МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
[Полное название вуза]

Факультет информационных технологий
Кафедра программной инженерии

ЛАБОРАТОРНАЯ РАБОТА № 4

по дисциплине «Алгоритмы и структуры данных»

на тему: «Экспериментальная оценка временной сложности алгоритмов сортировки»

Выполнил(а): студент гр. ____

_____ / И. О. Фамилия

Проверил(а): _____

_____ / И. О. Фамилия

[Город] 2026

Пример работы, выполненной в Универ AI · univerai.ru

1. Цель работы

Экспериментально проверить, как время работы трёх алгоритмов сортировки растёт с размером массива: сортировки вставками, сортировки слиянием и встроенной функции sorted() языка Python. По измерениям нужно оценить показатель степени в зависимости $t \sim n^p$, сравнить его с теоретической оценкой сложности и выяснить, насколько надёжны сами замеры времени.

В отличие от многих учебных отчётов, здесь все времена **измерены на самом деле**: программа из раздела 4 запускалась 22.10.2026 на виртуальной машине, описанной в таблице 1. На другом компьютере абсолютные числа будут другими, но отношения времён и показатели степени должны получиться близкими.

2. Оборудование и программное обеспечение

Таблица 1 – Условия эксперимента

Параметр	Значение
Процессор	Intel Xeon, 2,80 ГГц, 4 логических ядра (виртуальная машина)
Оперативная память	16 ГБ
Операционная система	Linux 6.18, x86-64
Интерпретатор	CPython 3.13.16
Таймер	time.perf_counter (по умолчанию в модуле timeit)
Средняя загрузка системы	2,2–2,4 по показанию uptime до и после замеров

Примечание – параметры машины, на которой выполнялись реальные измерения.

Последняя строка таблицы важна. Средняя загрузка около 2,3 при четырёх ядрах означает, что параллельно с нашей программой на машине работали другие процессы. Они отнимали процессорное время и создавали разброс замеров, который мы увидим в разделе 5.

3. Краткая теория

Время работы алгоритма обычно описывают асимптотической оценкой — функцией от размера входа n, с точностью до постоянного множителя [3]. Сортировка вставками в среднем и худшем случае выполняет порядка n^2 операций, а на уже упорядоченном массиве — порядка n. Сортировка слиянием всегда тратит порядка $n \cdot \log_2 n$ операций. Встроенная sorted() в CPython реализует алгоритм Timsort, гибрид слияния и вставок, который ищет в данных уже упорядоченные участки и в худшем случае также работает за O(n·log n) [6]. Строго эту оценку доказали лишь в 2018 году, заодно получив O(n + n·log ρ), где ρ — число упорядоченных участков [4].

Для сортировки вставками можно точно посчитать среднее число сравнений. Каждый новый элемент в среднем сдвигается на половину уже отсортированной части, поэтому [2]:

$$C_{ins}(n) \approx \frac{n^2}{4}, \tag{1}$$

где Cins — среднее число сравнений на случайном массиве из n различных элементов.

У сортировки слиянием число сравнений не превышает $n \cdot \log_2 n$, а на случайных данных оказывается немного меньше этой величины [2; 3].

Если $t \approx c \cdot n^p$, то после логарифмирования получаем прямую $\lg t = \lg c + p \cdot \lg n$. Значит, показатель p можно найти как угловой коэффициент прямой в двойном логарифмическом масштабе:

$$p = \frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{\sum (x_i - \bar{x})^2}, \qquad x_i = \lg n_i,\ y_i = \lg t_i. \tag{2}$$

где n_i — размер массива, t_i — измеренное время.

Для n·log n наклон не равен ровно единице. Локально он составляет 1 + 1/ln n, то есть при n от 500 до 8000 около 1,13. Ещё удобнее смотреть на

отношение времён при удвоении размера: для n^2 оно равно 4, для n — 2, для $n \cdot \log_2 n$ — $2 \cdot \log_2(2n)/\log_2 n$, то есть 2,17–2,22 в нашем диапазоне.

