**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ   
РОССИЙСКОЙ ФЕДЕРАЦИИ**

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ**

**ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. ШУХОВА»**

**(БГТУ им. В.Г. Шухова)**



ИНСТИТУТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И УПРАВЛЯЮЩИХ СИСТЕМ

**Лабораторная работа №7**

по дисциплине: Архитектура вычислительных систем

тема: «Способы вызова ассемблерных подпрограмм

в языках высокого уровня»

Выполнил: ст. группы ПВ-223

Пахомов Владислав Андреевич

Проверили:

ст. пр. Осипов Олег Васильевич

Белгород 2024 г.

**Лабораторная работа №7  
Способы вызова ассемблерных подпрограмм**

**в языках высокого уровня  
Вариант 8**

**Цель работы:** изучение команд поразрядной обработки данных.

**Задания для выполнения к работе:**

1. Написать и отладить подпрограммы на masm32 в разных стилях вызова для решения задачи соответствующего варианта. Глобальные переменные в подпрограммах использовать не разрешается. Если нужна дополнительная память, выделять её в стеке.
2. Подпрограммы собрать и скомпилировать в виде dll-библиотеки. Библиотека должна содержать:
   1. подпрограммы в стилях stdcall, cdecl, fastcall, написанные на ассемблере без явного перечисления аргументов в заголовке;
   2. Подпрограммы в стилях stdcall, cdecl, написанные, наоборот, с перечислением аргументов в заголовке подпрограммы.
3. Подключить все подпрограммы из dll-библиотеки к проектам на C# и С++ статическим и динамическим способом. Убедиться в правильности вызова всех подпрограмм.
4. Написать подпрограмму для решения задачи варианта с использованием ассемблерной вставки на языке C++.
5. Написать подпрограммы для решения задачи варианта с использованием обычного высокоуровнего языка C# и C++ (или любого другого).
6. Сравнить скорость выполнения полученных подпрограмм на одних и тех же тестовых данных. Для сравнения выбрать: подпрограмму на ассемблере в masm32 (какую-нибудь одну из пяти), вызываемую из программы на языке C++ или C#; подпрограмму на C#; подпрограмму на C++; подпрограмму на С++ с использованием ассемблерной вставки. Построить на одной плоскости графики зависимости времени выполнения подпрограмм от длины массивов (не менее 10 точек для каждой подпрограммы). Для замера лучше передавать в подпрограммы массивы большой длины. Время замерять в миллисекундах с помощью API-функции GetTickCount(). Проверить, что подпрограммы при одинаковых тестовых данных выдают одинаковый результат. Для заполнения массивов использовать генератор случайных чисел.
7. В отчёт включить весь исходный код и графики.
8. Сделать выводы по работе.

**Задание:**

|  |  |
| --- | --- |
| Варианты 8 - 16  Из массива *a* длиной *length* скопировать отрицательные числа в массив *neg\_res*, положительные – в массив *pos\_res*. Под массивы *neg\_res* и *pos\_res* в основной программе зарезервировать памяти столько, сколько занимает массив *a*. Полученные массивы отсортировать. Количество отрицательных чисел записать в выходной параметр *neg\_count*, положительных – в выходной параметр *pos\_count*. Исходный массив *a* оставить без изменений. Для удобства, можно реализовать в dll-библиотеке отдельную процедуру для сортировки одного массива.  Пример: *a* = {1, 3, 4, -5, 7, -2, -1, 3, 5, -5}, *length =* 10;  *pos\_res* = {1, 3, 3, 4, 5, 7} (сортировка по не убыванию);  *neg\_res* = {-1, -2, -5, -5} (сортировка по не возрастанию);  *pos\_count* = 6;  *neg\_count =* 4. | |
| 8 | Сортировка методом вставок по не убыванию.  int sort (int\* a, int length, int\* pos\_res,  int\* neg\_res, int\* neg\_count).  pos\_count возвратить. |

Исходный код (asm):

.686

.model flat, stdcall

option casemap: none

include windows.inc

include kernel32.inc

include msvcrt.inc

includelib  kernel32.lib

includelib  msvcrt.lib

.code

DllMain proc hlnstDLL:dword, reason: dword, unused: dword

mov eax, 1

ret

DllMain endp

; ARGUMENT\_AMOUNT 12

; int sort\_stdcall\_noarg (int\* a, int length, int\* pos\_res, int\* neg\_res, int\* neg\_count)

sort\_stdcall\_noarg proc

    push ebp

    mov ecx, [esp + 8]       ; ecx = a

    mov edx, [esp + 8 + 4]   ; edx = length

    ; neg\_count = 0

    mov edi, dword ptr [esp + 8 + 16]

    mov dword ptr [edi], 0

    ; Используемые аргументы

    ; Выделяем место под локальную переменную current, j, current\_comparing

    sub esp, 12

    ; ebx будет нашим счётчиком

    mov ebx, 0

    ; Обнуление eax

    xor eax, eax

sort\_stdcall\_noarg\_loop\_a:

        ; ebp = a

        mov ebp, ecx

        ; current = a[ebx]

        mov edi, dword ptr [ebp + ebx \* 4]

        mov dword ptr [esp], edi

        ; current > 0?

        cmp dword ptr [esp], 0

        jge sort\_stdcall\_noarg\_current\_pos\_zero\_more

        jmp sort\_stdcall\_noarg\_current\_pos\_zero\_less

sort\_stdcall\_noarg\_current\_pos\_zero\_more:

            ; ebp = pos\_res

            mov ebp, [esp + 8 + 12 + 8]

            ; j = pos\_count

            mov dword ptr [esp + 4], eax

            push eax

            push edx

            mov eax, dword ptr [esp + 8 + 4]

            mov edx, 4

            mul edx

            mov dword ptr [esp + 8 + 4], eax

            pop edx

            pop eax

sort\_stdcall\_noarg\_current\_pos\_zero\_more\_loop:

                ; j <= 0? Да - выход, нет - проверяем.

                cmp dword ptr [esp + 4], 0

                je sort\_stdcall\_noarg\_current\_pos\_zero\_more\_loop\_end

                ; current\_comparing = pos\_res - 1

                lea edi, dword ptr [ebp - 4]

                ; current\_comparing = pos\_res - 1 + j

                add edi, dword ptr [esp + 4]

                ; current\_comparing = \*(pos\_res - 1 + j)

                mov edi, dword ptr [edi]

