УДК *004.421.2*

И.В. Овсянников, М.Ю. Смоленцев

Иркутский государственный университет путей сообщений, г. Иркутск, Российская Федерация

ПРИМЕНЕНИЕ АЛГОРИТМОВ СОРТИРОВКИ В СИСТЕМАХ АВТОМАТИКИ И ТЕЛЕМЕХАНИКИ НА ЯЗЫКЕ АССЕМБЛЕРА MASM

Аннотация. *В данной статье рассматривается разработка и оптимизация алгоритмов сортировки, реализованных на мощном и низкоуровневом языке ассемблера — MASM (Macro Assembler), с акцентом на их применение в системах автоматики и телемеханики. Рассмотрены основные принципы написания алгоритмов сортировки на языке ассемблера, а также предоставлены примеры кода для каждого алгоритма.*

Статья анализирует преимущества использования языка ассемблера для оптимизации производительности алгоритмов сортировки в контексте автоматики, телемеханики и связи, такие как возможность контролировать непосредственно аппаратные ресурсы и использовать специфические инструкции процессора. Авторы приводят примеры кода, демонстрирующие реализацию и оптимизацию каждого из представленных алгоритмов, а также представляют результаты сравнительного анализа производительности. Для наглядности представлена разработанная авторами программа для визуализации процесса сортировки.

Результаты исследования могут быть использованы для оптимизации существующих алгоритмов и разработки новых методов обработки данных на языке ассемблера в области автоматики, телемеханики и связи. Статья полезна как для специалистов в области программирования на ассемблере, так и для тех, кто хочет понять принципы работы и оптимизации алгоритмов сортировки на низком уровне в рамках указанных областей.

Ключевые слова: *ассемблер, MASM, алгоритмы сортировки, оптимизация алгоритмов, визуализация сортировок, низкоуровневое программирование, программная реализация*.

I.V. Ovsyannikov, M.Y. Smolentsev

Irkutsk State Transport University, Irkutsk, the Russian Federation

APPLICATION OF SORTING ALGORITHMS IN AUTOMATION AND TELEMECHANICS SYSTEMS IN THE ASSEMBLY LANGUAGE MASM

Abstract. *This paper discusses the development and optimization of sorting algorithms implemented in the powerful and low-level assembly language, MASM (Macro Assembler), with a focus on their application in automation and telemechanic systems. The main principles of writing sorting algorithms in Assembler are discussed, and code examples for each algorithm are given.*

The paper analyzes advantages of using Assembler language for performance optimization of sorting algorithms in the context of automatics, telemechanic and communications, such as the ability to control hardware resources directly and to use specific processor instructions. The authors provide code examples to demonstrate the implementation and optimization of each of the presented algorithms and also present the results of comparative performance analysis. A program developed by the authors to visualize the sorting process is presented for your reference.

The results can be used for optimization of existing algorithms and development of new methods of data processing in Assembler language in the field of automatics, telemechanic and communication. The paper is useful both for experts in assembly language programming and for those who want to understand the principles of low-level sorting algorithms and optimization within the above areas.

Keywords: *assembler, MASM, sorting algorithms, algorithms optimization, sorting visualization, low-level programming, program realization*.

Введение

В современном мире информационных технологий процесс обработки и сортировки больших массивов данных становится всё более важным. В связи с этим, оптимизация алгоритмов сортировки является одним из ключевых направлений исследований в области компьютерных наук [2], [3]. Настоящая статья посвящена разработке и улучшению

алгоритмов сортировки, реализованных на низкоуровневом языке программирования – ассемблере MASM (Macro Assembler).

Язык ассемблера MASM позволяет работать непосредственно с аппаратными ресурсами компьютера [1], что предоставляет возможность внедрять различные стратегии оптимизации, направленные на ускорение обработки данных и улучшение производительности. В рамках данной работы проведен анализ разнообразных методов сортировки, таких как пузырьковая, быстрая, вставками и слиянием, а также рассмотрены способы их оптимизации с использованием особенностей языка ассемблера MASM.

Начнем с изучения теоретических основ каждого алгоритма сортировки, а затем перейдем к их практической реализации с использованием инструкций и команд ассемблера MASM. В процессе реализации описаны различные техники оптимизации, включая эффективное использование регистров и кэш-памяти, а также уменьшение числа выполняемых операций.

