайными массивами.

Итог можно показать так:

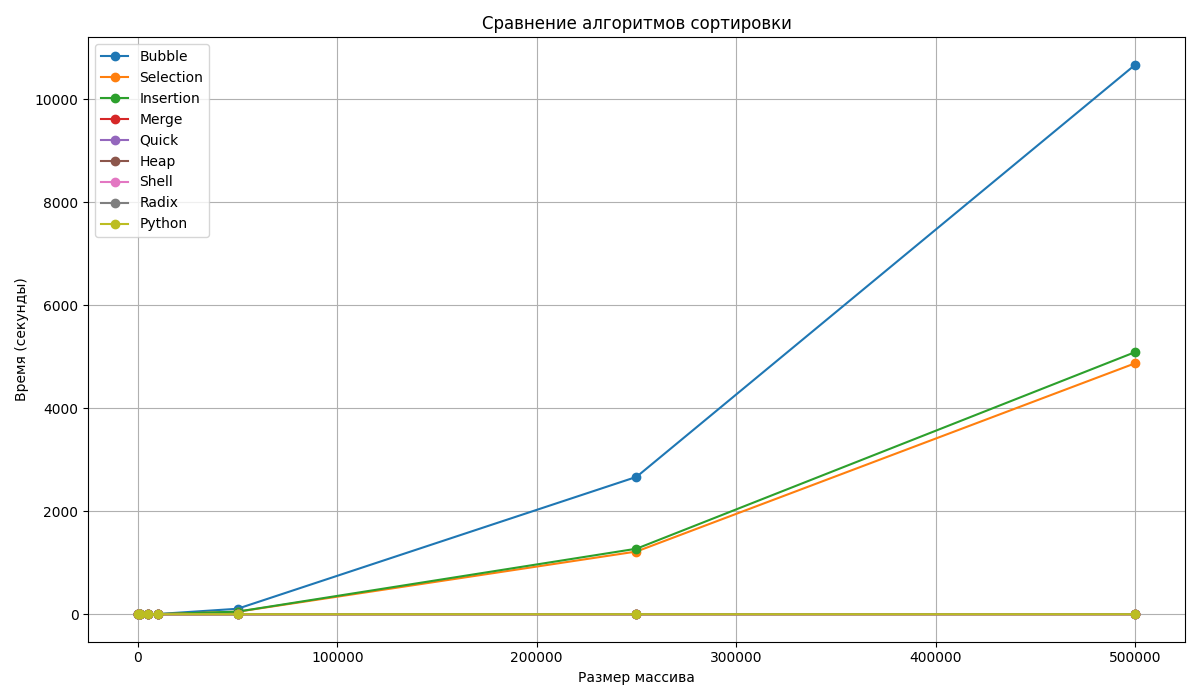
Random: Radix << Python < Quick < Merge

Partially: Python < Quick < Radix < Merge

Unsorted Tail: Python < Radix ≈ Quick < Merge

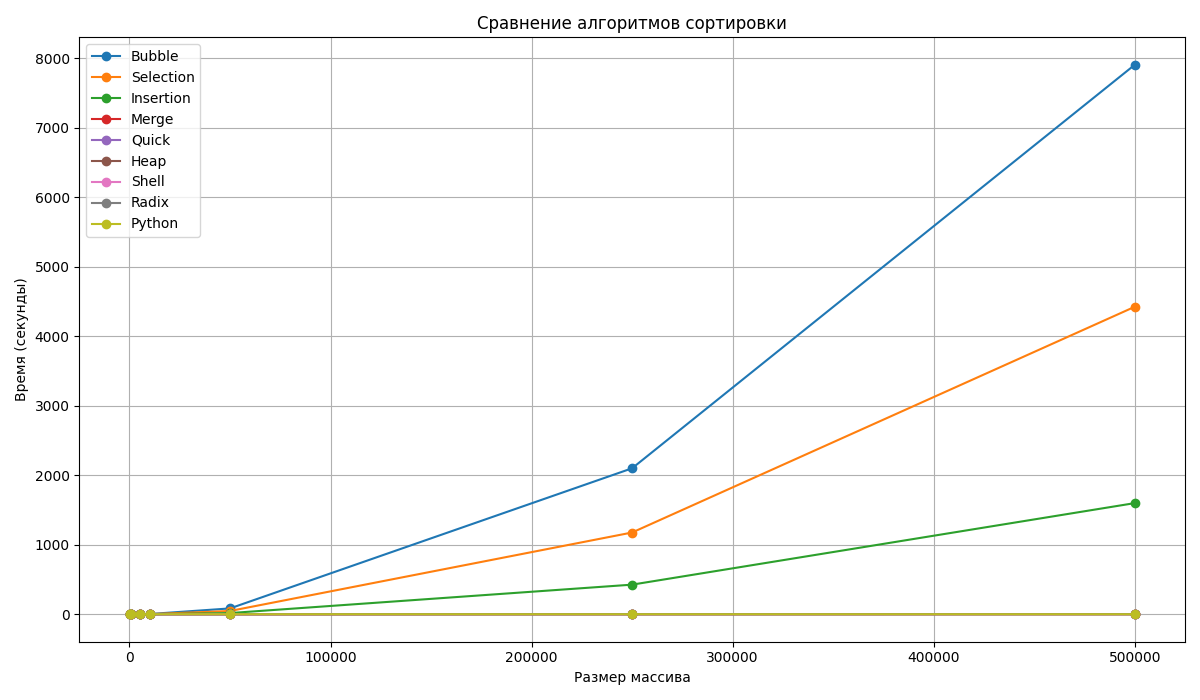
Тип string:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Размер** | **Bubble** | **Selection** | **Insertion** | **Merge** | **Quick** | **Heap** | **Shell** | **Radix** | **Python** |
| **100** | 0,00029 | 0,00017 | 0,00013 | 0,00012 | 0,0001 | 0,00012 | 6,5E-05 | 0,00021 | 0,00001 |
| **500** | 0,00834 | 0,00453 | 0,00377 | 0,00063 | 0,00074 | 0,00082 | 0,00055 | 0,00089 | 0,00007 |
| **1000** | 0,04175 | 0,01863 | 0,01738 | 0,00146 | 0,00131 | 0,00206 | 0,00157 | 0,0015 | 0,00012 |
| **5000** | 1,00526 | 0,49348 | 0,46096 | 0,00981 | 0,00928 | 0,01344 | 0,01139 | 0,00819 | 0,00074 |
| **10000** | 4,18974 | 1,87638 | 1,84519 | 0,0193 | 0,01775 | 0,02613 | 0,02548 | 0,01396 | 0,00155 |
| **50000** | 106,675 | 48,7261 | 50,8953 | 0,10388 | 0,10016 | 0,18079 | 0,19308 | 0,08931 | 0,01006 |
| **250000** | 2666,88 | 1218,15 | 1272,38 | 0,51938 | 0,50078 | 0,90393 | 0,96539 | 0,44654 | 0,05028 |
| **500000** | 10667,5 | 4872,61 | 5089,53 | 1,03876 | 1,00155 | 1,80786 | 1,93077 | 0,89308 | 0,10055 |