Каждый размер измеряли R = 7 раз. Для серии вычисляли среднее, выборочное стандартное отклонение s и доверительную границу случайной погрешности при P = 0,95 [1]:

$$\Delta \bar{t} = t_{0,95;\,R} \frac{s}{\sqrt{R}}, \qquad \delta = \frac{\Delta \bar{t}}{\bar{t}} \cdot 100\,\%, \tag{3}$$

где t(0,95; 7) = 2,45 — коэффициент Стьюдента; δ — относительная погрешность.

Документация модуля timeit предупреждает, что среднее и стандартное отклонение по повторам «не очень полезны»: разброс создаёт не Python, а другие процессы, и самое информативное число — минимум [5]. Поэтому в отчёте приведены и минимум, и медиана, а для оценки показателя степени взята медиана как величина, устойчивая к редким выбросам.

4. Ход работы

Алгоритмы вставок и слияния написаны на чистом Python по классическим схемам [3]. Обе функции возвращают новый список и не меняют исходный, так же как sorted(). Код приведён в приложении А (листинг А.1). Перед замерами программа проверила, что все три способа дают одинаковый результат на каждом массиве.

Массивы из 500, 1000, 2000, 4000 и 8000 случайных чисел типа float сгенерированы один раз с зерном 2026. Для сортировки вставками отдельно измерен лучший случай — заранее упорядоченный массив.

Первый вариант программы измерял размеры по порядку, от меньшего к большему. Результаты вышли неровными: например, время слияния при переходе от 2000 к 4000 элементам выросло всего в 1,4 раза, хотя теория требует около 2,2. Причина в том, что фоновая нагрузка на машину менялась во времени и по-разному задевала разные размеры. Во втором варианте

порядок замеров перемешивался в каждом из семи повторов, а число вызовов подбиралось так, чтобы один замер длился не меньше 5 мс. Схема замеров показана в листинге А.2. Эти данные и обработаны ниже. Первый прогон в отчёт не вошёл, но его итог полезен как предупреждение: последовательные замеры на общей машине могут обмануть даже в простом опыте, где теория заранее известна и ошибку легко заметить.

5. Результаты измерений

Медианные времена всех серий сведены в таблицу 2. Главное видно сразу. Сортировка вставками при росте n в 16 раз замедлилась примерно в 310 раз, а слияние — только в 22 раза.

Таблица 2 – Медиана времени сортировки по 7 повторам, мс

n	Вставки, случайный	Вставки, упорядоченный	Слияние	sorted()
500	5,163	0,0456	0,692	0,0162
1000	22,20	0,0936	1,470	0,0604
2000	93,80	0,2023	3,389	0,1963
4000	408,7	0,4027	7,172	0,4168
8000	1605	0,7848	15,22	0,9368

Примечание – реальные измерения, CPython 3.13.16, 22.10.2026; условия – в таблице 1.

Таблица 3 показывает разброс повторов для случайных массивов. Относительная погрешность среднего колеблется от 2 до 32 % и почти не связана с размером задачи. Больше всего пострадали короткие замеры и серии, в которые попал один-два «медленных» повтора.

Таблица 3 – Среднее и доверительная граница по 7 повторам (P = 0,95), мс

n	Вставки $\bar{t} \pm \Delta$	δ, %	Слияние $\bar{t} \pm \Delta$	δ, %	sorted() $\bar{t} \pm \Delta$	δ, %
500	5,24 ± 0,16	3,1	0,87 ± 0,23	26	0,020 ± 0,006	32
1000	25,2 ± 4,5	17,7	1,55 ± 0,21	13,8	0,0607 ± 0,0021	3,4
2000	95,3 ± 3,9	4,1	3,65 ± 0,83	22,7	0,195 ± 0,023	11,8
4000	406 ± 23	5,6	7,22 ± 0,23	3,2	0,448 ± 0,060	13,3
8000	1643 ± 78	4,7	15,30 ± 0,30	2,0	1,02 ± 0,13	12,9

Примечание – реальные измерения; $\Delta = 2{,}45 \cdot s/\sqrt{7}$.

Пример серии с выбросом — вставки при n = 1000. Минимум 21,17 мс, медиана 22,20 мс, а максимум 31,79 мс. Один-два долгих повтора подняли среднее до 25,2 мс и раздули погрешность до 18 %. По медиане эта серия ничем не выделяется, что и оправдывает выбор медианы для дальнейших расчётов.