После представления примеров кода и детального описания каждого алгоритма сортировки, сравнивается их производительность с помощью различных тестовых наборов данных, чтобы определить наиболее эффективные подходы к оптимизации. Сделаем выводы о возможностях улучшения производительности алгоритмов сортировки и предложим дальнейшие направления исследований в данной области. Кроме того, авторы статьи разработали специальную программу для визуализации процесса сортировки, что позволяет наглядно продемонстрировать эффективность и особенности работы каждого рассматриваемого алгоритма сортировки. Данное приложение представляет собой инструмент, который можно использовать как для образовательных целей, так и для анализа производительности различных алгоритмов сортировки в реальных условиях.

Понятие сортировки

С алгоритмической точки зрения, сортировка представляет собой процесс упорядочивания элементов некоторой последовательности (обычно числовой или текстовой) в соответствии с заданным критерием [2]. Алгоритмы сортировки являются ключевыми компонентами многих компьютерных приложений, обрабатывающих данные, и их эффективность напрямую влияет на производительность таких систем. В компьютерных науках существует множество различных алгоритмов сортировки, каждый из которых имеет свои преимущества и недостатки, в зависимости от типа и объема данных, а также от конкретных требований к производительности.

С математической точки зрения, сортировка может быть определена как перестановка элементов множества или последовательности таким образом, чтобы они следовали в определенном порядке (возрастанию или убыванию). Допустим, есть последовательность чисел $A = \{a_1, a_2, \dots, a_n\}$. Сортировка этой последовательности может быть описана с помощью перестановки $P = \{p_1, p_2, \dots, p_n\}$, где каждый элемент p_i является индексом элемента a_i в отсортированной последовательности $B = \{b_1, b_2, \dots, b_n\}$.

Свойства сортировок

Разнообразие алгоритмов сортировки делает их идеальным объектом для изучения и сравнительного анализа. Основные свойства, которые могут быть использованы для описания и сравнения алгоритмов сортировки:

1. Временная сложность

Временная сложность — это оценка количества времени, необходимого для выполнения алгоритма в зависимости от размера входных данных. Она обычно выражается в виде асимптотической нотации, такой как $O(n)$ – линейная сложность, $O(n^2)$ — квадратическая, $O(\log_2 n)$ — логарифмическая, или $O(n \times \log_2 n)$ – логарифмически-линейная, где n —количество элементов в массиве. Временная сложность является ключевым показателем эффективности алгоритма. [2], [3], [4]

2. Пространственная сложность

Пространственная сложность — это оценка количества памяти, требуемой для выполнения алгоритма. Алгоритмы могут быть "на месте" (in-place), если они используют

ограниченное количество дополнительной памяти, или "внешние", если они требуют некоторого количества дополнительной памяти для работы.

3. Устойчивость

Устойчивость – это свойство алгоритма сортировки, при котором одинаковые элементы сохраняют свой относительный порядок после сортировки. Устойчивые алгоритмы сортировки предпочтительны в случаях, когда важно сохранить первоначальный порядок элементов.

Изучение основных свойств алгоритмов сортировки позволяет лучше понять их преимущества и недостатки, что облегчает выбор наиболее подходящего метода для конкретного приложения или задачи. Важно заметить, что не существует "идеального" алгоритма сортировки, подходящего для всех сценариев. Выбор наиболее эффективного алгоритма зависит от специфических требований задачи, таких как размер входных данных, их распределение, доступная память и другие факторы.

Задачи, требующие сортировки данных

Сортировка данных играет важную роль в обработке информации в областях автоматики, телемеханики и связи, так как она способствует оптимизации процессов и улучшению производительности систем. В данном пункте мы рассмотрим основные задачи, возникающие в этих областях, которые приводят к необходимости сортировки данных.

1. Оптимизация маршрутов и координация движения:

Сортировка данных о местоположении объектов и их приоритетности позволяет определить оптимальный порядок посещения, сокращая время и затраты на перемещение. Это актуально для роботов, беспилотных транспортных средств и дистанционно управляемых аппаратов.

2. Управление очередностью обработки задач и запросов:

В автоматических системах управления и связи, сортировка данных обеспечивает оптимальное распределение ресурсов, снижает задержки и увеличивает производительность системы путем определения приоритетов выполнения задач и обработки запросов.