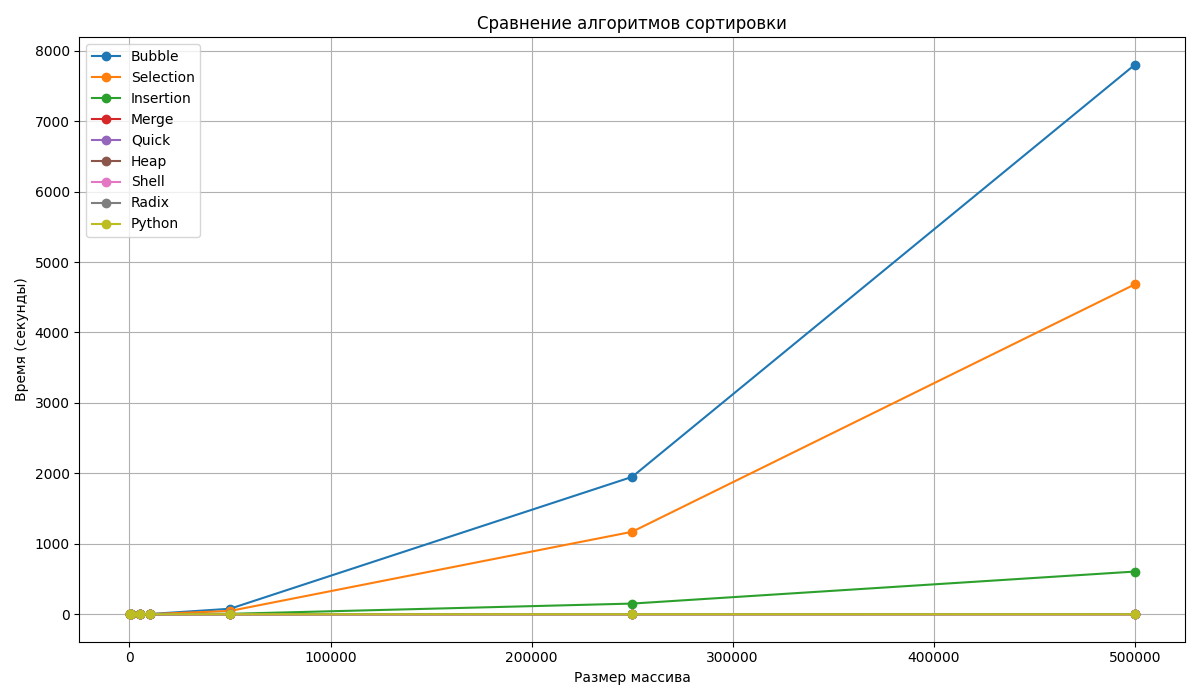
1. Случайный для типа string

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Размер** | **Bubble** | **Selection** | **Insertion** | **Merge** | **Quick** | **Heap** | **Shell** | **Radix** | **Python** |
| **100** | 0,00023 | 0,00028 | 0,00013 | 0,00012 | 0,00016 | 0,00024 | 5,9E-05 | 0,00023 | 9E-06 |
| **500** | 0,00674 | 0,00473 | 0,00168 | 0,00056 | 0,0005 | 0,00089 | 0,00046 | 0,00097 | 2,6E-05 |
| **1000** | 0,02935 | 0,01922 | 0,00681 | 0,00109 | 0,00106 | 0,00189 | 0,00114 | 0,00166 | 4,8E-05 |
| **5000** | 0,78272 | 0,46835 | 0,16205 | 0,00601 | 0,00622 | 0,01214 | 0,00854 | 0,00655 | 0,00022 |
| **10000** | 3,16612 | 1,91911 | 0,68525 | 0,01281 | 0,01272 | 0,02649 | 0,01975 | 0,01412 | 0,00044 |
| **50000** | 83,9819 | 47,0477 | 17,0213 | 0,07125 | 0,07255 | 0,16479 | 0,14055 | 0,07622 | 0,00264 |
| **250000** | 2100,13 | 1176,12 | 425,945 | 0,45347 | 0,4535 | 1,03349 | 0,88343 | 0,48235 | 0,013 |
| **500000** | 7905,3 | 4423,97 | 1598,99 | 0,8924 | 0,89335 | 2,06739 | 1,78596 | 0,94125 | 0,02523 |