6. Обработка результатов

6.1 Отношение времён при удвоении размера

Таблица 4 – Отношение t(2n) / t(n) по медианам

Переход n → 2n	**Вставки**	**Вставки, упорядоченный**	**Слияние**	**sorted()**	**Теория для $n \cdot \log_2 n$**
500 → 1000	4,30	2,05	2,12	3,72	2,22
1000 → 2000	4,23	2,16	2,30	3,25	2,20
2000 → 4000	4,36	1,99	2,12	2,12	2,18
4000 → 8000	3,93	1,95	2,12	2,25	2,17

Примечание – рассчитано по таблице 2; для n^2 теоретическое отношение равно 4, для n – 2.

Вставки на случайных данных дают отношение 3,9–4,4, то есть квадратичный рост. На упорядоченном массиве отношение около 2: внутренний цикл сразу завершается, и алгоритм просто один раз проходит по массиву. Слияние держится у 2,1–2,3, рядом с теоретическими 2,17–2,22. С функцией sorted() всё сложнее. На малых размерах отношение 3,3–3,7, и лишь начиная с 2000 элементов оно опускается до 2,1–2,3.

6.2 Показатель степени по методу наименьших квадратов

По пяти точкам (lg n; lg t) для каждого алгоритма провели прямую по формуле (2). Стандартную ошибку наклона умножали на коэффициент Стьюдента t(0,95; 3) = 3,18, так как у прямой по пяти точкам три степени свободы.

Таблица 5 – Оценка показателя p в зависимости $t \sim n^p$

Алгоритм	p	Δp (P = 0,95)	Теоретическое значение
Вставки, случайный массив	2,08	0,04	2
Вставки, упорядоченный массив	1,03	0,05	1
Слияние	1,12	0,04	≈ 1,13 (n·log n)
sorted()	1,45	0,33	≈ 1,13 (n·log n)

Примечание – расчёт по медианам таблицы 2 (реальные измерения).

Для вставок и слияния результат совпал с теорией в пределах погрешности. Хороший итог. Показатель вставок 2,08 ± 0,04 чуть больше двух; причину покажет таблица 6: время одного сравнения растёт с размером массива. Все четыре серии в двойном логарифмическом масштабе показаны на рисунке 1. У sorted() показатель 1,45 с огромной погрешностью, потому что точки не лежат на одной прямой. Если отбросить n = 500 и n = 1000, наклон по оставшимся точкам будет около 1,1.

