Gnome sort

1. Общая постановка задачи (41)

Необходимо разработать компонент, реализующий универсальный алгоритм сортировки Gnome Sort, который работает с массивами произвольных типов данных. Алгоритм должен быть интегрирован в структуру IEcoLab1 и поддерживать стандартную сигнатуру функции сортировки.

Компонент должен корректно работать с различными типами данных (int, float, double, char, long double и др.) и обеспечивать безопасное использование памяти через системный аллокатор.

2. Реализуемый алгоритм сортировки

Алгоритм Gnome Sort — это простой сортировочный метод, концептуально близкий к Insertion Sort, но использующий другой принцип перемещения элементов. Он последовательно проходит массив и переставляет соседние элементы, если они нарушают порядок. В случае обмена алгоритм возвращается на один шаг назад, пока порядок не будет восстановлен.

Основная идея:

- Если текущий элемент больше или равен предыдущему — перейти к следующему элементу;
- Если меньше — поменять местами и шагнуть назад;
- Если достигнут первый элемент — перейти вперёд.

Псевдокод:

```
index = 0

while index < nmemb:

  if index == 0:

    index += 1

  else:

    if arr[index] >= arr[index - 1]:

      index += 1

    else:

      arr[index], arr[index - 1] = arr[index - 1], arr[index]

      index -= 1
```

Особенности реализации:

- Универсальность обеспечена использованием указателей void* и функции сравнения compar;
- Обмен элементов реализован через вспомогательную функцию eco_memcpy, обеспечивающую побайтовое копирование без зависимости от типа данных;
- Временный буфер temp выделяется динамически через системный аллокатор pIMem->pVTbl->Alloc;
- При завершении работы буфер освобождается через pIMem->pVTbl->Free.

3. Асимптотический анализ

Характеристика	Gnome Sort	qsort (QuickSort, типичная реализация)
Временная сложность (средняя)	$O(N^2)$	O(N log N)
Временная сложность (лучшая)	O(N)	O(N log N)
Временная сложность (худшая)	$O(N^2)$	$O(N^2)$
Пространственная сложность	O(1)	O(log N) (рекурсивный стек)

Алгоритм Gnome Sort имеет квадратичную временную сложность и неэффективен на больших массивах, но прост в реализации и не требует дополнительной памяти, кроме временного буфера для обмена элементов.

4. Реализация и интерфейс

Реализация выполнена в файле CEcoLab1.c, где в таблицу виртуальных методов IEcoLab1VTbl добавлен новый метод GnomeSort.

Компонент интегрирован в систему Eco через функции createCEcoLab1 и deleteCEcoLab1.

Реализованы стандартные методы COM-подобной структуры (QueryInterface, AddRef, Release) и тестовый модуль EcoMain, в котором проводится комплексное тестирование и сравнение с qsort.

```
typedef struct IEcoLab1VTbl {
   /* IEcoUnknown */

   int16_t (ECOCALLMETHOD *QueryInterface)(/* in */ IEcoLab1Ptr_t me, /* in */ const
UGUID* riid, /* out */ voidptr_t* ppv);
   uint32_t (ECOCALLMETHOD *AddRef)(/* in */ IEcoLab1Ptr_t me);
   uint32_t (ECOCALLMETHOD *Release)(/* in */ IEcoLab1Ptr_t me);



   /* IEcoLab1 */
   nt16_t (ECOCALLMETHOD *GnomeSort)(/* in */ IEcoLab1Ptr_t me, /* in */ void* base,
                              /* in */ size_t nmemb, /* in */ size_t size,
                              /* in */ int (*compar)(const void*, const void*));
} IEcoLab1VTbl, *IEcoLab1VTblPtr;

interface IEcoLab1 {
   struct IEcoLab1VTbl *pVTbl;
} IEcoLab1;
```

5. Тестирование

Тестирование проводилось на различных типах данных:

- int, float, double, long long, long double, char
- Сценарии тестов:
    1. Неотсортированный массив int
    2. Уже отсортированный массив
    3. Массив в обратном порядке
    4. Массив с повторяющимися элементами
    5. Один элемент
    6. Пустой массив
    7. Массив char
    8. Массив double
    9. Массив float
    10. Массив long
    11. Массив long long
    12. Массив long double

Каждый тест проверял корректность результата с помощью функции check_is_sorted, выводящей PASS при успешной сортировке и FAIL при ошибке.