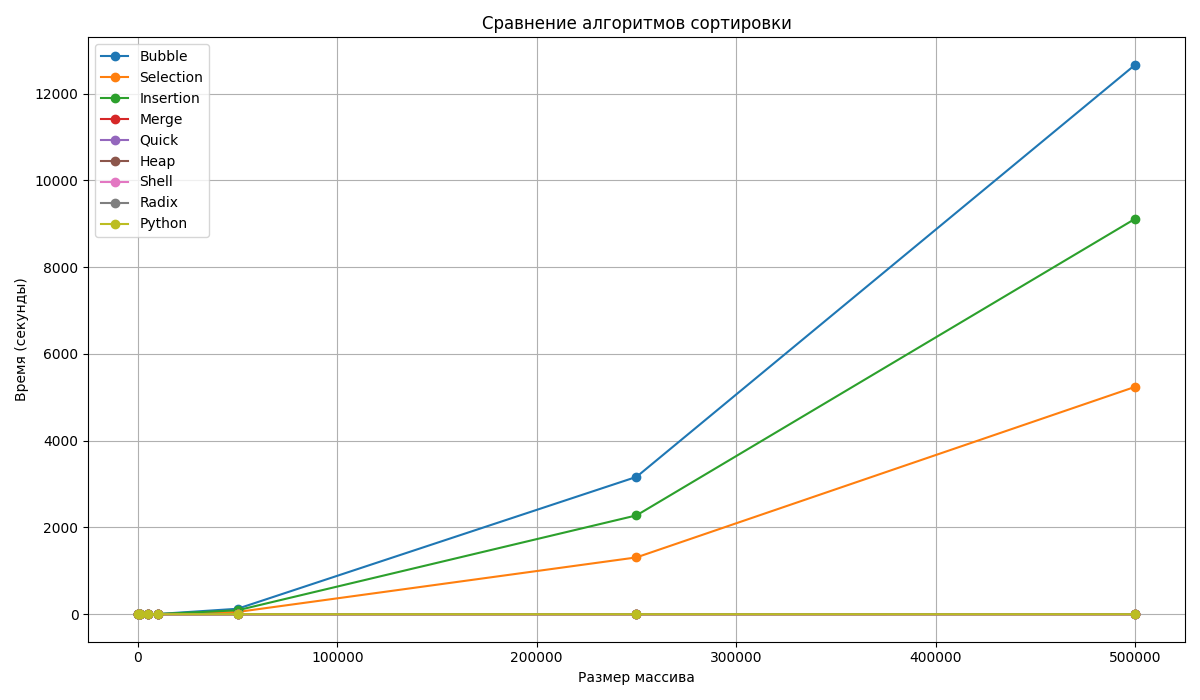
1. Unsorted tail для string

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Размер** | **Bubble** | **Selection** | **Insertion** | **Merge** | **Quick** | **Heap** | **Shell** | **Radix** | **Python** |
| **100** | 0,00021 | 0,00029 | 4,9E-05 | 0,00016 | 0,00012 | 0,00024 | 0,00009 | 0,00023 | 9E-06 |
| **500** | 0,00576 | 0,00448 | 0,00069 | 0,00071 | 0,00052 | 0,00097 | 0,00045 | 0,0008 | 0,00004 |
| **1000** | 0,02641 | 0,01925 | 0,00223 | 0,00124 | 0,00103 | 0,00218 | 0,00101 | 0,00155 | 7,3E-05 |
| **5000** | 0,70199 | 0,46651 | 0,06268 | 0,00772 | 0,0059 | 0,01205 | 0,00815 | 0,00683 | 0,00034 |
| **10000** | 2,86921 | 1,93858 | 0,25795 | 0,01641 | 0,01255 | 0,02881 | 0,01873 | 0,01358 | 0,00066 |
| **50000** | 77,9909 | 46,8208 | 6,05925 | 0,09302 | 0,06772 | 0,16197 | 0,14277 | 0,08539 | 0,00378 |
| **250000** | 1949,77 | 1170,52 | 151,481 | 0,46511 | 0,33862 | 0,80984 | 0,71384 | 0,42697 | 0,01888 |
| **500000** | 7799,09 | 4682,08 | 605,925 | 0,93021 | 0,67723 | 1,61968 | 1,42768 | 0,85394 | 0,03775 |



1. Partially sorted для string

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Размер** | **Bubble** | **Selection** | **Insertion** | **Merge** | **Quick** | **Heap** | **Shell** | **Radix** | **Python** |