                ; current\_comparing > current ?

                cmp edi, dword ptr [esp]

                jg sort\_stdcall\_noarg\_current\_pos\_zero\_more\_loop\_current\_more

                jmp sort\_stdcall\_noarg\_current\_pos\_zero\_more\_loop\_current\_less

sort\_stdcall\_noarg\_current\_pos\_zero\_more\_loop\_current\_more:

                    push eax

                    ; eax = pos\_res[j - 1]

                    mov eax, edi

                    ; edi = pos\_res

                    mov edi, ebp

                    ; edi = pos\_res + j

                    add edi, dword ptr [esp + 8]

                    ; pos\_res[j] = pos\_res[j - 1]

                    mov dword ptr [edi], eax

                    pop eax

                    jmp sort\_stdcall\_noarg\_current\_pos\_zero\_more\_loop\_current\_end

sort\_stdcall\_noarg\_current\_pos\_zero\_more\_loop\_current\_less:

                    jmp sort\_stdcall\_noarg\_current\_pos\_zero\_more\_loop\_end

sort\_stdcall\_noarg\_current\_pos\_zero\_more\_loop\_current\_end:

                sub dword ptr [esp + 4], 4

                jmp sort\_stdcall\_noarg\_current\_pos\_zero\_more\_loop

sort\_stdcall\_noarg\_current\_pos\_zero\_more\_loop\_end:

            ; ebp = pos\_res

            mov ebp, [esp + 8 + 12 + 8]

            ; current\_comparing = pos\_res

            lea edi, dword ptr [ebp]

            ; current\_comparing = pos\_res + j

            add edi, dword ptr [esp + 4]

            push eax

            mov eax, dword ptr [esp + 4]

            mov dword ptr [edi], eax

            pop eax

            ; pos\_count++

            inc eax

            jmp sort\_stdcall\_noarg\_current\_pos\_zero\_end

sort\_stdcall\_noarg\_current\_pos\_zero\_less:

            ; ebp = neg\_res

            ; swap a

            mov edi, [esp + 8 + 12 + 16]

            xor eax, dword ptr [edi]

            xor dword ptr [edi], eax

            xor eax, dword ptr [edi]

            ; ebp = neg\_res

            mov ebp, [esp + 8 + 12 + 12]

            ; j = pos\_count

            mov dword ptr [esp + 4], eax

            push eax

            push edx

            mov eax, dword ptr [esp + 8 + 4]

            mov edx, 4

            mul edx

            mov dword ptr [esp + 8 + 4], eax

            pop edx

            pop eax

sort\_stdcall\_noarg\_current\_pos\_zero\_less\_loop:

                ; j <= 0? Да - выход, нет - проверяем.

                cmp dword ptr [esp + 4], 0

                je sort\_stdcall\_noarg\_current\_pos\_zero\_less\_loop\_end

                ; current\_comparing = pos\_res - 1

                lea edi, dword ptr [ebp - 4]

                ; current\_comparing = pos\_res - 1 + j

                add edi, dword ptr [esp + 4]

                ; current\_comparing = \*(pos\_res - 1 + j)

                mov edi, dword ptr [edi]

                ; current\_comparing > current ?

                cmp edi, dword ptr [esp]

                jg sort\_stdcall\_noarg\_current\_pos\_zero\_less\_loop\_current\_more

                jmp sort\_stdcall\_noarg\_current\_pos\_zero\_less\_loop\_current\_less

sort\_stdcall\_noarg\_current\_pos\_zero\_less\_loop\_current\_more:

                    push eax

                    ; eax = pos\_res[j - 1]

                    mov eax, edi

                    ; edi = pos\_res

                    mov edi, ebp

                    ; edi = pos\_res + j

                    add edi, dword ptr [esp + 8]

                    ; pos\_res[j] = pos\_res[j - 1]

                    mov dword ptr [edi], eax

                    pop eax

                    jmp sort\_stdcall\_noarg\_current\_pos\_zero\_less\_loop\_current\_end

sort\_stdcall\_noarg\_current\_pos\_zero\_less\_loop\_current\_less:

                    jmp sort\_stdcall\_noarg\_current\_pos\_zero\_less\_loop\_end

sort\_stdcall\_noarg\_current\_pos\_zero\_less\_loop\_current\_end:

                sub dword ptr [esp + 4], 4

                jmp sort\_stdcall\_noarg\_current\_pos\_zero\_less\_loop

sort\_stdcall\_noarg\_current\_pos\_zero\_less\_loop\_end:

            ; ebp = pos\_res

            mov ebp, [esp + 8 + 12 + 12]

            ; current\_comparing = pos\_res

            lea edi, dword ptr [ebp]

            ; current\_comparing = pos\_res + j

            add edi, dword ptr [esp + 4]

            push eax

            mov eax, dword ptr [esp + 4]

            mov dword ptr [edi], eax

            pop eax

            inc eax

            ; swap b

            mov edi, [esp + 8 + 12 + 16]

            xor eax, dword ptr [edi]

            xor dword ptr [edi], eax

            xor eax, dword ptr [edi]

            jmp sort\_stdcall\_noarg\_current\_pos\_zero\_end

sort\_stdcall\_noarg\_current\_pos\_zero\_end:

        ; ebx++

        inc ebx

        ; Если ebx < length, переходим на следующую итерацию цикла

        cmp ebx, edx

        jl sort\_stdcall\_noarg\_loop\_a

    add esp, 12

    pop ebp

    ret 5 \* 4

sort\_stdcall\_noarg endp

; int sort\_cdecl\_noarg (int\* a, int length, int\* pos\_res, int\* neg\_res, int\* neg\_count)

sort\_cdecl\_noarg proc

    push ebp

    mov ecx, [esp + 8]       ; ecx = a

    mov edx, [esp + 8 + 4]   ; edx = length

    ; neg\_count = 0

    mov edi, dword ptr [esp + 8 + 16]

    mov dword ptr [edi], 0

    ; Используемые аргументы

    ; Выделяем место под локальную переменную current, j, current\_comparing

    sub esp, 12

    ; ebx будет нашим счётчиком

    mov ebx, 0

    ; Обнуление eax

    xor eax, eax

sort\_cdecl\_noarg\_loop\_a:

        ; ebp = a

        mov ebp, ecx

        ; current = a[ebx]

        mov edi, dword ptr [ebp + ebx \* 4]

        mov dword ptr [esp], edi

        ; current > 0?