Результат тестов:

Все тесты успешно прошли (PASS), что подтверждает корректность работы алгоритма с разными типами данных и граничными случаями.

```
=== Testing GnomeSort ===
Test 1: Normal int sort       PASS
Test 2: Already sorted array  PASS
Test 3: Reverse order         PASS
Test 4: Doubled element       PASS
Test 5: One element           PASS
Test 6: Zero element          PASS
Test 7: Char elements         PASS
Test 8: Double elements       PASS
Test 9: Float elements        PASS
Test 10: Long elements        PASS
Test 11: Long long elements   PASS
Test 12: Long double elements PASS
 === INTEGER ===
---Size 10---
GnomeSort time: 0.004 ms
Qsort     time: 0.001 ms
---Size 30---
GnomeSort time: 0.005 ms
Qsort     time: 0.003 ms
---Size 50---
GnomeSort time: 0.018 ms
Qsort     time: 0.003 ms
---Size 70---
GnomeSort time: 0.031 ms
Qsort     time: 0.004 ms
---Size 90---
GnomeSort time: 0.052 ms
Qsort     time: 0.005 ms
---Size 100---
GnomeSort time: 0.059 ms
Qsort     time: 0.007 ms
```

6. Сравнение производительности (Gnome Sort vs qsort)

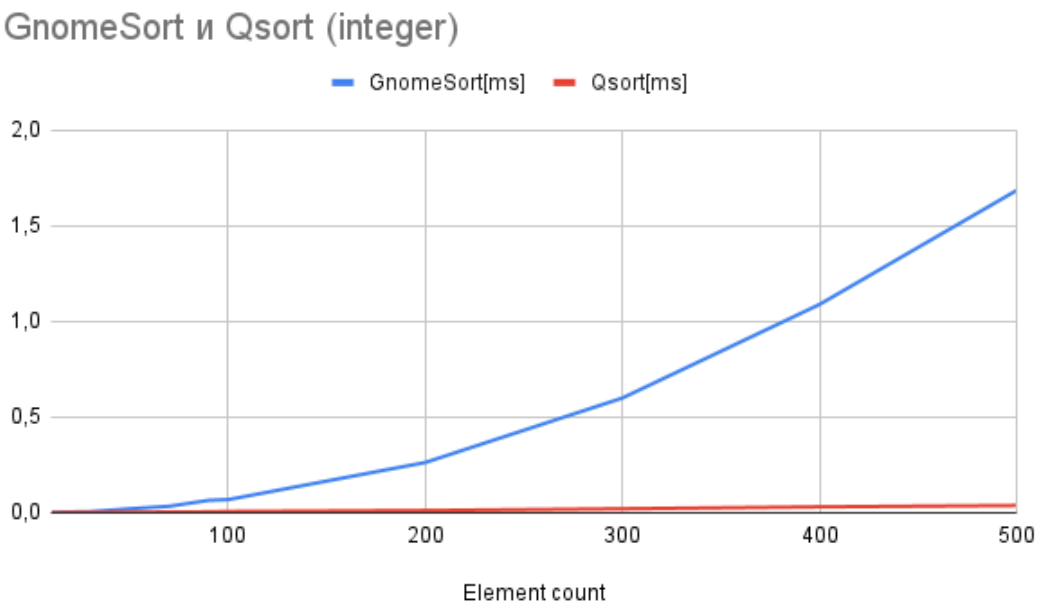
Были проведены сравнительные замеры времени выполнения GnomeSort и стандартной qsort для различных типов данных и размеров массива (от 10 до 20 000 элементов). Для объективности измерений массивы заполнялись случайными значениями с помощью функции rand().