| **100** | 0,00039 | 0,0003 | 0,00054 | 0,00013 | 7,7E-05 | 0,0002 | 6,5E-05 | 0,00024 | 4E-06 |
| **500** | 0,00963 | 0,00506 | 0,00743 | 0,0005 | 0,00046 | 0,00073 | 0,00041 | 0,00085 | 9E-06 |
| **1000** | 0,04368 | 0,0203 | 0,03195 | 0,0011 | 0,00087 | 0,00169 | 0,00097 | 0,00147 | 1,7E-05 |
| **5000** | 1,22506 | 0,51459 | 0,88103 | 0,00555 | 0,00522 | 0,01186 | 0,00686 | 0,00692 | 5,1E-05 |
| **10000** | 5,14149 | 2,2025 | 3,89622 | 0,01238 | 0,01111 | 0,02419 | 0,01526 | 0,01507 | 0,00011 |
| **50000** | 126,662 | 52,4239 | 91,1646 | 0,06937 | 0,07255 | 0,15282 | 0,08647 | 0,07814 | 0,00059 |
| **250000** | 3166,55 | 1310,6 | 2279,11 | 0,34686 | 0,36276 | 0,76411 | 0,43237 | 0,39068 | 0,00297 |
| **500000** | 12666,2 | 5242,39 | 9116,46 | 0,69371 | 0,72551 | 1,52822 | 0,86473 | 0,78136 | 0,00593 |



1. Reverse sorted для string

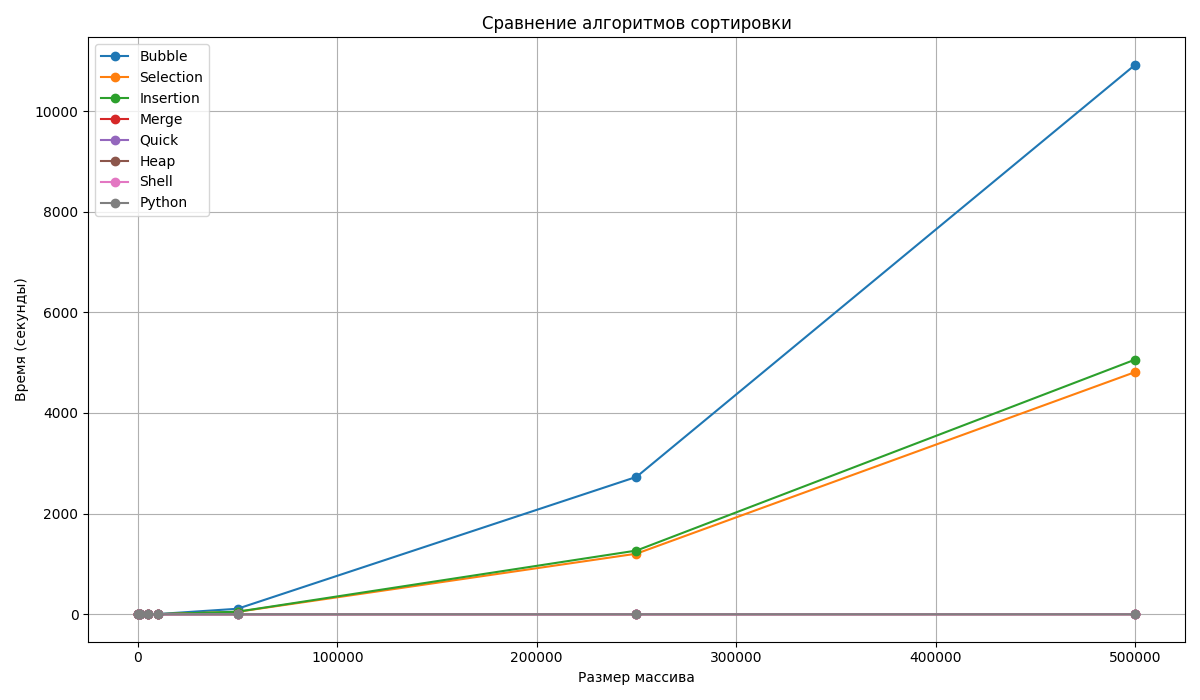
Python — Timsort. Он лидирует по производительности на всех типах строковых массивов благодаря своим оптимизациям, учитывающим частичную упорядоченность и специфику строковых сравнений. Это делает его универсальным решением для большинства задач сортировки строк.