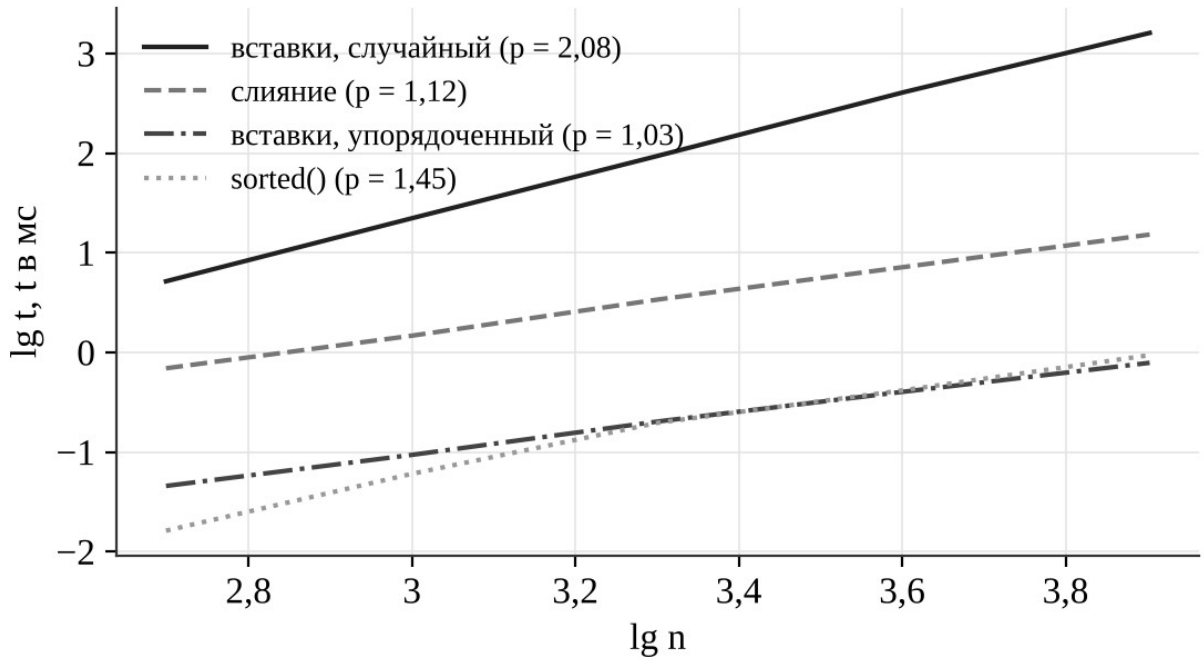


Рисунок 1 – Время сортировки в двойном логарифмическом масштабе

Построено по медианам таблицы 2 (реальные измерения).

6.3 Время на одну операцию

Время можно сопоставить с числом выполненных сравнений, которое программа подсчитала отдельно, без замера времени, на тех же массивах.

Таблица 6 – Число сравнений и время на одно сравнение

n	**Сравнений (вставки)**	**$n^2/4$**	**Время на сравнение, нс**	**Сравнений (слияние)**	**$n \cdot \log_2 n$**
500	62 886	62 500	82	3 846	4 483
1000	243 629	250 000	91	8 684	9 966
2000	990 066	1 000 000	95	19 370	21 932
4000	4 012 504	4 000 000	102	42 860	47 863
8000	15 730 930	16 000 000	102	93 616	103 726

Примечание – число сравнений подсчитано программой точно; время на сравнение – медиана из таблицы 2, делённая на число сравнений вставок.

Число сравнений у вставок отличается от $n^2/4$ не больше чем на 2,5 %, как и предсказывает формула (1). У слияния сравнений на 10–14 % меньше, чем $n \cdot \log_2 n$: при слиянии двух половин одна из них часто заканчивается раньше, и хвост второй копируется без сравнений.

Время одного сравнения у вставок выросло с 82 до 102 нс. Рост шёл ступенями и к 4000 элементам остановился. Список из 4000 ссылок вместе с самими объектами float занимает уже больше сотни килобайт. Почему? Вероятное объяснение — данные перестают помещаться в быстрый кэш процессора, и каждое обращение к элементу дорожает. Этим же можно объяснить и рост удельного времени sorted() на малых n: при n = 500 на элемент приходится 3,6 нс на один шаг $n \cdot \log_2 n$, а начиная с 2000 — около 9 нс. Проверять эту гипотезу на виртуальной машине с чужой нагрузкой мы не стали; её стоило бы проверить отдельным опытом.

7. Вывод

Измерения подтвердили теоретические оценки сложности. Сортировка вставками на случайных данных растёт как n^2 (показатель 2,08 ± 0,04, отношение при удвоении около 4), на упорядоченных — линейно (1,03 ± 0,05). Сортировка слиянием показала наклон 1,12 ± 0,04 при теоретическом 1,13 для $n \cdot \log n$.

Разница в асимптотике быстро превращается в огромную разницу во времени. И чем больше n, тем она заметнее. При n = 8000 вставки работали 1,6 с, слияние — 15 мс, встроенная sorted() — меньше 1 мс. Слияние и

sorted() относятся к одному классу сложности, но sorted() примерно в 16 раз быстрее, так как написана на C и пользуется готовыми упорядоченными участками данных.

Отдельный вывод касается самих замеров. На загруженной машине погрешность среднего доходила до 20–30 %, а замеры подряд по возрастанию n исказили картину. Перемешивание порядка замеров, медиана вместо среднего и длительность одного замера не меньше нескольких миллисекунд сделали результаты воспроизводимыми.

8. Контрольные вопросы

1. Почему по графику в логарифмическом масштабе можно определить порядок сложности? Степенная зависимость $t = c \cdot n^p$ после логарифмирования превращается в прямую с наклоном p. Постоянный множитель c, который зависит от компьютера и языка, уходит в свободный член и на наклон не влияет.

