

Quicksort 'Magnetica' - FASTEST Quicksort?!... No!

A short review by sarmayce@sarmayce.com, 2022-Mar-14



The FASTEST Quicksort is to be released, Magnetica incoming, hold on to your hats: <https://youtu.be/SSOMJ84EM1s>

// Laptop "Compressionette", Intel 'Kaby Lake' i5-7200U 3.1GHz max turbo, 36GB DDR4 2133MHz:

Performer/Keys	#1, FEW distinct	#2, MANY distinct	#3, MANYmore distinct	#4, ALL distinct	#5, ALLmore distinct	#6, ALLmax distinct
Operating System,	Windows 10,	Fedora 35,	Windows 10,	Fedora 35,	Windows 10,	Fedora 35,
Compiler, -O3	Intel v15.0	GCC 11.2.1	Intel v15.0	GCC 11.2.1	Intel v15.0	GCC 11.2.1
qsort	59 seconds	377 seconds	336 seconds	541 seconds	157 seconds	195 seconds
Magnetica v.13	31 seconds	30 seconds	202 seconds	196 seconds	88 seconds	85 seconds
Bentley-McIlroy	38 seconds	36 seconds	205 seconds	208 seconds	92 seconds	94 seconds
Crumsort	30 seconds	32 seconds	132 seconds	150 seconds	64 seconds	70 seconds

Performer/Keys	#1, FEW distinct	#2, MANY distinct	#3, MANYmore distinct	#4, ALL distinct	#5, ALLmore distinct	#6, ALLmax distinct
Operating System,	Fedora 35, GCC 11.2.1	Fedora 35, GCC 11.2.1	Fedora 35, GCC 11.2.1	Fedora 35, GCC 11.2.1	Fedora 35, GCC 11.2.1	Fedora 35, GCC 11.2.1
Compiler, -O3	instructions; IPC	instructions; IPC	instructions; IPC	instructions; IPC	instructions; IPC	instructions; IPC
qsort	3,302,993,934,921; 2.75	2,983,579,082,155; 1.75	886,263,153,476; 1.41	2,352,769,705,563; 1.38	4,527,367,288,814; 1.39	N.A.
Magnetica v.13	131,873,917,282; 1.00	658,478,895,600; 1.02	309,149,594,748; 1.08	884,297,729,161; 1.06	1,726,931,029,634; 1.08	N.A.
Bentley-McIlroy	164,915,835,204; 1.09	681,956,155,364; 1.00	322,584,089,352; 1.04	944,038,959,690; 1.02	1,719,825,062,847; 0.97	N.A.
Crumsort	312,328,497,447; 2.29	1,295,551,817,497; 2.57	603,276,911,007; 2.53	1,597,685,291,532; 2.46	3,091,001,982,856; 2.51	N.A.

// Speed Roster, (the base speed 1.00x is GLIBC's qsort):

// Bank #1: 2683/820= 3.27x = 32*150+ 70*192+ 376= 820 seconds for Crumsort
// Bank #2: 2683/1054= 2.54x = 30*196+ 85*250+ 493= 1054 seconds for Magnetica v.13
// Bank #3: 2683/1172= 2.28x = 36*208+ 94*281+ 553= 1172 seconds for Bentley-McIlroy
// Bank #4: 2683/2683= 1.00x = 377*541+195*534+1036= 2683 seconds for GLIBC's qsort

// Legend (The time is exactly the Sort process time):

// #1,FEW = **2,233,861,800** keys, of them distinct = 10; 178,708,944 bytes **22338618.WORDS.bin**; elements = 178,708,944/8 *100; // Keys are 100 times duplicated
// #2,MANY = **2,482,300,900** keys, of them distinct = 2,847,531; 24,823,016 bytes **manythesaurus.txt**; elements = 24823016 -8*1; // BuildingBlocks are size-order+1, they are 100 times duplicated
// #3,MANYmore = **1,137,582,073** keys, of them distinct = 77,275,994; 1,137,582,080 bytes **linux-5.15.25.tar**; elements = 1137582080 -8*1; // BuildingBlocks are size-order+1
// #4,ALL = **2,009,333,753** keys, of them distinct = 1,912,608,132; 2,009,333,760 bytes **Fedora-Workstation-Live-x86_64-35-1.2.iso**; elements = 2009333760 -8*1; // BuildingBlocks are size-order+1
// #5,ALLmore = **3,803,483,825** keys, of them distinct = 3,946,259,533; 3,803,483,832 bytes **Fedora-Workstation-35-1.2.aarch64.raw.xz**; elements = 3803483832 -8*1; // BuildingBlocks are size-order+1
// #6,ALLmax = **7,798,235,435** keys, of them distinct = 6,770,144,405; 7,798,235,442 bytes **math.stackexchange.com_en_all_2019-02.zim**; elements = 7798235442 -8*1; // BuildingBlocks are size-order+1

// Notes:

// - All the runs were in "Current priority class is REALTIME_PRIORITY_CLASS" for Windows and "Current priority is -20." for Linux;
// - All the runs were executed by Core 1 (i.e. SetProcessAffinityMask(GetCurrentProcess(), 1);) in Windows, not in Linux (the task was migrating between the cores);
// - Benchmark needs 32GB RAM, and 64GB for the 6th testset;
// - The whole package (except the 3rd, 4th, 5th and 6th datasets) is downloadable at: <https://forum.gb64.org/index.php?topic=3518.msg141225#msg141225>
// - 3rd dataset is downloadable at: <https://cdn.kernel.org/pub/linux/kernel/v5.x/linux-5.15.25.tar.xz>
// - 4th dataset is downloadable at: https://download.fedoraproject.org/pub/fedora/linux/releases/35/Workstation/x86_64/iso/Fedora-Workstation-Live-x86_64-35-1.2.iso
// - 5th dataset is downloadable at: <https://download.fedoraproject.org/pub/fedora/linux/releases/35/Workstation/aarch64/images/Fedora-Workstation-35-1.2.aarch64.raw.xz>
// - 6th dataset is downloadable at: https://download.kiwix.org/zim/stack_exchange/math.stackexchange.com_en_all_2019-02.zim

Enfun!