Radix Sort хорошо справляется с полностью случайными строками со скоростью, близкую к Merge Sort, однако на частично упорядоченных и обратно отсортированных данных уступает Timsort и Quick Sort. Особенность Radix Sort сравнивает строки посимвольно из-за чего скорость сортировки может упасть.

Квадратичные алгоритмы — Bubble, Selection и Insertion Sort — полностью неэффективны при работе с большими строковыми массивами.

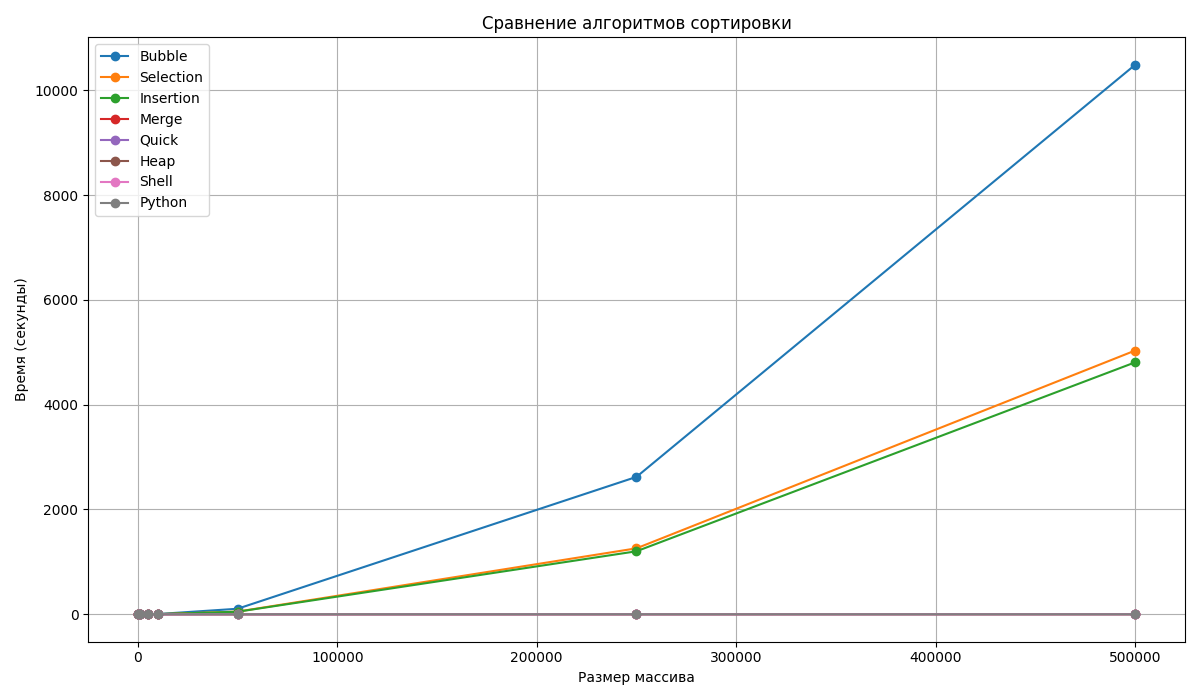
Тип date:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Размер** | **Bubble** | **Selection** | **Insertion** | **Merge** | **Quick** | **Heap** | **Shell** | **Python** |
| **100** | 0,00028 | 0,00027 | 0,00019 | 0,00015 | 0,00016 | 0,00012 | 6,6E-05 | 1,2E-05 |
| **500** | 0,00816 | 0,00457 | 0,00406 | 0,00072 | 0,00067 | 0,00082 | 0,00056 | 6,5E-05 |
| **1000** | 0,0358 | 0,01812 | 0,01706 | 0,00137 | 0,00137 | 0,00186 | 0,00131 | 0,00015 |
| **5000** | 1,01387 | 0,46845 | 0,46264 | 0,00838 | 0,00803 | 0,01172 | 0,00994 | 0,00085 |
| **10000** | 3,95016 | 1,83705 | 1,79327 | 0,01716 | 0,0158 | 0,02522 | 0,02294 | 0,00211 |
| **50000** | 109,188 | 48,1182 | 50,5994 | 0,102 | 0,08171 | 0,15989 | 0,17127 | 0,01161 |
| **250000** | 2730,07 | 1203,84 | 1265,63 | 0,59754 | 0,47579 | 0,93214 | 0,99985 | 0,06769 |
| **500000** | 10919,2 | 4812,97 | 5060,96 | 1,18854 | 0,95956 | 1,85084 | 1,99854 | 0,13503 |