        cmp dword ptr [esp], 0

        jge sort\_cdecl\_noarg\_current\_pos\_zero\_more

        jmp sort\_cdecl\_noarg\_current\_pos\_zero\_less

sort\_cdecl\_noarg\_current\_pos\_zero\_more:

            ; ebp = pos\_res

            mov ebp, [esp + 8 + 12 + 8]

            ; j = pos\_count

            mov dword ptr [esp + 4], eax

            push eax

            push edx

            mov eax, dword ptr [esp + 8 + 4]

            mov edx, 4

            mul edx

            mov dword ptr [esp + 8 + 4], eax

            pop edx

            pop eax

sort\_cdecl\_noarg\_current\_pos\_zero\_more\_loop:

                ; j <= 0? Да - выход, нет - проверяем.

                cmp dword ptr [esp + 4], 0

                je sort\_cdecl\_noarg\_current\_pos\_zero\_more\_loop\_end

                ; current\_comparing = pos\_res - 1

                lea edi, dword ptr [ebp - 4]

                ; current\_comparing = pos\_res - 1 + j

                add edi, dword ptr [esp + 4]

                ; current\_comparing = \*(pos\_res - 1 + j)

                mov edi, dword ptr [edi]

                ; current\_comparing > current ?

                cmp edi, dword ptr [esp]

                jg sort\_cdecl\_noarg\_current\_pos\_zero\_more\_loop\_current\_more

                jmp sort\_cdecl\_noarg\_current\_pos\_zero\_more\_loop\_current\_less

sort\_cdecl\_noarg\_current\_pos\_zero\_more\_loop\_current\_more:

                    push eax

                    ; eax = pos\_res[j - 1]

                    mov eax, edi

                    ; edi = pos\_res

                    mov edi, ebp

                    ; edi = pos\_res + j

                    add edi, dword ptr [esp + 8]

                    ; pos\_res[j] = pos\_res[j - 1]

                    mov dword ptr [edi], eax

                    pop eax

                    jmp sort\_cdecl\_noarg\_current\_pos\_zero\_more\_loop\_current\_end

sort\_cdecl\_noarg\_current\_pos\_zero\_more\_loop\_current\_less:

                    jmp sort\_cdecl\_noarg\_current\_pos\_zero\_more\_loop\_end

sort\_cdecl\_noarg\_current\_pos\_zero\_more\_loop\_current\_end:

                sub dword ptr [esp + 4], 4

                jmp sort\_cdecl\_noarg\_current\_pos\_zero\_more\_loop

sort\_cdecl\_noarg\_current\_pos\_zero\_more\_loop\_end:

            ; ebp = pos\_res

            mov ebp, [esp + 8 + 12 + 8]

            ; current\_comparing = pos\_res

            lea edi, dword ptr [ebp]

            ; current\_comparing = pos\_res + j

            add edi, dword ptr [esp + 4]

            push eax

            mov eax, dword ptr [esp + 4]

            mov dword ptr [edi], eax

            pop eax

            ; pos\_count++

            inc eax

            jmp sort\_cdecl\_noarg\_current\_pos\_zero\_end

sort\_cdecl\_noarg\_current\_pos\_zero\_less:

            ; ebp = neg\_res

            ; swap a

            mov edi, [esp + 8 + 12 + 16]

            xor eax, dword ptr [edi]

            xor dword ptr [edi], eax

            xor eax, dword ptr [edi]

            ; ebp = neg\_res

            mov ebp, [esp + 8 + 12 + 12]

            ; j = pos\_count

            mov dword ptr [esp + 4], eax

            push eax

            push edx

            mov eax, dword ptr [esp + 8 + 4]

            mov edx, 4

            mul edx

            mov dword ptr [esp + 8 + 4], eax

            pop edx

            pop eax

sort\_cdecl\_noarg\_current\_pos\_zero\_less\_loop:

                ; j <= 0? Да - выход, нет - проверяем.

                cmp dword ptr [esp + 4], 0

                je sort\_cdecl\_noarg\_current\_pos\_zero\_less\_loop\_end

                ; current\_comparing = pos\_res - 1

                lea edi, dword ptr [ebp - 4]

                ; current\_comparing = pos\_res - 1 + j

                add edi, dword ptr [esp + 4]

                ; current\_comparing = \*(pos\_res - 1 + j)

                mov edi, dword ptr [edi]

                ; current\_comparing > current ?

                cmp edi, dword ptr [esp]

                jg sort\_cdecl\_noarg\_current\_pos\_zero\_less\_loop\_current\_more

                jmp sort\_cdecl\_noarg\_current\_pos\_zero\_less\_loop\_current\_less

sort\_cdecl\_noarg\_current\_pos\_zero\_less\_loop\_current\_more:

                    push eax

                    ; eax = pos\_res[j - 1]

                    mov eax, edi

                    ; edi = pos\_res

                    mov edi, ebp

                    ; edi = pos\_res + j

                    add edi, dword ptr [esp + 8]

                    ; pos\_res[j] = pos\_res[j - 1]

                    mov dword ptr [edi], eax

                    pop eax

                    jmp sort\_cdecl\_noarg\_current\_pos\_zero\_less\_loop\_current\_end

sort\_cdecl\_noarg\_current\_pos\_zero\_less\_loop\_current\_less:

                    jmp sort\_cdecl\_noarg\_current\_pos\_zero\_less\_loop\_end

sort\_cdecl\_noarg\_current\_pos\_zero\_less\_loop\_current\_end:

                sub dword ptr [esp + 4], 4

                jmp sort\_cdecl\_noarg\_current\_pos\_zero\_less\_loop

sort\_cdecl\_noarg\_current\_pos\_zero\_less\_loop\_end:

            ; ebp = pos\_res

            mov ebp, [esp + 8 + 12 + 12]

            ; current\_comparing = pos\_res

            lea edi, dword ptr [ebp]

            ; current\_comparing = pos\_res + j

            add edi, dword ptr [esp + 4]

            push eax

            mov eax, dword ptr [esp + 4]

            mov dword ptr [edi], eax

            pop eax

            inc eax

            ; swap b

            mov edi, [esp + 8 + 12 + 16]

            xor eax, dword ptr [edi]

            xor dword ptr [edi], eax

            xor eax, dword ptr [edi]

            jmp sort\_cdecl\_noarg\_current\_pos\_zero\_end

sort\_cdecl\_noarg\_current\_pos\_zero\_end:

        ; ebx++

        inc ebx

        ; Если ebx < length, переходим на следующую итерацию цикла

        cmp ebx, edx

        jl sort\_cdecl\_noarg\_loop\_a

    add esp, 12

    pop ebp

    ret

sort\_cdecl\_noarg endp

; int sort\_fastcall\_noarg (int\* a, int length, int\* pos\_res, int\* neg\_res, int\* neg\_count)

sort\_fastcall\_noarg proc

    push ebp

    ; ecx = a

    ; edx = length

    ; neg\_count = 0

    mov edi, dword ptr [esp + 8 + 8]

    mov dword ptr [edi], 0

    ; Используемые аргументы

    ; Выделяем место под локальную переменную current, j, current\_comparing

    sub esp, 12

    ; ebx будет нашим счётчиком

    mov ebx, 0

    ; Обнуление eax

    xor eax, eax

sort\_fastcall\_noarg\_loop\_a:

        ; ebp = a

        mov ebp, ecx

        ; current = a[ebx]

        mov edi, dword ptr [ebp + ebx \* 4]

        mov dword ptr [esp], edi

        ; current > 0?

        cmp dword ptr [esp], 0

        jge sort\_fastcall\_noarg\_current\_pos\_zero\_more

        jmp sort\_fastcall\_noarg\_current\_pos\_zero\_less

sort\_fastcall\_noarg\_current\_pos\_zero\_more:

            ; ebp = pos\_res

            mov ebp, [esp + 8 + 12]

            ; j = pos\_count

            mov dword ptr [esp + 4], eax

            push eax

            push edx

            mov eax, dword ptr [esp + 8 + 4]

            mov edx, 4

            mul edx

            mov dword ptr [esp + 8 + 4], eax

            pop edx

            pop eax

sort\_fastcall\_noarg\_current\_pos\_zero\_more\_loop:

                ; j <= 0? Да - выход, нет - проверяем.

                cmp dword ptr [esp + 4], 0

                je sort\_fastcall\_noarg\_current\_pos\_zero\_more\_loop\_end

                ; current\_comparing = pos\_res - 1

                lea edi, dword ptr [ebp - 4]

                ; current\_comparing = pos\_res - 1 + j

                add edi, dword ptr [esp + 4]

                ; current\_comparing = \*(pos\_res - 1 + j)

                mov edi, dword ptr [edi]

                ; current\_comparing > current ?

                cmp edi, dword ptr [esp]

                jg sort\_fastcall\_noarg\_current\_pos\_zero\_more\_loop\_current\_more

                jmp sort\_fastcall\_noarg\_current\_pos\_zero\_more\_loop\_current\_less

sort\_fastcall\_noarg\_current\_pos\_zero\_more\_loop\_current\_more:

                    push eax

                    ; eax = pos\_res[j - 1]

                    mov eax, edi

                    ; edi = pos\_res

                    mov edi, ebp

                    ; edi = pos\_res + j

                    add edi, dword ptr [esp + 8]

                    ; pos\_res[j] = pos\_res[j - 1]

                    mov dword ptr [edi], eax

                    pop eax

                    jmp sort\_fastcall\_noarg\_current\_pos\_zero\_more\_loop\_current\_end

sort\_fastcall\_noarg\_current\_pos\_zero\_more\_loop\_current\_less:

                    jmp sort\_fastcall\_noarg\_current\_pos\_zero\_more\_loop\_end

sort\_fastcall\_noarg\_current\_pos\_zero\_more\_loop\_current\_end:

                sub dword ptr [esp + 4], 4

                jmp sort\_fastcall\_noarg\_current\_pos\_zero\_more\_loop

sort\_fastcall\_noarg\_current\_pos\_zero\_more\_loop\_end:

            ; ebp = pos\_res

            mov ebp, [esp + 8 + 12]

            ; current\_comparing = pos\_res

            lea edi, dword ptr [ebp]

            ; current\_comparing = pos\_res + j

            add edi, dword ptr [esp + 4]

            push eax

            mov eax, dword ptr [esp + 4]

            mov dword ptr [edi], eax

            pop eax

            ; pos\_count++

            inc eax

            jmp sort\_fastcall\_noarg\_current\_pos\_zero\_end

sort\_fastcall\_noarg\_current\_pos\_zero\_less:

            ; ebp = neg\_res

            ; swap a

            mov edi, [esp + 8 + 12 + 8]

            xor eax, dword ptr [edi]

            xor dword ptr [edi], eax

            xor eax, dword ptr [edi]

            ; ebp = neg\_res

            mov ebp, [esp + 8 + 12 + 4]

            ; j = pos\_count

            mov dword ptr [esp + 4], eax

            push eax

            push edx

            mov eax, dword ptr [esp + 8 + 4]

            mov edx, 4

            mul edx

            mov dword ptr [esp + 8 + 4], eax

            pop edx

            pop eax

sort\_fastcall\_noarg\_current\_pos\_zero\_less\_loop:

                ; j <= 0? Да - выход, нет - проверяем.

                cmp dword ptr [esp + 4], 0

                je sort\_fastcall\_noarg\_current\_pos\_zero\_less\_loop\_end

                ; current\_comparing = pos\_res - 1

                lea edi, dword ptr [ebp - 4]

                ; current\_comparing = pos\_res - 1 + j

                add edi, dword ptr [esp + 4]

                ; current\_comparing = \*(pos\_res - 1 + j)

                mov edi, dword ptr [edi]

                ; current\_comparing > current ?

                cmp edi, dword ptr [esp]

                jg sort\_fastcall\_noarg\_current\_pos\_zero\_less\_loop\_current\_more

                jmp sort\_fastcall\_noarg\_current\_pos\_zero\_less\_loop\_current\_less

sort\_fastcall\_noarg\_current\_pos\_zero\_less\_loop\_current\_more:

                    push eax

                    ; eax = pos\_res[j - 1]

                    mov eax, edi

                    ; edi = pos\_res

                    mov edi, ebp

                    ; edi = pos\_res + j

                    add edi, dword ptr [esp + 8]