3. Распределение каналов связи и управление трафиком:

Сортировка данных о передаваемой информации позволяет оптимизировать процесс передачи данных, управлять трафиком и предотвратить перегрузку сети. Упорядочивание пакетов данных по приоритетам или времени отправки обеспечивает эффективное использование каналов связи.

4. Анализ и обработка сигналов от датчиков и актуаторов:

Сортировка данных о температуре, давлении, влажности и других параметрах позволяет выявить аномалии, определить тренды и обеспечить эффективное управление и регулирование параметров среды в автоматических системах управления и телемеханике.

5. Классификация и категоризация данных:

Сортировка данных упрощает анализ и ускоряет обработку информации в автоматике, телемеханике и связи. Это позволяет системам группировать и категоризировать данные по определенным критериям, облегчая доступ к необходимой информации и улучшая общую эффективность системы.

Виды алгоритмов сортировки

В данной статье приведено несколько популярных алгоритмов сортировки, которые разбиты на 2 класса: простые и сложные. К простым сортировкам относятся: сортировка пузырьком (Bubble Sort), сортировка вставками (Insertion Sort), сортировка выбором (Selection Sort), сортировка перемешиванием (Shaker Sort). К сложным сортировкам относятся: быстрая сортировка (Quick Sort), сортировка кучей (Heap Sort), сортировка слиянием (Merge Sort), сортировка Шелла (Shell Sort) [3], [4], [5].

Простые сортировки, как правило, имеют квадратичную временную сложность $O(n^2)$, но просты в реализации. Хорошо подходят для небольших или частично отсортированных массивов, становятся неэффективными для больших наборов данных.

Сложные сортировки имеют логарифмически-линейную временную сложность $O(n \times \log_2 n)$ или линейную сложность $O(n)$, требуют более продуманных алгоритмов. Эффективны для больших наборов данных, обеспечивают более быструю сортировку по сравнению с простыми сортировками.

Ниже представлена сравнительная таблица временной сложности представленных алгоритмов.

Таблица 1

Сравнительная таблица временной сложности представленных алгоритмов

Алгоритм сортировки	**Лучший случай**	**Средний случай**	**Худший случай**
Сортировка пузырьком	$O(n)$	$O(n^2)$	$O(n^2)$
Сортировка вставками	$O(n)$	$O(n^2)$	$O(n^2)$
Сортировка выбором	$O(n^2)$	$O(n^2)$	$O(n^2)$
Сортировка перемешиванием	$O(n)$	$O(n^2)$	$O(n^2)$
Быстрая сортировка	$O(n \times \log_2 n)$	$O(n \times \log_2 n)$	$O(n^2)$
Сортировка кучей	$O(n \times \log_2 n)$	$O(n \times \log_2 n)$	$O(n \times \log_2 n)$
Сортировка слиянием	$O(n \times \log_2 n)$	$O(n \times \log_2 n)$	$O(n \times \log_2 n)$
Сортировка Шелла	$O(n)$	$O(n \times (\log_2 n)^2)$	$O(n \times (\log_2 n)^2)$

Краткое описание работы каждого из представленных алгоритмов:

1. Сортировка пузырьком (Bubble Sort): Простой алгоритм, сравнивает соседние элементы массива и меняет их местами, если они расположены в неправильном порядке. Процесс повторяется до тех пор, пока массив не будет полностью отсортирован.

2. Сортировка вставками (Insertion Sort): Алгоритм проходит по массиву и вставляет каждый элемент на соответствующее место в отсортированной части массива. Особенно эффективен для частично отсортированных массивов.

3. Сортировка выбором (Selection Sort): Алгоритм находит наименьший элемент в массиве, перемещает его на первую позицию, затем находит следующий наименьший элемент и перемещает его на вторую позицию и так далее, пока массив не будет полностью отсортирован.

4. Сортировка перемешиванием (Shaker Sort): также известна как коктейльная сортировка, является вариацией сортировки пузырьком. Алгоритм проходит по массиву в обоих направлениях, сравнивая соседние элементы и обменивая их местами, если они находятся в неправильном порядке. Это улучшает эффективность сортировки пузырьком.