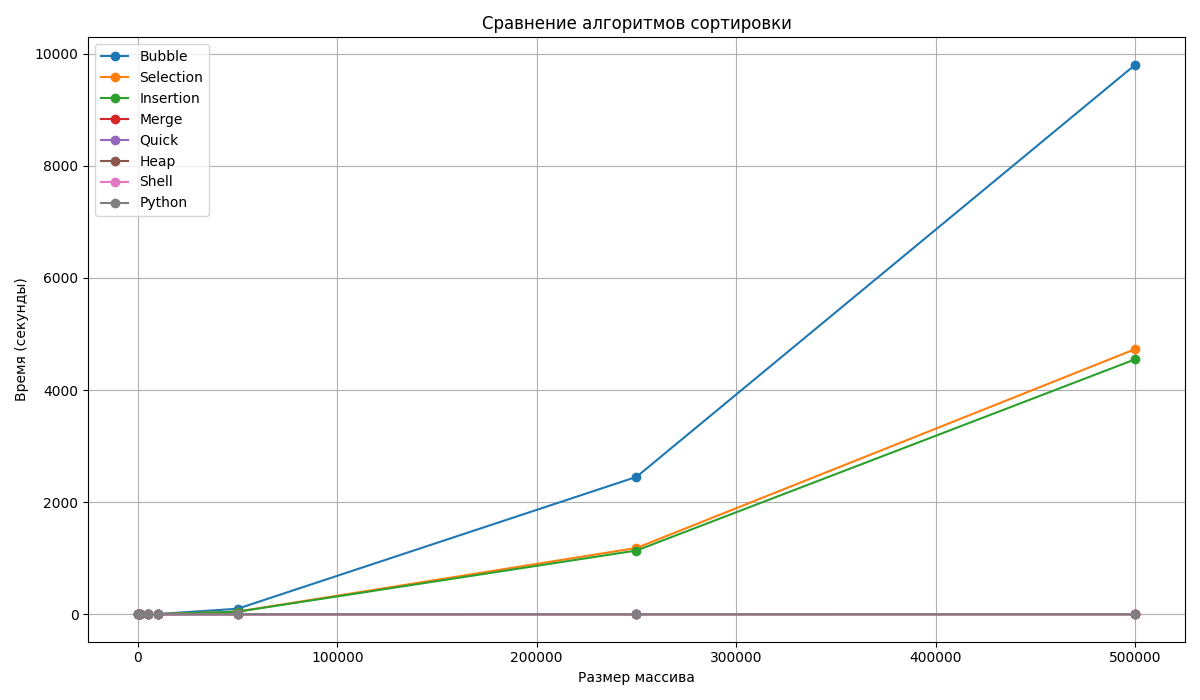
1. Случайный для типа date

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Размер** | **Bubble** | **Selection** | **Insertion** | **Merge** | **Quick** | **Heap** | **Shell** | **Python** |
| **100** | 0,00032 | 0,00017 | 0,00019 | 0,00018 | 0,00015 | 0,00013 | 8,6E-05 | 1,6E-05 |
| **500** | 0,00843 | 0,00445 | 0,00397 | 0,00092 | 0,00075 | 0,00082 | 0,00054 | 8,8E-05 |
| **1000** | 0,03655 | 0,01862 | 0,01853 | 0,00136 | 0,00215 | 0,00181 | 0,00124 | 0,00014 |
| **5000** | 1,07734 | 0,50728 | 0,48971 | 0,0084 | 0,00886 | 0,01609 | 0,0182 | 0,0008 |
| **10000** | 4,01364 | 1,85405 | 1,8631 | 0,01705 | 0,01978 | 0,02576 | 0,02517 | 0,00142 |
| **50000** | 104,896 | 50,3453 | 48,0943 | 0,09414 | 0,11937 | 0,16502 | 0,22274 | 0,00471 |
| **250000** | 2622,4 | 1258,63 | 1202,36 | 0,47071 | 0,59685 | 0,82509 | 1,11368 | 0,02357 |
| **500000** | 10489,6 | 5034,53 | 4809,43 | 0,94142 | 1,1937 | 1,65018 | 2,22736 | 0,04714 |



