Алгоритм - Сортировка Шелла / Shell sort (функция qsort)

Выполнил Холев А. Ю 22ПИ2

1. Постановка задачи

Цель работы - реализовать и исследовать эффективность двух алгоритмов сортировки и интеграция в EcoLab1:

1. Shell sort (сортировка Шелла) - алгоритм сортировки с постепенным уменьшением интервала сравнения элементов
2. qsort - стандартная библиотечная функция быстрой сортировки из stdlib.h

Сортировки исследовались на массивах различных типов данных: int, float, double, string. Размеры массивов: 80 000, 90 000, 100 000, 200 000, 300 000, 400 000 элементов.

Задачи работы:

- Реализовать компонент сортировки, поддерживающий разные типы данных
- Замерить время выполнения алгоритмов
- Сравнить производительность Shell sort и qsort
- Проанализировать результаты и сделать выводы
- Интегрировать алгоритм в EcoLab1

2. Реализуемые алгоритмы

2.1. Shell sort

- Методика: последовательное уменьшение интервала сравнения элементов (gap), пока gap ≠ 1
- Сложность:
  - Средняя: $O(n \log^2 n)$
  - Худшая: $O(n^2)$
- Память: O(1)

2.2. qsort

- Стандартная библиотечная функция быстрой сортировки
- Использует рекурсивное деление массива на части относительно pivot
- Оптимизирована для числовых и строковых массивов
- Сложность:

    - Средняя: O(n log n)
    - Худшая: O(n²) (редко)
- Память: O(log n)

3. Shell sort

Shell sort - это улучшенный вариант сортировки вставками.

- В обычной сортировке вставками каждый элемент сравнивается с предыдущими и вставляется на правильное место.
- В Shell sort сравнения сначала происходят на больших интервалах, называемых **gap** (шаг интервала).
- С каждым проходом gap уменьшается, пока не становится равным 1, после чего выполняется обычная сортировка вставками.

Преимущество: меньшее количество перемещений элементов по сравнению с обычной сортировкой вставками, особенно на больших массивах.

Особенности алгоритма

1. Эффективен для небольших и средних массивов
2. Не требует дополнительной памяти
3. Для больших массивов уступает qsort из-за почти квадратичной временной сложности
4. Можно оптимизировать, используя более эффективные последовательности gap (Хиббард, Седжвик)

Метод **CEcoLab1_ShellSort** реализует универсальный алгоритм сортировки Shell для массивов любого типа:

```
static int16_t ECOCALLMETHOD CEcoLab1_ShellSort(
    IEcoLab1Ptr_t me,
    void *arrPrt,
    size_t arrSize,
    size_t elemSize,
    int (*compare)(const void *, const void *)
) {
    CEcoLab1* pCMe = (CEcoLab1*)me;
    char* tmp = (char*)pCMe->m_pIMem->pVTbl->Alloc(pCMe->m_pIMem, elemSize);
    char* arr = (char *)arrPrt;

    for (ssize_t gap = arrSize / 2; gap > 0; gap /= 2) {
```

```
        for (ssize_t i = gap; i < arrSize; i++) {
            memcpy(tmp, arr + i * elemSize, elemSize);
            ssize_t j = i;
            while (j >= gap && compare(arr + (j - gap) * elemSize, tmp) > 0) {
                memcpy(arr + j * elemSize, arr + (j - gap) * elemSize, elemSize);
                j -= gap;
            }
            memcpy(arr + j * elemSize, tmp, elemSize);
        }
    }
    pCMe->m_pIMem->pVTbl->Free(pCMe->m_pIMem, tmp);
    return ERR_ECO_SUCCESES;
}
```

Пояснение:

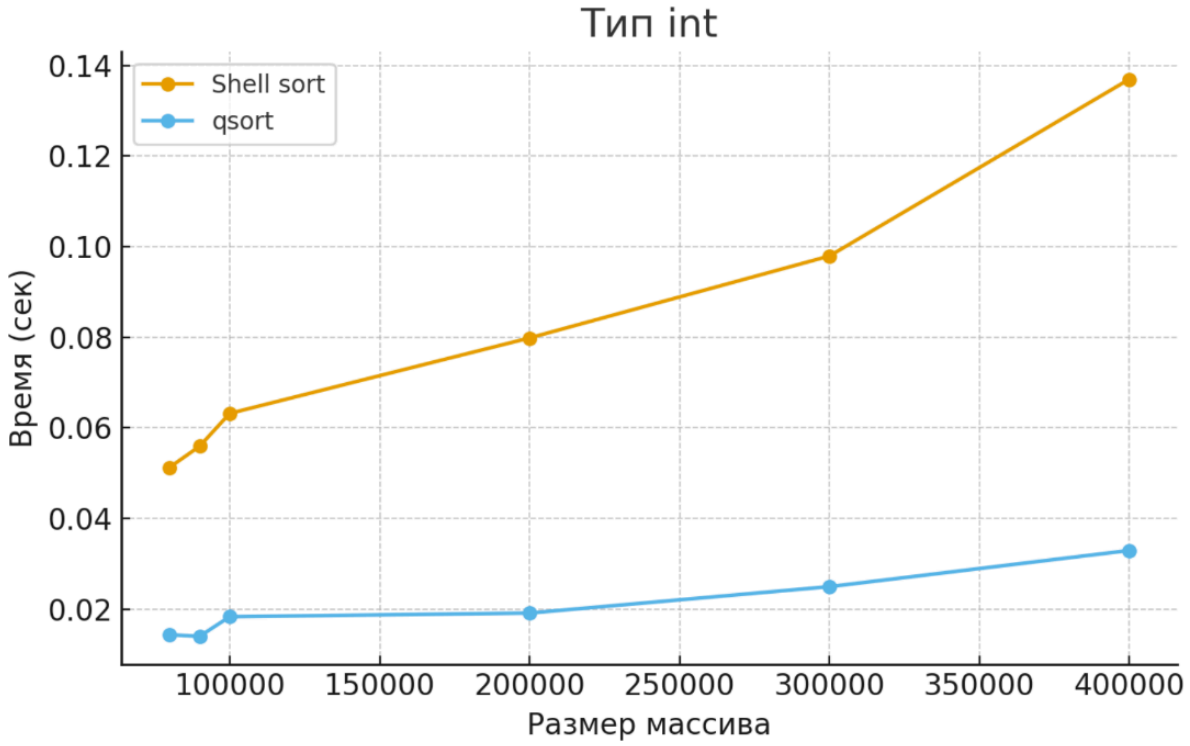
- arrPrt – указатель на массив для сортировки
- arrSize – количество элементов
- elemSize – размер одного элемента
- compare – функция сравнения двух элементов
- Выделяется временный буфер tmp для хранения копируемого элемента
- Используется стандартная последовательность разрывов gap (N/2, N/4, ..., 1) для Shell Sort
- Копирование элементов осуществляется через memcpy, что позволяет работать с массивами любого типа

4. Результаты работы (время выполнения)

4.1. Тип int

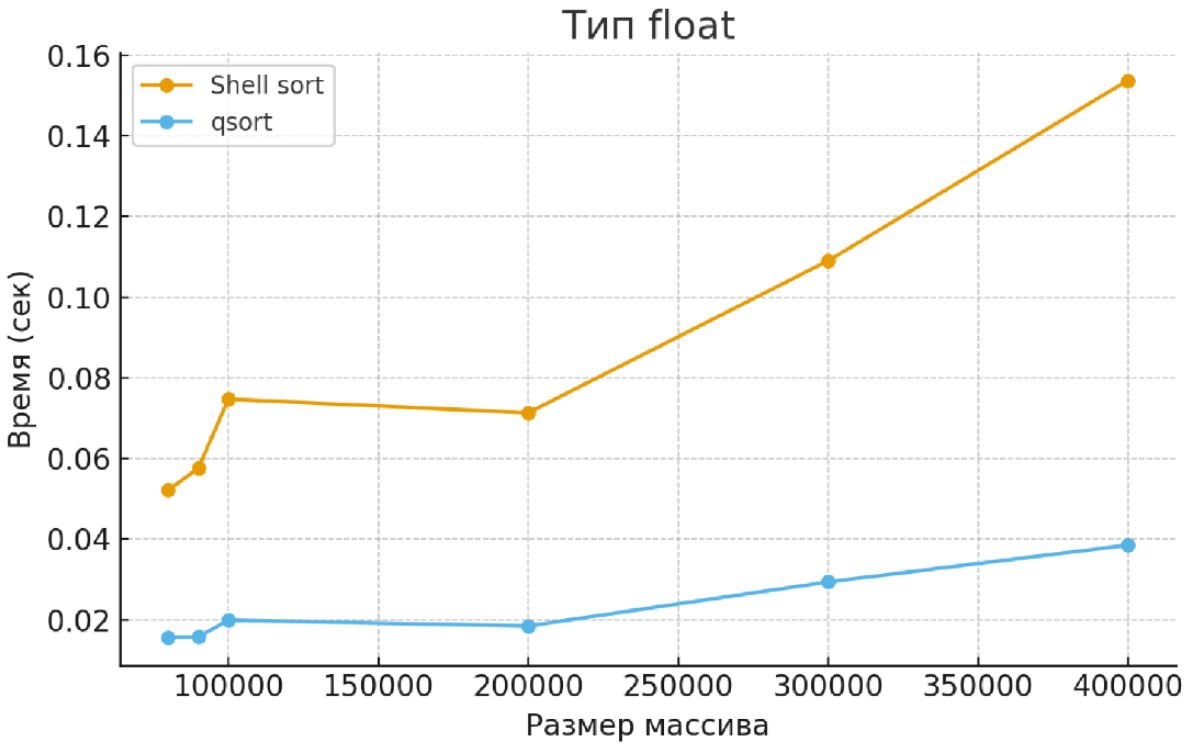
Размер массива	Shell sort (c)	qsort (c)
80 000	0.051284	0.014319
90 000	0.055943	0.013994
100 000	0.063172	0.018311
200 000	0.079831	0.019120
300 000	0.097913	0.024944