5. Быстрая сортировка (Quick Sort): Рекурсивный алгоритм сортировки "разделяй и властвуй", который выбирает опорный элемент и разделяет массив на две части: элементы, меньшие или равные опорному, и элементы, большие опорного. Затем процесс повторяется для каждой из частей.

6. Сортировка кучей (Heap Sort): Алгоритм использует двоичную кучу для сортировки. Сначала алгоритм строит максимальную кучу из массива, затем последовательно извлекает максимальный элемент из кучи и вставляет его в конец массива, пока куча не опустеет.

7. Сортировка слиянием (Merge Sort): Рекурсивный алгоритм "разделяй и властвуй", разбивает массив на подмассивы до тех пор, пока в каждом подмассиве не останется по одному элементу. Затем подмассивы сливаются обратно в правильном порядке.

8. Сортировка Шелла (Shell Sort): Улучшенный вариант сортировки вставками, сначала сравнивает и сортирует элементы на определенном расстоянии друг от друга (шаг сортировки). Шаг сортировки последовательно уменьшается до 1, после чего алгоритм заканчивает сортировку. Эффективность сортировки Шелла зависит от выбранной последовательности шагов.

Реализация алгоритмов сортировки на языке ассемблера

Язык ассемблера представляет собой набор мнемонических кодов, которые соответствуют машинным инструкциям процессора. Реализация алгоритмов сортировки на языке ассемблера позволяет достичь оптимальной производительности за счет

непосредственного управления аппаратными ресурсами и использования специфических инструкций процессора. В данной статье для реализации алгоритмов сортировки используется MASM (Macro Assembler) — мощный и низкоуровневый ассемблер, разработанный Microsoft.

Преимущества ассемблера заключаются в его способности использовать все возможности компьютера и постоянно развивающиеся технологии обработки данных, такие как MMX, SSE, AVX и др. Ассемблер позволяет легко работать с 16-, 32- и 64-разрядными регистрами, обеспечивая полное раскрытие архитектуры компьютера. Благодаря близости к машинному коду, ассемблер обеспечивает компактный исполнительный код и быстроту выполнения программы. Он также способствует параллельной обработке данных, используя различные размерности регистров микропроцессора, что дает превосходное быстродействие при обработке массивов данных. Ассемблер является ключевым инструментом для работы с регистрами и их отладки, что делает его неотъемлемой частью изучения и разработки компьютерной архитектуры.

При реализации алгоритмов сортировки на языке ассемблера следует учитывать различия между архитектурами процессоров и их поддерживаемыми инструкциями. Это требует адаптации кода для конкретной архитектуры или разработки универсальных решений, поддерживающих несколько архитектур.

Реализация алгоритмов сортировки на языке ассемблера сложная, но интересная и перспективная задача, позволяющая углубить понимание принципов работы и оптимизации алгоритмов на низком уровне. Результаты исследования можно использовать для оптимизации существующих алгоритмов и разработки новых методов обработки данных на языке ассемблера.

Программный код алгоритмов

Ниже представлен листинг кода для некоторых алгоритмов сортировки на языке ассемблера MASM (2 для простых сортировок и 2 для сложных). Программный код включает комментарии, поясняющие работу каждой инструкции и облегчающие понимание алгоритма. Примеры кода содержат различные оптимизации, специфичные для языка ассемблера, такие как использование специфических инструкций процессора или оптимизация управления потоком выполнения.

Сортировка вставками

```
insertionSort proc <5, 8, 4> arr:qword, len:dword
xor rdi, rdi  ; i
xor rsi, rsi  ; j
xor r9, r9    ; key
mov rbx, rcx
i_loop:
        inc edi
        cmp edi, len
        jnl @f
        mov eax, edi
        dec eax
        mov esi, eax
        mov r9d, dword ptr [rbx + rdi * 4]
        j_loop:
            push r9
            invoke Sleep, 1
            pop r9
            cmp esi, 0
            jl change
            cmp r9d, dword ptr [rbx + rsi * 4]
            jnl change
```

```
                mov eax, esi
                inc eax
                mov ecx, dword ptr [rbx + rsi * 4]
                mov dword ptr [rbx + rax * 4], ecx
                dec esi
                jmp j_loop
        change:
                mov eax, esi
                inc eax
                mov dword ptr [rbx + rax * 4], r9d
                jmp i_loop
@@:
ret
insertionSort endp
```