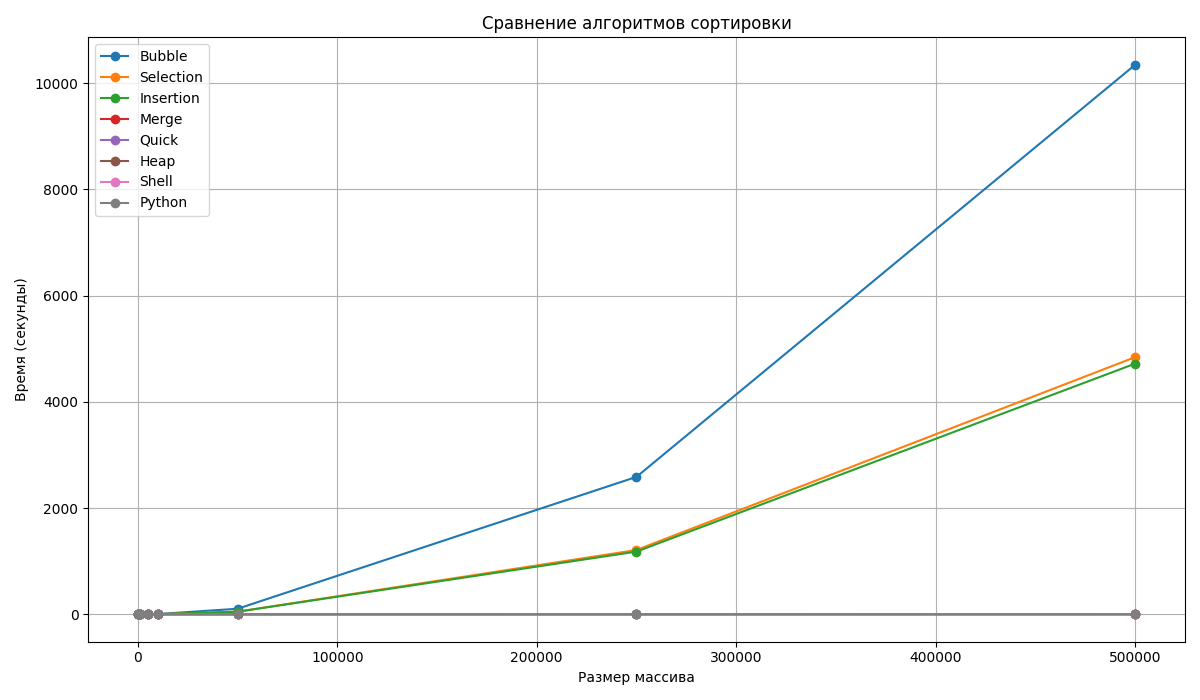
1. Reverse sorted для date

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Размер** | **Bubble** | **Selection** | **Insertion** | **Merge** | **Quick** | **Heap** | **Shell** | **Python** |
| **100** | 0,00031 | 0,00025 | 0,00037 | 0,00027 | 0,0004 | 0,00017 | 9,1E-05 | 1,6E-05 |
| **500** | 0,00797 | 0,00453 | 0,00385 | 0,00069 | 0,00097 | 0,00082 | 0,00059 | 6,6E-05 |
| **1000** | 0,03703 | 0,01853 | 0,01673 | 0,00134 | 0,00145 | 0,00187 | 0,00125 | 0,00014 |
| **5000** | 1,04207 | 0,47168 | 0,44773 | 0,00778 | 0,00819 | 0,01178 | 0,01028 | 0,00076 |
| **10000** | 4,29764 | 1,93209 | 2,01498 | 0,01666 | 0,01636 | 0,02543 | 0,02773 | 0,00156 |
| **50000** | 98,0199 | 47,2928 | 45,4783 | 0,09473 | 0,07843 | 0,15635 | 0,18782 | 0,00768 |
| **250000** | 2450,5 | 1182,32 | 1136,96 | 0,47365 | 0,39213 | 0,78175 | 0,9391 | 0,03839 |
| **500000** | 9801,99 | 4729,28 | 4547,83 | 0,94729 | 0,78425 | 1,56349 | 1,87819 | 0,07677 |