400 000	0.136932	0.032932



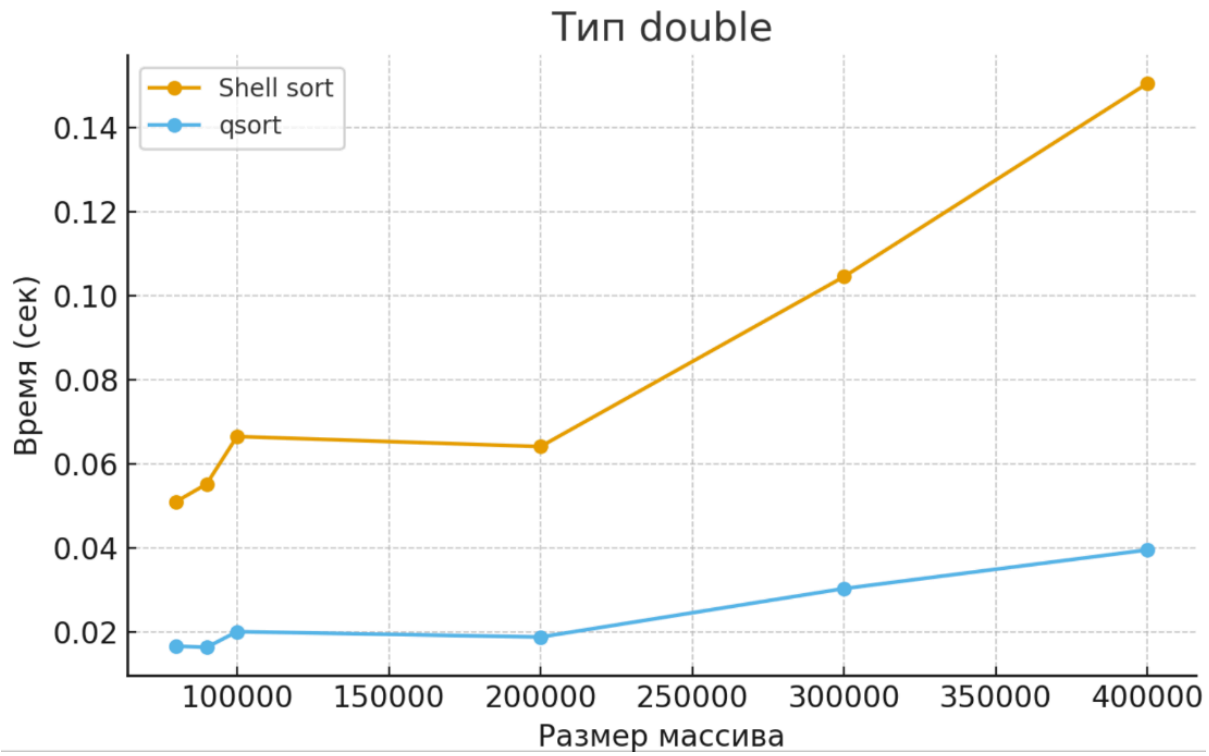
4.2. Тип float

Размер массива	Shell sort (c)	qsort (c)
80 000	0.052178	0.015655
90 000	0.057564	0.015757
100 000	0.074702	0.019906
200 000	0.071310	0.018484
300 000	0.108984	0.029396
400 000	0.153731	0.038500