2. Почему сортировка вставками на упорядоченном массиве работает за линейное время? Для каждого нового элемента внутренний цикл выполняет одно сравнение и сразу останавливается. Сдвигов нет. Всего получается $n - 1$ сравнение.

3. Почему функция sorted() быстрее сортировки слиянием, если их сложность одинакова? Асимптотика скрывает постоянный множитель. А он велик. Встроенная функция написана на C и не тратит время на интерпретацию каждого шага. Вдобавок Timsort находит в массиве уже упорядоченные участки и для коротких участков использует вставки [6].

4. Почему документация timeit советует смотреть на минимум, а не на среднее? Время выполнения кода не может стать меньше «чистого» времени, а все задержки добавляются сверху другими процессами. Минимум ближе всего к собственному времени кода, среднее же зависит от случайной фоновой нагрузки [5].

5. Зачем перемешивать порядок замеров? Фоновая нагрузка меняется во времени, иногда за доли секунды. Если измерять размеры по очереди, медленный период целиком попадёт на один размер и исказит его. При перемешивании помехи распределяются по всем сериям равномерно и превращаются в случайную погрешность, которую видно по разбросу повторов.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. ГОСТ Р 8.736–2011. Государственная система обеспечения единства измерений. Измерения прямые многократные. Методы обработки результатов измерений. Основные положения : национальный стандарт Российской Федерации : дата введения 2013–01–01. – URL: https://cn.gostinfo.ru/catalog/Details/?id=4682351 (дата обращения: 22.10.2026). – Текст : электронный.

2. Кнут, Д. Э. Искусство программирования. Т. 3. Сортировка и поиск / Д. Э. Кнут. – 2-е изд. – Москва : Вильямс, 2007. – 824 с.

3. Кормен, Т. Алгоритмы. Построение и анализ / Т. Кормен, Ч. Лейзерсон, Р. Ривест, К. Штайн ; пер. с англ. – 3-е изд. – Москва ; Санкт-Петербург : Диалектика, 2019. – 1323 с. – ISBN 978-5-907114-11-1.

4. Auger, N. On the worst-case complexity of TimSort / N. Auger, V. Jugé, C. Nicaud, C. Pivoteau // 26th Annual European Symposium on Algorithms (ESA 2018). – Dagstuhl : Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. – P. 4:1–4:13. – (LIPIcs ; vol. 112). – DOI 10.4230/LIPIcs.ESA.2018.4.

5. timeit — Measure execution time of small code snippets // Python 3 documentation. – URL: https://docs.python.org/3/library/timeit.html (дата обращения: 22.10.2026). – Текст : электронный.

6. listsort.txt : описание алгоритма сортировки списков CPython (Timsort) // GitHub : репозиторий python/cpython. – URL: https://github.com/python/cpython/blob/main/Objects/listsort.txt (дата обращения: 22.10.2026). – Текст : электронный.

ПРИЛОЖЕНИЕ А

Листинги программы

Листинг А.1 – Реализация сортировки вставками и слиянием

```
def insertion_sort(a):
    a = a[:]
    for i in range(1, len(a)):
        x = a[i]; j = i - 1
        while j >= 0 and a[j] > x:
            a[j + 1] = a[j]; j -= 1
        a[j + 1] = x
    return a

def merge_sort(a):
    if len(a) <= 1:
        return a[:]
    m = len(a) // 2
    L, R = merge_sort(a[:m]), merge_sort(a[m:])
    res, i, j = [], 0, 0
    while i < len(L) and j < len(R):
        if L[i] <= R[j]: res.append(L[i]); i += 1
        else: res.append(R[j]); j += 1
    return res + L[i:] + R[j:]
```

Листинг А.2 – Схема замеров с перемешиванием порядка (фрагмент)

```
random.seed(2026)
sizes = (500, 1000, 2000, 4000, 8000)
data = {n: [random.random() for _ in range(n)] for n in sizes}
jobs = [(alg, n) for alg in funcs for n in data]
for r in range(7):                  # 7 повторов
    random.shuffle(jobs)            # порядок замеров каждый раз новый
    for alg, n in jobs:
        N = calls[alg, n]           # замер не короче 5 мс
        t = timeit.timeit(lambda: funcs[alg](n), number=N) / N
        res[alg, n].append(t)
```