                    ; pos\_res[j] = pos\_res[j - 1]

                    mov dword ptr [edi], eax

                    pop eax

                    jmp sort\_fastcall\_noarg\_current\_pos\_zero\_less\_loop\_current\_end

sort\_fastcall\_noarg\_current\_pos\_zero\_less\_loop\_current\_less:

                    jmp sort\_fastcall\_noarg\_current\_pos\_zero\_less\_loop\_end

sort\_fastcall\_noarg\_current\_pos\_zero\_less\_loop\_current\_end:

                sub dword ptr [esp + 4], 4

                jmp sort\_fastcall\_noarg\_current\_pos\_zero\_less\_loop

sort\_fastcall\_noarg\_current\_pos\_zero\_less\_loop\_end:

            ; ebp = pos\_res

            mov ebp, [esp + 8 + 12 + 4]

            ; current\_comparing = pos\_res

            lea edi, dword ptr [ebp]

            ; current\_comparing = pos\_res + j

            add edi, dword ptr [esp + 4]

            push eax

            mov eax, dword ptr [esp + 4]

            mov dword ptr [edi], eax

            pop eax

            inc eax

            ; swap b

            mov edi, [esp + 8 + 12 + 8]

            xor eax, dword ptr [edi]

            xor dword ptr [edi], eax

            xor eax, dword ptr [edi]

            jmp sort\_fastcall\_noarg\_current\_pos\_zero\_end

sort\_fastcall\_noarg\_current\_pos\_zero\_end:

        ; ebx++

        inc ebx

        ; Если ebx < length, переходим на следующую итерацию цикла

        cmp ebx, edx

        jl sort\_fastcall\_noarg\_loop\_a

    add esp, 12

    pop ebp

    ret 3 \* 4

sort\_fastcall\_noarg endp

; int sort\_stdcall (int\* a, int length, int\* pos\_res, int\* neg\_res, int\* neg\_count)

sort\_stdcall proc stdcall a: DWORD, len: DWORD, pos\_res: DWORD, neg\_res: DWORD, neg\_count: DWORD

    push esi

    mov ecx, a       ; ecx = a

    mov edx, len   ; edx = length

    ; neg\_count = 0

    mov edi, dword ptr [neg\_count]

    mov dword ptr [edi], 0

    ; Используемые аргументы

    ; Выделяем место под локальную переменную current, j, current\_comparing

    sub esp, 12

    ; ebx будет нашим счётчиком

    mov ebx, 0

    ; Обнуление eax

    xor eax, eax

sort\_stdcall\_loop\_a:

        ; esi = a

        mov esi, ecx

        ; current = a[ebx]

        mov edi, dword ptr [esi + ebx \* 4]

        mov dword ptr [esp], edi

        ; current > 0?

        cmp dword ptr [esp], 0

        jge sort\_stdcall\_current\_pos\_zero\_more

        jmp sort\_stdcall\_current\_pos\_zero\_less

sort\_stdcall\_current\_pos\_zero\_more:

            ; esi = pos\_res

            mov esi, [pos\_res]

            ; j = pos\_count

            mov dword ptr [esp + 4], eax

            push eax

            push edx

            mov eax, dword ptr [esp + 8 + 4]

            mov edx, 4

            mul edx

            mov dword ptr [esp + 8 + 4], eax

            pop edx

            pop eax

sort\_stdcall\_current\_pos\_zero\_more\_loop:

                ; j <= 0? Да - выход, нет - проверяем.

                cmp dword ptr [esp + 4], 0

                je sort\_stdcall\_current\_pos\_zero\_more\_loop\_end

                ; current\_comparing = pos\_res - 1

                lea edi, dword ptr [esi - 4]

                ; current\_comparing = pos\_res - 1 + j

                add edi, dword ptr [esp + 4]

                ; current\_comparing = \*(pos\_res - 1 + j)

                mov edi, dword ptr [edi]

                ; current\_comparing > current ?

                cmp edi, dword ptr [esp]

                jg sort\_stdcall\_current\_pos\_zero\_more\_loop\_current\_more

                jmp sort\_stdcall\_current\_pos\_zero\_more\_loop\_current\_less

sort\_stdcall\_current\_pos\_zero\_more\_loop\_current\_more:

                    push eax

                    ; eax = pos\_res[j - 1]

                    mov eax, edi

                    ; edi = pos\_res

                    mov edi, esi

                    ; edi = pos\_res + j

                    add edi, dword ptr [esp + 8]

                    ; pos\_res[j] = pos\_res[j - 1]

                    mov dword ptr [edi], eax

                    pop eax

                    jmp sort\_stdcall\_current\_pos\_zero\_more\_loop\_current\_end

sort\_stdcall\_current\_pos\_zero\_more\_loop\_current\_less:

                    jmp sort\_stdcall\_current\_pos\_zero\_more\_loop\_end

sort\_stdcall\_current\_pos\_zero\_more\_loop\_current\_end:

                sub dword ptr [esp + 4], 4

                jmp sort\_stdcall\_current\_pos\_zero\_more\_loop

sort\_stdcall\_current\_pos\_zero\_more\_loop\_end:

            ; esi = pos\_res

            mov esi, [pos\_res]

            ; current\_comparing = pos\_res

            lea edi, dword ptr [esi]

            ; current\_comparing = pos\_res + j

            add edi, dword ptr [esp + 4]

            push eax

            mov eax, dword ptr [esp + 4]

            mov dword ptr [edi], eax

            pop eax

            ; pos\_count++

            inc eax

            jmp sort\_stdcall\_current\_pos\_zero\_end

sort\_stdcall\_current\_pos\_zero\_less:

            ; esi = neg\_res

            ; swap a

            mov edi, [neg\_count]

            xor eax, dword ptr [edi]

            xor dword ptr [edi], eax

            xor eax, dword ptr [edi]

            ; esi = neg\_res

            mov esi, [neg\_res]

            ; j = pos\_count

            mov dword ptr [esp + 4], eax

            push eax

            push edx

            mov eax, dword ptr [esp + 8 + 4]

            mov edx, 4

            mul edx

            mov dword ptr [esp + 8 + 4], eax

            pop edx

            pop eax

sort\_stdcall\_current\_pos\_zero\_less\_loop:

                ; j <= 0? Да - выход, нет - проверяем.

                cmp dword ptr [esp + 4], 0

                je sort\_stdcall\_current\_pos\_zero\_less\_loop\_end

                ; current\_comparing = pos\_res - 1

                lea edi, dword ptr [esi - 4]