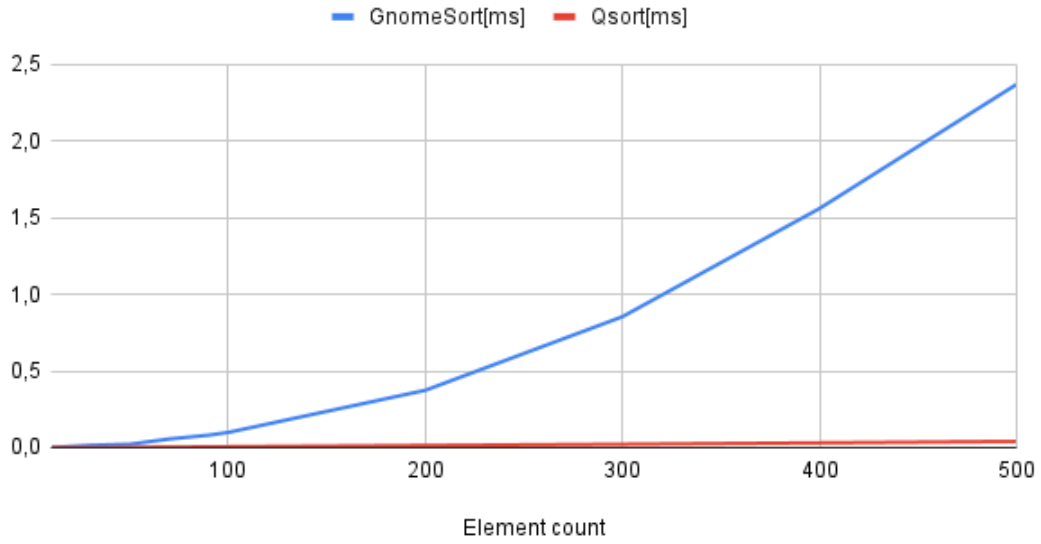
Время измерялось с помощью функции clock() из библиотеки <time.h>.

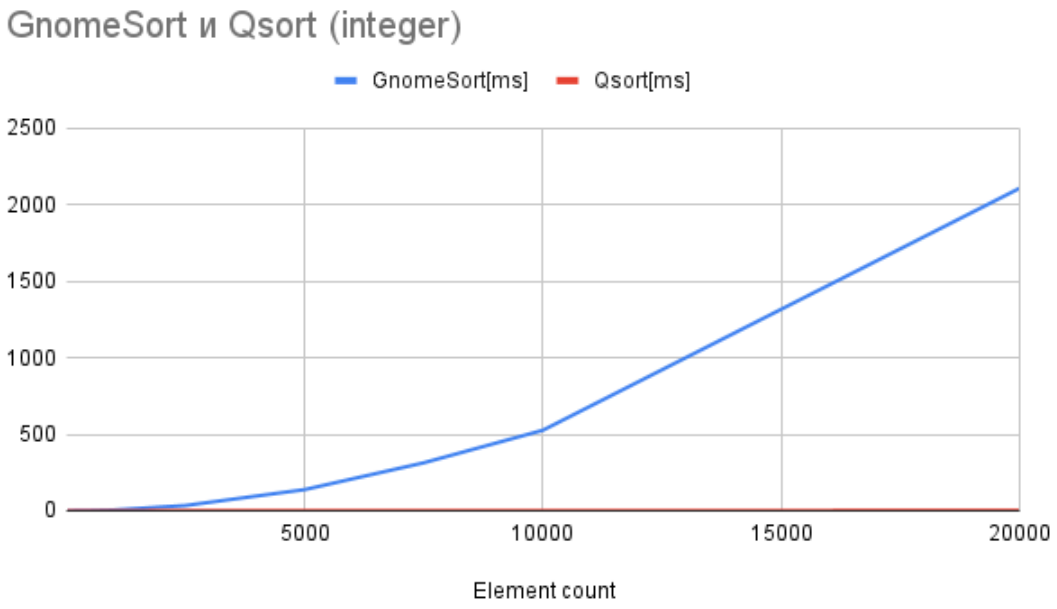
Ниже приведены результаты одного прогона для разных размеров массивов:

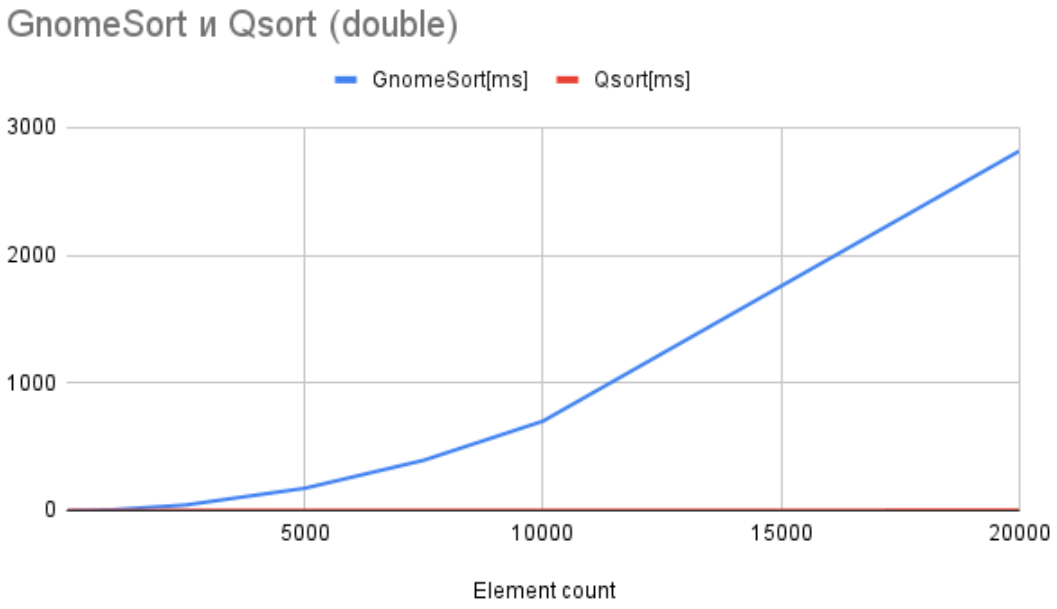
Integer		
Size	**GnomeSort[ms]**	**Qsort[ms]**
10	0,003	0,000
30	0,005	0,001
50	0,015	0,002
70	0,029	0,004
90	0,080	0,005
100	0,059	0,005
200	0,225	0,014
300	0,480	0,019
400	0,905	0,027
500	1,365	0,035
1000	5,489	0,080
2500	34,166	0,220
5000	138,155	0,474
7500	313,356	0,757
10000	526,130	0,965
20000	2107,293	2,043

Double		
Size	**GnomeSort[ms]**	**Qsort[ms]**
10	0,002	0,001
30	0,007	0,002
50	0,020	0,003
70	0,038	0,006
90	0,062	0,006
100	0,071	0,006
200	0,308	0,014
300	0,666	0,021
400	1,165	0,030
500	1,678	0,039
1000	6,749	0,081
2500	43,451	0,230
5000	174,340	0,506
7500	393,850	0,771
10000	700,051	1,062
20000	2816,105	2,291


GnomeSort и Qsort (integer)
GnomeSort[ms]
Qsort[ms]
2,0
1,5
1,0
0,5
0,0
100
200
300
400
500
Element count


GnomeSort и Qsort (double)
GnomeSort[ms]
Qsort[ms]
2,5
2,0
1,5
1,0
0,5
0,0
100
200
300
400
500
Element count





Результаты наблюдений:

- На малых массивах (до ~100 элементов) GnomeSort показывает схожее время с qsort;
- С ростом размера массива время выполнения GnomeSort резко увеличивается из-за квадратичной сложности;
- На массивах свыше 1000 элементов qsort работает значительно быстрее;
- Время сортировки GnomeSort заметно зависит от типа данных — для double и long double оно выше из-за увеличенных операций копирования.

7. Заключение

В рамках работы был реализован универсальный компонент CEcoLab1, содержащий метод GnomeSort, полностью интегрированный в инфраструктуру EcoSystem.

Алгоритм корректно сортирует массивы любых типов данных, успешно прошёл все тесты.

Сравнение с qsort показало, что GnomeSort проигрывает по скорости на больших массивах из-за квадратичной временной сложности, но является хорошим примером простого алгоритма, который показывает принципы обменной сортировки без рекурсии.