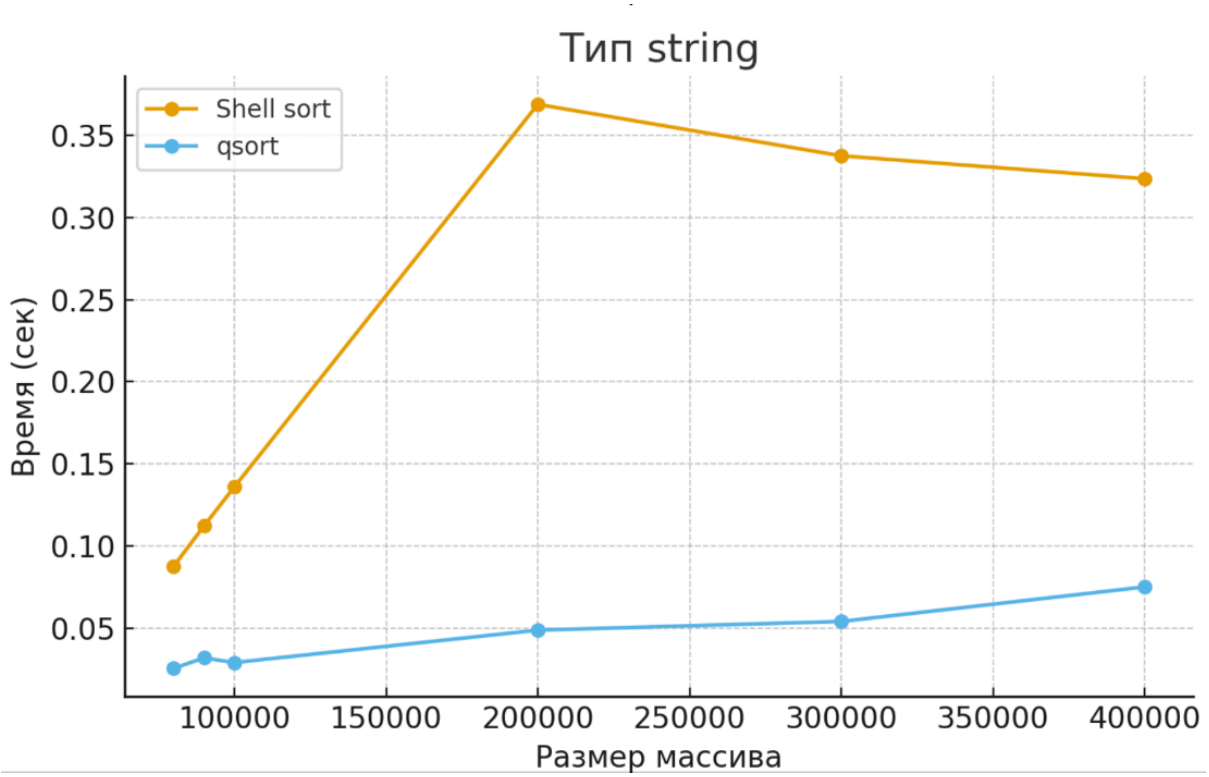
4.3. Тип double

Размер массива	Shell sort (c)	qsort (c)
80 000	0.051070	0.016633
90 000	0.055242	0.016345
100 000	0.066548	0.020089
200 000	0.064148	0.018806
300 000	0.104596	0.030341
400 000	0.150625	0.039544



4.4. Тип string

Размер массива	Shell sort (с)	qsort (с)
80 000	0.087576	0.025522
90 000	0.112179	0.031885
100 000	0.135910	0.029064
200 000	0.368807	0.048784
300 000	0.337512	0.054090
400 000	0.323564	0.075081



5. Сравнительный анализ

1. Производительность:
   - qsort быстрее Shell sort для всех типов данных и всех размеров массивов
   - Разница особенно заметна на больших массивах и при сортировке строк (string)
2. Зависимость от типа данных:
   - Для чисел (int, float, double) Shell sort медленнее примерно в 3-4 раза
   - Для строк Shell sort проигрывает в 5-7 раз
3. Рост времени с увеличением массива:
   - Shell sort демонстрирует почти квадратичный рост времени
   - qsort растёт почти линейно с логарифмом размера массива

6. Выводы

1. Рекомендуемый алгоритм: qsort для больших массивов и любых типов данных.
2. Использование Shell sort:
   - Подходит для обучения или небольших массивов.
   - Обеспечивает стабильную работу, но медленнее библиотечных решений.

3. Особенности:
    - Shell sort плохо масштабируется на массивы с большим количеством элементов, особенно со строками.
    - qsort использует оптимизации, которые делают его эффективным для любых массивов.

7. Методика измерения времени

- Время измерялось с помощью функции clock() из библиотеки time.h.

Формула:

```
double time_spent = (double)(end - start) / CLOCKS_PER_SEC;
```