1. Partially sorted для date

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Размер** | **Bubble** | **Selection** | **Insertion** | **Merge** | **Quick** | **Heap** | **Shell** | **Python** |
| **100** | 0,00028 | 0,00024 | 0,00028 | 0,00013 | 0,00014 | 0,00013 | 6,2E-05 | 1,3E-05 |
| **500** | 0,00825 | 0,00455 | 0,00397 | 0,00064 | 0,00065 | 0,00078 | 0,00069 | 6,6E-05 |
| **1000** | 0,03624 | 0,01941 | 0,01672 | 0,00131 | 0,0014 | 0,0018 | 0,00141 | 0,00016 |
| **5000** | 0,96928 | 0,47871 | 0,44843 | 0,00791 | 0,00858 | 0,01162 | 0,01068 | 0,00077 |
| **10000** | 4,01863 | 1,84308 | 1,83716 | 0,0165 | 0,01852 | 0,02545 | 0,02511 | 0,00157 |
| **50000** | 103,43 | 48,3898 | 47,1803 | 0,10103 | 0,1057 | 0,15873 | 0,20246 | 0,0065 |
| **250000** | 2585,74 | 1209,75 | 1179,51 | 0,50513 | 0,52852 | 0,79364 | 1,01231 | 0,03251 |
| **500000** | 10343 | 4839,98 | 4718,03 | 1,01026 | 1,05704 | 1,58728 | 2,02461 | 0,06502 |



1. Unsorted для date

В общем итоге по результатам тестирования сортировок на датах можно выделить следующие ключевые выводы. Встроенный алгоритм Python — Timsort — все так же является абсолютным лидером по скорости для всех типов данных: случайных, частично упорядоченных, обратно отсортированных и с неотсортированным хвостом.

Radix Sort в данных тестах не рассматривался, так как он требует дополнительной адаптации для сортировки дат, это неэффективно

Из классических алгоритмов Quick Sort и Merge Sort показывают сопоставимые результаты, однако Quick Sort чаще выигрывает на случайных данных, тогда как Merge Sort проявляет большую стабильность на обратно отсортированных массивах. Квадратичные алгоритмы — Bubble, Selection и Insertion Sort — оказались непригодными для больших наборов данных свыше 10 тысяч элементов из-за крайне высокой временной сложности.

В ходе экспериментов была проведена оценка времени работы различных алгоритмов сортировки на разных типах данных и объемах выборок. Полученные результаты сравнивались с классическими теоретическими оценками временной сложности:

Квадратичные алгоритмы (Bubble Sort, Selection Sort, Insertion Sort) продемонстрировали рост времени работы приблизительно пропорционально O(n2), что подтверждает ожидаемое поведение на больших объёмах данных — время резко увеличивается при росте размера массива.

Быстрые алгоритмы (Quick Sort, Merge Sort) показали время работы, близкое к O(nlogn), что соответствует их теоретической сложности. При этом Quick Sort в среднем работал быстрее, но в худших случаях (обратно отсортированные данные) наблюдалось падение скорости

Radix Sort продемонстрировал линейную сложность O(n) при сортировке числовых данных с фиксированной разрядностью, что соответствует теоретическим предположениям, но при работе со строками и частично отсортированными данными возникали проблемы.

Таким образом, экспериментально подтверждается, что реализованные алгоритмы в целом соответствуют своим теоретическим временным характеристикам, что говорит о корректности реализации.

Заключение

Встроенные и адаптивные алгоритмы Python Timsort стабильно показывает лучшие или близкие к лучшим результаты на большинстве типов данных и структур — будь то числа, строки или даты. Это связано с его адаптивностью к уже частично отсортированным данным, оптимизациям для строк и высокой стабильности. Поэтому для большинства практических задач лучше всего использовать встроенную сортировку Python.

Алгоритмы с линейной или близкой к линейной сложностью Radix Sort отлично работает на больших наборах числовых данных с фиксированной разрядностью, показывая высокую производительность, зачастую превосходящую классические алгоритмы, особенно на случайных и неотсортированных данных. Однако для строк и дат этот алгоритм требует существенной доработки, и на этих типах данных он уступает адаптивным алгоритмам.

Классические алгоритмы сравнения Quick Sort и Merge Sort — самые универсальные классические алгоритмы с хорошей производительностью. Quick Sort зачастую быстрее на случайных данных, но менее стабилен. Merge Sort устойчив, но может работать медленнее. Heap Sort находится в среднем по скорости и используется реже.

Квадратичные алгоритмы подходят только для очень маленьких наборов данных (до нескольких тысяч элементов). При больших объемах их использование нецелесообразно из-за резко возрастающего времени работы.

Для чисел и дат лучше всего подходят Timsort, Radix Sort (для чисел), Quick Sort и Merge Sort.

Для строк наилучший выбор — Timsort благодаря его оптимизациям под частичную упорядоченность и специфику строк. Radix Sort для строк требует дополнительной адаптации и не всегда оправдан.

Частично отсортированные данные выгодно использовать с адаптивными алгоритмами (Timsort, Insertion Sort для малых объемов), а плохо отсортированные лучше сортировать классическими эффективными алгоритмами.

Приложение 1. Программный код

Весь проект опубликован на GitHub: [obratno123/sort](https://github.com/obratno123/sort)