                ; current\_comparing = pos\_res - 1 + j

                add edi, dword ptr [esp + 4]

                ; current\_comparing = \*(pos\_res - 1 + j)

                mov edi, dword ptr [edi]

                ; current\_comparing > current ?

                cmp edi, dword ptr [esp]

                jg sort\_stdcall\_current\_pos\_zero\_less\_loop\_current\_more

                jmp sort\_stdcall\_current\_pos\_zero\_less\_loop\_current\_less

sort\_stdcall\_current\_pos\_zero\_less\_loop\_current\_more:

                    push eax

                    ; eax = pos\_res[j - 1]

                    mov eax, edi

                    ; edi = pos\_res

                    mov edi, esi

                    ; edi = pos\_res + j

                    add edi, dword ptr [esp + 8]

                    ; pos\_res[j] = pos\_res[j - 1]

                    mov dword ptr [edi], eax

                    pop eax

                    jmp sort\_stdcall\_current\_pos\_zero\_less\_loop\_current\_end

sort\_stdcall\_current\_pos\_zero\_less\_loop\_current\_less:

                    jmp sort\_stdcall\_current\_pos\_zero\_less\_loop\_end

sort\_stdcall\_current\_pos\_zero\_less\_loop\_current\_end:

                sub dword ptr [esp + 4], 4

                jmp sort\_stdcall\_current\_pos\_zero\_less\_loop

sort\_stdcall\_current\_pos\_zero\_less\_loop\_end:

            ; esi = pos\_res

            mov esi, [neg\_res]

            ; current\_comparing = pos\_res

            lea edi, dword ptr [esi]

            ; current\_comparing = pos\_res + j

            add edi, dword ptr [esp + 4]

            push eax

            mov eax, dword ptr [esp + 4]

            mov dword ptr [edi], eax

            pop eax

            inc eax

            ; swap b

            mov edi, [neg\_count]

            xor eax, dword ptr [edi]

            xor dword ptr [edi], eax

            xor eax, dword ptr [edi]

            jmp sort\_stdcall\_current\_pos\_zero\_end

sort\_stdcall\_current\_pos\_zero\_end:

        ; ebx++

        inc ebx

        ; Если ebx < length, переходим на следующую итерацию цикла

        cmp ebx, edx

        jl sort\_stdcall\_loop\_a

    add esp, 12

    pop esi

    ret

sort\_stdcall endp

; int sort\_cdecl (int\* a, int length, int\* pos\_res, int\* neg\_res, int\* neg\_count)

sort\_cdecl proc c a: DWORD, len: DWORD, pos\_res: DWORD, neg\_res: DWORD, neg\_count: DWORD

push esi

    mov ecx, a       ; ecx = a

    mov edx, len   ; edx = length

    ; neg\_count = 0

    mov edi, dword ptr [neg\_count]

    mov dword ptr [edi], 0

    ; Используемые аргументы

    ; Выделяем место под локальную переменную current, j, current\_comparing

    sub esp, 12

    ; ebx будет нашим счётчиком

    mov ebx, 0

    ; Обнуление eax

    xor eax, eax

sort\_cdecl\_loop\_a:

        ; esi = a

        mov esi, ecx

        ; current = a[ebx]

        mov edi, dword ptr [esi + ebx \* 4]

        mov dword ptr [esp], edi

        ; current > 0?

        cmp dword ptr [esp], 0

        jge sort\_cdecl\_current\_pos\_zero\_more

        jmp sort\_cdecl\_current\_pos\_zero\_less

sort\_cdecl\_current\_pos\_zero\_more:

            ; esi = pos\_res

            mov esi, [pos\_res]

            ; j = pos\_count

            mov dword ptr [esp + 4], eax

            push eax

            push edx

            mov eax, dword ptr [esp + 8 + 4]

            mov edx, 4

            mul edx

            mov dword ptr [esp + 8 + 4], eax

            pop edx

            pop eax

sort\_cdecl\_current\_pos\_zero\_more\_loop:

                ; j <= 0? Да - выход, нет - проверяем.

                cmp dword ptr [esp + 4], 0

                je sort\_cdecl\_current\_pos\_zero\_more\_loop\_end

                ; current\_comparing = pos\_res - 1

                lea edi, dword ptr [esi - 4]

                ; current\_comparing = pos\_res - 1 + j

                add edi, dword ptr [esp + 4]

                ; current\_comparing = \*(pos\_res - 1 + j)

                mov edi, dword ptr [edi]

                ; current\_comparing > current ?

                cmp edi, dword ptr [esp]

                jg sort\_cdecl\_current\_pos\_zero\_more\_loop\_current\_more

                jmp sort\_cdecl\_current\_pos\_zero\_more\_loop\_current\_less

sort\_cdecl\_current\_pos\_zero\_more\_loop\_current\_more:

                    push eax

                    ; eax = pos\_res[j - 1]

                    mov eax, edi

                    ; edi = pos\_res

                    mov edi, esi

                    ; edi = pos\_res + j

                    add edi, dword ptr [esp + 8]

                    ; pos\_res[j] = pos\_res[j - 1]

                    mov dword ptr [edi], eax

                    pop eax

                    jmp sort\_cdecl\_current\_pos\_zero\_more\_loop\_current\_end

sort\_cdecl\_current\_pos\_zero\_more\_loop\_current\_less:

                    jmp sort\_cdecl\_current\_pos\_zero\_more\_loop\_end

sort\_cdecl\_current\_pos\_zero\_more\_loop\_current\_end:

                sub dword ptr [esp + 4], 4

                jmp sort\_cdecl\_current\_pos\_zero\_more\_loop

sort\_cdecl\_current\_pos\_zero\_more\_loop\_end:

            ; esi = pos\_res

            mov esi, [pos\_res]

            ; current\_comparing = pos\_res

            lea edi, dword ptr [esi]

            ; current\_comparing = pos\_res + j

            add edi, dword ptr [esp + 4]

            push eax

            mov eax, dword ptr [esp + 4]

            mov dword ptr [edi], eax

            pop eax

            ; pos\_count++

            inc eax

            jmp sort\_cdecl\_current\_pos\_zero\_end

sort\_cdecl\_current\_pos\_zero\_less:

            ; esi = neg\_res

            ; swap a

            mov edi, [neg\_count]

            xor eax, dword ptr [edi]

            xor dword ptr [edi], eax

            xor eax, dword ptr [edi]