Сортировка перемешиванием

```
shakerSort proc <5, 8, 4> arr:qword, len:dword
local demmy:qword
xor rdi, rdi  ; start
xor rsi, rsi
mov eax, len
dec eax
mov esi, eax        ; end
mov rbx, arr
while_loop:
        cmp rdi, rsi
        jnl bye
        invoke Sleep, 50
        mov r8, rdi
i_loop:
        cmp r8, rsi
        je i_end
        mov ecx, dword ptr [rbx + r8 * 4] ; arr[i]
        mov rax, r8
        inc rax
        cmp ecx, dword ptr [rbx + rax * 4] ; arr[i] > arr[i+1] ?
        jng @f
                mov eax, dword ptr [rbx + rax * 4] ; arr[i+1]
                xchg ecx, eax
                mov dword ptr [rbx + r8 * 4], ecx ; arr[i]
                mov rdx, r8
                inc rdx
                mov dword ptr [rbx + rdx * 4], eax
        @@:
                ;push r8
                ;invoke Sleep, 1
                ;pop r8
        inc r8
        jmp i_loop
i_end:
        mov r9, rsi
j_loop:
```

```
        cmp r9, rdi
        je j_end
        mov ecx, dword ptr [rbx + r9 * 4] ; arr[j]
        mov rax, r9
        dec rax
        cmp ecx, dword ptr [rbx + rax * 4] ; arr[i] > arr[i+1] ?
        jnl @f
                mov eax, dword ptr [rbx + rax * 4] ; arr[i+1]
                xchg ecx, eax
                mov dword ptr [rbx + r9 * 4], ecx ; arr[i]
                mov rdx, r9
                dec rdx
                mov dword ptr [rbx + rdx * 4], eax
        @@:
                ;push r9
                ;invoke Sleep, 1
                ;pop r9
        dec r9
        jmp j_loop
j_end:
        dec esi
        inc rdi
        jmp while_loop
bye:
ret
shakerSort endp
```

Быстрая сортировка

```
quickSort proc <5, 8, 4> arr:qword, left:dword, right:dword
local pivot:dword
sub rsp, 28h
cmp rdx, r8
        jg bye
mov arr, rcx
mov left, edx   ; left
mov right, r8d     ; right
xor rax, rax
mov eax, left
mov eax, dword ptr [rcx + rax * 4]
mov pivot, eax
xor rdi, rdi
xor rsi, rsi
mov edi, left
mov esi, right
mov rbx, rcx
_while:
        cmp rdi, rsi
        jg __while
        _wright:
                mov eax, pivot
                cmp dword ptr [rbx + rdi * 4], eax
                jnl _lright
```

```
            inc rdi
            jmp _wright
        _lright:
            mov eax, pivot
            cmp dword ptr [rbx + rsi * 4], eax
            jng _change
            dec rsi
            jmp _lright
        _change:
            cmp rdi, rsi
            jg _while
            mov eax, dword ptr [rbx + rdi * 4]
            xchg eax, dword ptr [rbx + rsi * 4]
            mov dword ptr [rbx + rdi * 4], eax
            inc rdi
            dec rsi
            mov rcx, 5
            call Sleep
        jmp _while
__while:
    mov rcx, arr
    mov edx, left
    mov r8, rsi
    call quickSort
    mov rcx, arr
    mov rdx, rdi
    mov r8d, right
    call quickSort
bye:
ret
quickSort endp
```

