Практическое занятие: Сравнительный анализ алгоритмов сортировки

Цель: Экспериментально сравнить время работы алгоритмов сортировки, усреднив результаты 10 запусков для повышения точности измерений.

⚠ Важное предупреждение

Алгоритмы **пузырьком**, **выбором** и **вставками** имеют сложность **$O(n^2)$**.

Элементов	Ожидаемое время (пузырьком, 1 запуск)	Ожидаемое время (10 запусков)
100	~0.001 сек	~0.01 сек
1 000	~0.1 сек	~1 сек
10 000	~10–15 сек	~2–3 **минуты**
100 000	~20–40 **минут** ⚠	~3–7 **часов** ⚠
1 000 000	~20–40 **часов** ⚠	~8–17 **дней** ⚠

Код программы (`sorting_lab.py`)

```
import random
import time

print("=" * 90)
print("СРАВНИТЕЛЬНЫЙ АНАЛИЗ АЛГОРИТМОВ СОРТИРОВКИ (10 запусков, среднее время)")
print("=" * 90)
print("Формат: Размер -> Среднее время за 10 запусков (сек)")
print("Совет: Для 100k и 1M прерывайте квадратичные сортировки (Ctrl+C)\n")

# Параметры теста
sizes = [100, 1000, 10000, 100000, 1000000]
RUNS = 10  # Количество запусков для усреднения

for n in sizes:
    print(f"\n>>> Тест: {n} элементов ({RUNS} запусков)")
    print("-" * 90)

    # ------------------- ПУЗЫРЬКОМ --------------------
    total_time = 0
    for run in range(RUNS):
        # Генерируем новые данные для каждого запуска!
        data = [random.randint(0, 1000000) for _ in range(n)]
        start = time.perf_counter()

        length = len(data)
```

```
        for i in range(length):
            for j in range(0, length - i - 1):
                if data[j] > data[j + 1]:
                    data[j], data[j + 1] = data[j + 1], data[j]

        end = time.perf_counter()
        total_time += (end - start)
    avg_time = total_time / RUNS
    print(f"Пузырьком:  {avg_time:.6f} сек (среднее за {RUNS} запусков)")

    # ------------------- ВЫБОРОМ --------------------
    total_time = 0
    for run in range(RUNS):
        data = [random.randint(0, 1000000) for _ in range(n)]
        start = time.perf_counter()

        length = len(data)
        for i in range(length):
            min_idx = i
            for j in range(i + 1, length):
                if data[j] < data[min_idx]:
                    min_idx = j
            data[i], data[min_idx] = data[min_idx], data[i]

        end = time.perf_counter()
        total_time += (end - start)
    avg_time = total_time / RUNS
    print(f"Выбором:    {avg_time:.6f} сек (среднее за {RUNS} запусков)")


    # ------------------- ВСТАВКАМИ --------------------
    total_time = 0
    for run in range(RUNS):
        data = [random.randint(0, 1000000) for _ in range(n)]
        start = time.perf_counter()

        length = len(data)
        for i in range(1, length):
            key = data[i]
            j = i - 1
            while j >= 0 and key < data[j]:
                data[j + 1] = data[j]
                j -= 1
            data[j + 1] = key

        end = time.perf_counter()
        total_time += (end - start)
    avg_time = total_time / RUNS
    print(f"Вставками:  {avg_time:.6f} сек (среднее за {RUNS} запусков)")

    # ------------------- ВСТРОЕННАЯ СОРТИРОВКА --------------------
    # Встроенная быстрая — можно тестировать все размеры
    total_time = 0
    for run in range(RUNS):
```

```
            data = [random.randint(0, 1000000) for _ in range(n)]
            start = time.perf_counter()
            data.sort()
            end = time.perf_counter()
            total_time += (end - start)
        avg_time = total_time / RUNS
        print(f"Встроенная: {avg_time:.6f} сек (среднее за {RUNS} запусков)")

print("\n" + "=" * 90)
print("ЗАВЕРШЕНО. Скопируйте результаты в Excel для построения графиков.")
print("=" * 90)
```

Инструкция для студентов

Шаг 1. Запуск

1. Создайте файл `sorting_lab.py`, вставьте код.
2. Запустите: `python sorting_lab.py`
3. Дождитесь результатов или прервите долгие тесты через `Ctrl+C`.

Шаг 2. Копирование в Excel

1. Выделите и скопируйте вывод программы.
2. В Excel: Правая кнопка → **Специальная вставка** → **Текст Unicode**.
3. При необходимости: Данные → Текст по столбцам → разделитель `:` или пробел.

Шаг 3. Подготовка таблицы

Приведите данные к виду:

Elements	Bubble (avg)	Selection (avg)	Insertion (avg)	Built-in (avg)
100	0.001234	0.000876	0.000654	0.000012
1000	0.123456	0.087654	0.065432	0.000089
10000	12.345678	8.765432	6.543210	0.001234
100000	120	88	66	0.015678
1000000	1300	880	660	0.198765

> Заполняйте только полученные значения. Пропуски отмечайте `N/A`.

Шаг 4. График в Excel

1. Выделите таблицу → Вставка → **Точечная с прямыми отрезками**.
2. Подпишите оси: **X** — "Количество элементов", **Y** — "Среднее время (сек)".
3. Добавьте легенду и заголовок.
4. Для встроенной сортировки при необходимости используйте **вторичную ось Y**.

Вопросы для отчёта

1. **Заполните таблицу** усреднёнными результатами.
2. **Рассчитайте рост времени**:
   - Во сколько раз выросло среднее время пузырьковой сортировки при переходе 100 → 1000? А 1000 → 10000?
   - Сравните с теорией: при росте n в 10 раз, $O(n^2)$ должен расти в ~100 раз.
3. **Сравнение алгоритмов**:
   - Какой из трёх «ручных» алгоритмов оказался самым быстрым?
   - Почему усреднение по 10 запускам даёт более надёжный результат?
4. **Встроенная сортировка**:
   - Во сколько раз она быстрее пузырьковой на 10 000 элементах?
   - Почему её кривая на графике растёт медленнее?
5. **Анализ графика**:
   - Какую форму имеют кривые квадратичных алгоритмов?
   - Почему для больших N не удалось получить данные для пузырька?