            ; esi = neg\_res

            mov esi, [neg\_res]

            ; j = pos\_count

            mov dword ptr [esp + 4], eax

            push eax

            push edx

            mov eax, dword ptr [esp + 8 + 4]

            mov edx, 4

            mul edx

            mov dword ptr [esp + 8 + 4], eax

            pop edx

            pop eax

sort\_cdecl\_current\_pos\_zero\_less\_loop:

                ; j <= 0? Да - выход, нет - проверяем.

                cmp dword ptr [esp + 4], 0

                je sort\_cdecl\_current\_pos\_zero\_less\_loop\_end

                ; current\_comparing = pos\_res - 1

                lea edi, dword ptr [esi - 4]

                ; current\_comparing = pos\_res - 1 + j

                add edi, dword ptr [esp + 4]

                ; current\_comparing = \*(pos\_res - 1 + j)

                mov edi, dword ptr [edi]

                ; current\_comparing > current ?

                cmp edi, dword ptr [esp]

                jg sort\_cdecl\_current\_pos\_zero\_less\_loop\_current\_more

                jmp sort\_cdecl\_current\_pos\_zero\_less\_loop\_current\_less

sort\_cdecl\_current\_pos\_zero\_less\_loop\_current\_more:

                    push eax

                    ; eax = pos\_res[j - 1]

                    mov eax, edi

                    ; edi = pos\_res

                    mov edi, esi

                    ; edi = pos\_res + j

                    add edi, dword ptr [esp + 8]

                    ; pos\_res[j] = pos\_res[j - 1]

                    mov dword ptr [edi], eax

                    pop eax

                    jmp sort\_cdecl\_current\_pos\_zero\_less\_loop\_current\_end

sort\_cdecl\_current\_pos\_zero\_less\_loop\_current\_less:

                    jmp sort\_cdecl\_current\_pos\_zero\_less\_loop\_end

sort\_cdecl\_current\_pos\_zero\_less\_loop\_current\_end:

                sub dword ptr [esp + 4], 4

                jmp sort\_cdecl\_current\_pos\_zero\_less\_loop

sort\_cdecl\_current\_pos\_zero\_less\_loop\_end:

            ; esi = pos\_res

            mov esi, [neg\_res]

            ; current\_comparing = pos\_res

            lea edi, dword ptr [esi]

            ; current\_comparing = pos\_res + j

            add edi, dword ptr [esp + 4]

            push eax

            mov eax, dword ptr [esp + 4]

            mov dword ptr [edi], eax

            pop eax

            inc eax

            ; swap b

            mov edi, [neg\_count]

            xor eax, dword ptr [edi]

            xor dword ptr [edi], eax

            xor eax, dword ptr [edi]

            jmp sort\_cdecl\_current\_pos\_zero\_end

sort\_cdecl\_current\_pos\_zero\_end:

        ; ebx++

        inc ebx

        ; Если ebx < length, переходим на следующую итерацию цикла

        cmp ebx, edx

        jl sort\_cdecl\_loop\_a

    add esp, 12

    pop esi

    ret

sort\_cdecl endp

end DllMain

libs.def:

LIBRARY libs

EXPORTS

\_sort\_stdcall\_noarg@20 = \_sort\_stdcall\_noarg@0

\_sort\_cdecl\_noarg = \_sort\_cdecl\_noarg@0

@sort\_fastcall\_noarg@20 = \_sort\_fastcall\_noarg@0

sort\_cdecl

sort\_stdcall

Исходный тестирующий код (C++):

#include <iostream>

#include <vector>

#include <assert.h>

#include <chrono>

#pragma comment(lib, "libs.lib")

extern "C" \_\_declspec(dllimport) int \_stdcall  sort\_stdcall\_noarg  (int\* a, int length, int\* pos\_res, int\* neg\_res, int\* neg\_count);

extern "C" \_\_declspec(dllimport) int \_cdecl    sort\_cdecl\_noarg    (int\* a, int length, int\* pos\_res, int\* neg\_res, int\* neg\_count);

extern "C" \_\_declspec(dllimport) int \_fastcall sort\_fastcall\_noarg (int\* a, int length, int\* pos\_res, int\* neg\_res, int\* neg\_count);

extern "C" \_\_declspec(dllimport) int \_stdcall  sort\_stdcall        (int\* a, int length, int\* pos\_res, int\* neg\_res, int\* neg\_count);

extern "C" \_\_declspec(dllimport) int \_cdecl    sort\_cdecl          (int\* a, int length, int\* pos\_res, int\* neg\_res, int\* neg\_count);

int sort\_native(int\* a, int length, int\* pos\_res, int\* neg\_res, int\* neg\_count) {

    int pos\_count = 0;

    \*neg\_count = 0;

    for (int i = 0; i < length; i++) {

        if (a[i] > 0) {

            int j = pos\_count;

            while (j > 0 && pos\_res[j - 1] > a[i]) {

                pos\_res[j] = pos\_res[j - 1];

                j--;

            }

            pos\_res[j] = a[i];

            pos\_count++;

        } else {

            int j = \*neg\_count;

            while (j > 0 && neg\_res[j - 1] > a[i]) {

                neg\_res[j] = neg\_res[j - 1];

                j--;

            }

            neg\_res[j] = a[i];

            (\*neg\_count)++;

        }

    }

    return pos\_count;

}

template <typename TestedFunction>

void test\_function1(TestedFunction func\_to\_test) {

    int a[] = { -1, -2, -3, -4, -5, -6 };

    int pos\_res[6] = {};

    int neg\_res[6] = {};

    int neg\_count;

    auto pos\_count = func\_to\_test(a, 6, pos\_res, neg\_res, (int\*)(&neg\_count));

    assert(pos\_count == 0);

    assert(neg\_count == 6);

    assert(

        neg\_res[0] == -6 &&

        neg\_res[1] == -5 &&

        neg\_res[2] == -4 &&

        neg\_res[3] == -3 &&

        neg\_res[4] == -2 &&

        neg\_res[5] == -1);

}

template <typename TestedFunction>

void test\_function2(TestedFunction func\_to\_test) {

    int a[] = { 6, 5, 4, 3, 2, 1 };

    int pos\_res[6] = {};

    int neg\_res[6] = {};

    int neg\_count;

    auto pos\_count = func\_to\_test(a, 6, pos\_res, neg\_res, (int\*)(&neg\_count));

    assert(pos\_count == 6);

    assert(neg\_count == 0);

    assert(

        pos\_res[0] == 1 &&

        pos\_res[1] == 2 &&

        pos\_res[2] == 3 &&

        pos\_res[3] == 4 &&

        pos\_res[4] == 5 &&

        pos\_res[5] == 6);