Сортировка слиянием

```
mergeSort proc <6, 8, 4> arr:qword, len:dword
local left:qword
local right:qword
local middle:dword
sub rsp, 30h
cmp len, 1
je bye
mov eax, len
shr eax, 1  ; div 2 - middle
mov middle, eax
mov ecx, eax
mov edx, 4
call calloc ; create left arr 0-middle
mov left, rax
mov rcx, rax
mov rdx, arr
mov eax, middle
shl eax, 2     ; eax = middle * 4
mov r8d, eax
```

```
call memcpy
; --------------------
mov eax, len
sub eax, middle
mov ecx, eax
mov edx, 4
call calloc ; create right arr middle-len - 1 size = len - middle
mov right, rax
mov rcx, rax        ; right
mov eax, middle ; middle
shl eax, 2     ; middle * 4
add rax, arr   ; + base - arr arr + middle * 4
mov rdx, rax
mov eax, len  ; size
sub eax, middle ; size - middle
shl eax, 2      ; (size - middle) * 4
mov r8d, eax
call memcpy
;---------rec------------
mov rcx, left
mov edx, middle
call mergeSort
mov rcx, right
mov eax, len
sub eax, middle
mov edx, eax
call mergeSort
;-------merge------------
mov rcx, arr          ; arr
mov rdx, left         ; left
mov r8d, middle   ; left sz
mov eax, len
sub eax, middle
mov r9, right    ; right
mov qword ptr [rsp + 20h], rax  ; right sz
call _merge                                    ; merge them
;---------------free-------
mov rcx, left
call free
mov rcx, right
call free                                      ; free mem after merge
bye:
ret
mergeSort endp
```

Все представленные в статье алгоритмы сортировки реализованы в виде библиотеки alglib.dll. Это делает их готовыми к транспортировке и подключению к другим программам, значительно упрощает интеграцию и повторное использование кода.

Благодаря использованию библиотеки alglib.dll, разработчики могут легко импортировать и использовать алгоритмы сортировки в своих приложениях, не тратя время на повторное написание кода.

Сравнительный анализ производительности алгоритмов сортировки

Сравнительный анализ производительности алгоритмов сортировки позволяет оценить эффективность каждого алгоритма в различных условиях и с разными наборами данных. Производительность сортировки зависит от разных факторов, таких как размер и структура входных данных, а также способы реализации алгоритма.

В данной статье для анализа производительности используется метрику временной сложности. Замеры будут проводиться на 3 массивах разной размерности, элементы в массиве расставлены случайным образом.

Для замера времени выполнения для каждого алгоритма, была написана небольшая утилита, позволяющая задавать размер сортируемого массива. Программа имеет графический интерфейс, который позволяет наглядно сравнить скорость выполнения разных алгоритмов.

Таблица 2

Сравнительная таблица скорости выполнения разных алгоритмов

Алгоритм сортировки	**10^3 элементов**	**10^5 элементов**	**10^6 элементов**
Сортировка пузырьком	*0.0006 сек.*	21.329 сек.	—
Сортировка вставками	0.0002 сек.	1.777 сек.	—
Сортировка выбором	0.0003 сек.	4.457 сек.	—
Сортировка перемешиванием	0.0006 сек.	15.944 сек.	—
Быстрая сортировка	0 сек.	0.052 сек.	0.344 сек.
Сортировка кучей	0 сек.	0.0107 сек.	1.189 сек.
Сортировка слиянием	0 сек.	0.0108 сек.	0.794 сек.
Сортировка Шелла	0 сек.	0.0096 сек.	0.655 сек.

Визуализация процесса сортировки

Визуализация процесса сортировки является полезным и наглядным способом представления работы алгоритмов сортировки. Она позволяет наблюдать за динамикой процесса сортировки в реальном времени и лучше понимать основные принципы работы каждого алгоритма.