}

template <typename TestedFunction>

void test\_function3(TestedFunction func\_to\_test) {

    int a[] = { -1, 2, -3, 3, 45, -6 };

    int pos\_res[6] = {};

    int neg\_res[6] = {};

    int neg\_count;

    auto pos\_count = func\_to\_test(a, 6, pos\_res, neg\_res, (int\*)(&neg\_count));

    assert(pos\_count == 3);

    assert(neg\_count == 3);

    assert(

        neg\_res[0] == -6 &&

        neg\_res[1] == -3 &&

        neg\_res[2] == -1 &&

        pos\_res[0] == 2 &&

        pos\_res[1] == 3 &&

        pos\_res[2] == 45);

}

template <typename TestedFunction>

void test\_function4(TestedFunction func\_to\_test) {

    int a[] = { -6, -5, -4, -3, -2, -1 };

    int pos\_res[6] = {};

    int neg\_res[6] = {};

    int neg\_count;

    auto pos\_count = func\_to\_test(a, 6, pos\_res, neg\_res, (int\*)(&neg\_count));

    assert(pos\_count == 0);

    assert(neg\_count == 6);

    assert(

        neg\_res[0] == -6 &&

        neg\_res[1] == -5 &&

        neg\_res[2] == -4 &&

        neg\_res[3] == -3 &&

        neg\_res[4] == -2 &&

        neg\_res[5] == -1);

}

template <typename TestedFunction>

void test\_function5(TestedFunction func\_to\_test) {

    int a[] = { 1, 2, 3, 4, 5, 6 };

    int pos\_res[6] = {};

    int neg\_res[6] = {};

    int neg\_count;

    auto pos\_count = func\_to\_test(a, 6, pos\_res, neg\_res, (int\*)(&neg\_count));

    assert(pos\_count == 6);

    assert(neg\_count == 0);

    assert(

        pos\_res[0] == 1 &&

        pos\_res[1] == 2 &&

        pos\_res[2] == 3 &&

        pos\_res[3] == 4 &&

        pos\_res[4] == 5 &&

        pos\_res[5] == 6);

}

template <typename TestedFunction>

void test\_function6(TestedFunction func\_to\_test) {

    int a[] = { 6, 3, 4, 2, 1, 5 };

    int pos\_res[6] = {};

    int neg\_res[6] = {};

    int neg\_count;

    auto pos\_count = func\_to\_test(a, 6, pos\_res, neg\_res, (int\*)(&neg\_count));

    assert(pos\_count == 6);

    assert(neg\_count == 0);

    assert(

        pos\_res[0] == 1 &&

        pos\_res[1] == 2 &&

        pos\_res[2] == 3 &&

        pos\_res[3] == 4 &&

        pos\_res[4] == 5 &&

        pos\_res[5] == 6);

}

template <typename TestedFunction>

void test\_function7(TestedFunction func\_to\_test) {

    int a[] = { -4, -2, -6, -1, -5, -3 };

    int pos\_res[6] = {};

    int neg\_res[6] = {};

    int neg\_count;

    auto pos\_count = func\_to\_test(a, 6, pos\_res, neg\_res, (int\*)(&neg\_count));

    assert(pos\_count == 0);

    assert(neg\_count == 6);

    assert(

        neg\_res[0] == -6 &&

        neg\_res[1] == -5 &&

        neg\_res[2] == -4 &&

        neg\_res[3] == -3 &&

        neg\_res[4] == -2 &&

        neg\_res[5] == -1);

}

template <typename TestedFunction>

void stress\_test(TestedFunction func\_to\_test, int amount) {

    srand(0);

    int \*a = (int\*)malloc(sizeof(int) \* amount);

    int \*pos\_res = (int\*)malloc(sizeof(int) \* amount);

    int \*neg\_res = (int\*)malloc(sizeof(int) \* amount);

    int neg\_count;

    for (int i = 0; i < amount; i++) {

        a[i] = rand() % 1000;

    }

    std::chrono::steady\_clock::time\_point begin = std::chrono::steady\_clock::now();

    auto pos\_count = func\_to\_test(a, amount, pos\_res, neg\_res, (int\*)(&neg\_count));

    std::chrono::steady\_clock::time\_point end = std::chrono::steady\_clock::now();

    auto delta = std::chrono::duration\_cast<std::chrono::milliseconds>(end - begin).count();

    std::cout << "Working time: " << delta / 1000.0 << std::endl;

    free(a);

    free(pos\_res);

    free(neg\_res);

}

template <typename TestedFunction>

void test\_function(TestedFunction func\_to\_test) {

    test\_function1(func\_to\_test);

    test\_function2(func\_to\_test);

    test\_function3(func\_to\_test);

    test\_function4(func\_to\_test);

    test\_function5(func\_to\_test);

    test\_function6(func\_to\_test);

    test\_function7(func\_to\_test);

    for (int i = 10000; i <= 25000; i += 1000) {

        stress\_test(func\_to\_test, i);

    }

}

int main() {

    std::cout << "Native function:" << std::endl;

    test\_function(sort\_native);

    std::cout << "\_\_cdecl auto parameters:" << std::endl;

    test\_function(sort\_cdecl);

    std::cout << "\_\_stdcall auto parameters:" << std::endl;

    test\_function(sort\_stdcall);

    std::cout << "\_\_fastcall manual parameters:" << std::endl;

    test\_function(sort\_fastcall\_noarg);

    std::cout << "\_\_stdcall manual parameters" << std::endl;

    test\_function(sort\_stdcall\_noarg);

    std::cout << "\_\_cdecl manual parameters" << std::endl;

    test\_function(sort\_cdecl\_noarg);

    return 0;

}

Графики времени выполнения:

Оранжевый – время выполнения для вручную написанного кода, синий – релизная версия функции сортировки, скомпилированная средствами Visual Studio 2022.

**Вывод:** в ходе лабораторной изучили способы вызова подпрограмм, написанных на разных языках программирования посредством dll-библиотек. В большинстве случаев скомпилированный код будет быстрее и надёжнее кода, написанного вручную.