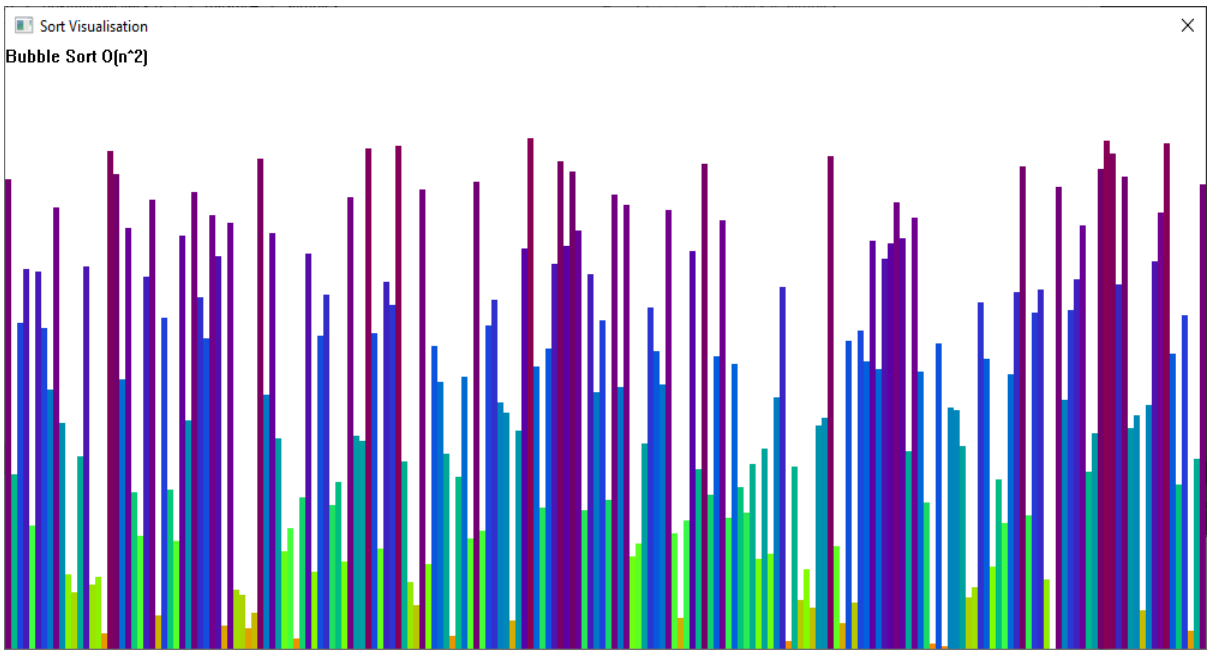

Рис.1. Элементы в массиве расставлены в случайном порядке

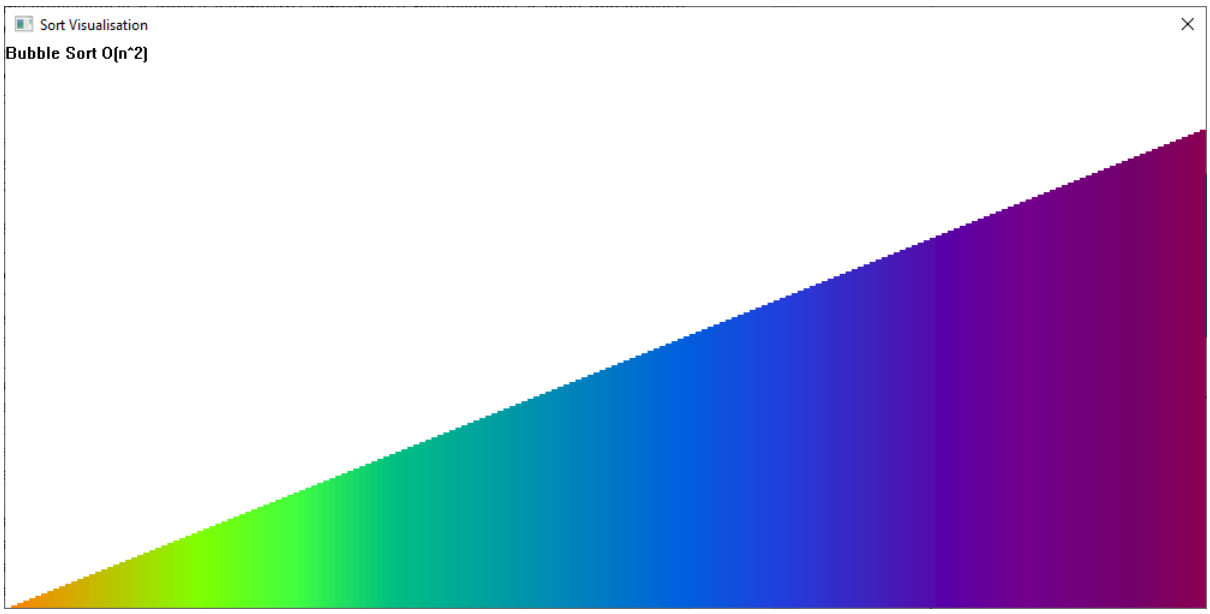

Рис.2. Отсортированный массив элементов

В данной статье представлена программа для визуализации процесса сортировки, реализованная на языке ассемблера MASM (Macro Assembler) и использующая Windows API для создания оконного приложения. Благодаря использованию библиотеки GDI (Graphics Device Interface), приложение обеспечивает графическое представление алгоритмов сортировки, позволяя наблюдать за их работой в реальном времени.

Функционал программы:

1. Визуализация массива: при запуске программы отображается массив элементов, перемешанный в случайном порядке, чтобы продемонстрировать начальное состояние данных перед сортировкой.
2. Перемешивание массива вручную: Пользователь может вручную перемешать массив, кликая правой кнопкой мыши, что позволяет управлять начальными условиями сортировки.
3. Визуализация процесса сортировки в режиме реального времени: при нажатии левой кнопки мыши программа начинает сортировку массива с небольшой временной задержкой между перестановками элементов. Это обеспечивает наглядность процесса сортировки и позволяет пользователям наблюдать за динамикой изменения массива.
4. Регулировка временной задержки: Пользователи могут управлять скоростью сортировки, настраивая временную задержку между перестановками элементов с помощью клавиш "+" и "-". Это позволяет увеличить или уменьшить скорость визуализации для более комфортного наблюдения.
5. Выбор алгоритма сортировки: В верхней части программы находится меню для выбора алгоритма сортировки. Пользователи могут выбирать между различными методами сортировки, для сравнения их эффективности и характеристик.

Заключение

Обсуждены преимущества использования языка ассемблера для оптимизации производительности алгоритмов сортировки, возможность контролировать непосредственно аппаратные ресурсы и использовать специфические инструкции процессора. Приведены примеры кода, демонстрирующие реализацию и оптимизацию каждого из представленных алгоритмов, а также результаты сравнительного анализа производительности.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Смоленцев М.Ю. Программирование на языке Ассемблера для 32/64-разрядных микропроцессоров семейства 80x86: Учебное пособие в 3-х частях. Часть 1. – Иркутск: ИрГУПС, 2009. – 192 с.

2. Кнут Д. Искусство программирования для ЭВМ. Т. 3. Сортировка и поиск. М.: Издательство «Мир», 1978. – 844 с.

3. Кормен, Томас Х., Лейзерсон, Чарльз И., Ривест, Рональд Л., Штайн, Клиффорд. Алгоритмы: построение и анализ, 2-е издание.: Пер. с англ. — М.: Издательский дом “Вильямс”, 2011. — 1296 с.: ил. — Парал. тит. англ. ISBN 978-5–8459–0857– (рус.)

4. Ниман Т. Сортировка и поиск: Рецептурный справочник. Санта-Круз, Калифорния, март 1995. — 49 с.

5. Альфред В. Ахо, Джон Е. Хопкрофт, и Джеффри Д. Ульман. Структуры данных и алгоритмы. Массачусетс: Addison-Wesley, 1983.

REFERENCES

1. Smolentsev M.Y. Assembler programming for 32/64-bit microprocessors of 80x86 family: tutorial in three parts. Part 1. – Irkutsk: IrGUPS, 2009. - 192 p.

2. Knuth D. The art of computer programming. T. 3. Sorting and searching. Moscow: Mir Publisher, 1978. – 844 p.

3. Cormen, Thomas H., Lazerson, Charles I., Rivest, Ronald L., Stein, Clifford. Algorithms: construction and analysis, 2nd ed: Ed. with English. - M.: Williams Publishing House, 2011. - 1296 p.: ill. - ISBN 978-5-8459-0857 – (Russian).

4. Niman T. Sorting and Searching: A Cookbook. Santa Cruz, California, March 1995 - 49 p.

5. Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. Data Structures and Algorithms. Massachusetts: Addison-Wesley, 1983.

Информация об авторах

Овсянников Иван Владимирович — студент 3 курса направления подготовки «Программная инженерия», Иркутский государственный университет путей сообщения, г. Иркутск, e-mail: bidanocka@gmail.com

Смоленцев Михаил Юрьевич — инженер кафедры «Автоматика, телемеханика и связь», Иркутский государственный университет путей сообщения, г. Иркутск, e-mail: miklirk@gmail.com

Information about the authors

Ovsyannikov Ivan Vladimirovich — third-year student of "Software Engineering" course, Irkutsk State Transport University, Irkutsk, e-mail: bidanocka@gmail.com

Smolentsev Mikhail Yurievich — engineer, "Automation, Telemechanics and Communications" department, Irkutsk State Transport University, Irkutsk, e-mail: miklirk@gmail.com