Quicksort Showdown - quicksort Magnetica partitioning vs quicksort Bentley McIlroy 3way partitioning; for more updates: <https://twitter.com/Sarmayce>

Quicksort 'Magnetica' - FASTEST Quicksort?! ... No!

A short review by sarmayce@sarmayce.com, 2022-Mar-10

// Laptop "Brutalitto", AMD 'Renoir' 4800H 4.3GHz max turbo, 64GB DDR4 3200MHz:

Performer/Keys	#1, FEW distinct	#2, MANY distinct	#3, MANYmore distinct	#4, ALL distinct	#5, ALLmore distinct	#6, ALLmax distinct
Operating System,	Windows 10,	Windows 10,	Windows 10,	Windows 10,	Windows 10,	Windows 10,
Compiler, -O3	Intel v15.0 GCC 11.2.1	Intel v15.0 GCC 11.2.1	Intel v15.0 GCC 11.2.1	Intel v15.0 GCC 11.2.1	Intel v15.0 GCC 11.2.1	Intel v15.0 GCC 11.2.1
qsort	42 seconds	45 seconds	242 seconds	280 seconds	113 seconds	131 seconds
Magnetica v.13	22 seconds	21 seconds	135 seconds	134 seconds	60 seconds	59 seconds
Bentley-McIlroy	24 seconds	24 seconds	146 seconds	142 seconds	66 seconds	64 seconds
Crumsort	20 seconds	19 seconds	91 seconds	81 seconds	44 seconds	38 seconds
Best Time (bare						
bone in-place QS):	19s for Crumsort	81s for Crumsort	38s for Crumsort	109s for Crumsort	211s for Crumsort	479s for Crumsort

// Speed Roster, (the base speed 1.00x is GLIBC's qsort):

// Rank #1: 2943/937= 3.14x = 19+ 81+ 38+109+211+ 479= 937 seconds for Crumsort
// Rank #2: 2943/1462= 2.01x = 21+134+ 59+177+349+ 722= 1462 seconds for Magnetica v.13
// Rank #3: 2943/1602= 1.83x = 24+142+ 64+193+376+ 803= 1602 seconds for Bentley-McIlroy
// Rank #4: 2943/2943= 1.00x = 45+280+131+354+695+1438= 2943 seconds for GLIBC's qsort

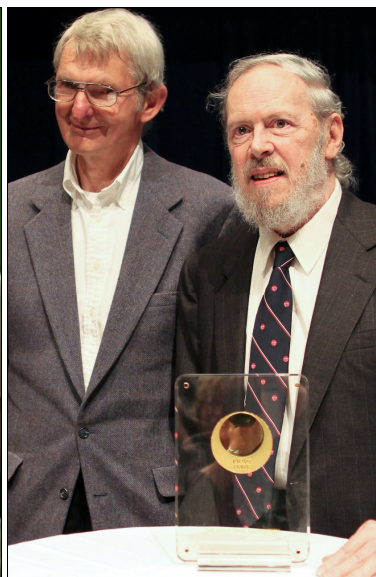
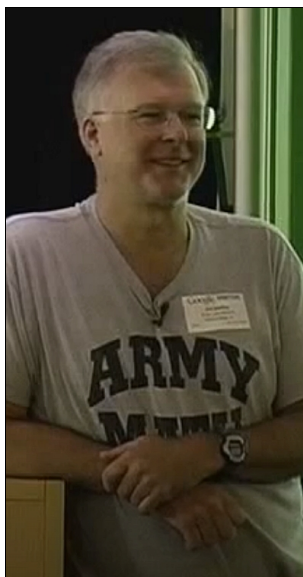
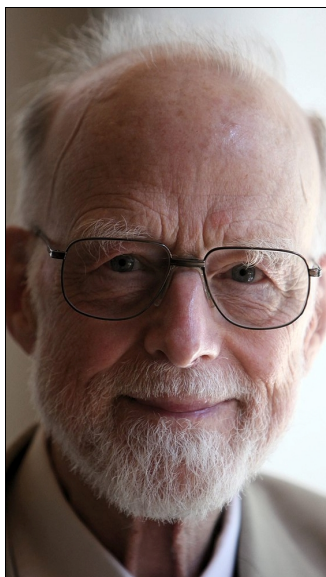
// My threads with Magnetica's source and binaries (Linux and Windows):

<https://www.overclock.net/threads/benchmark-quicksort-says-sorting-2-billion-qwords.1794855/>
<https://www.gb64.org/forum/index.php?topic=3518.msg137645#msg137645>
<https://www.linuxquestions.org/questions/programming-9/qsort-vs-%27magnetica%27-quicksort-4175703333/#post6299782>

- Open-source and 100% FREE, licenseless;
- A nifty playground for further PRACTICAL ... playoffs.

```
//The main loop of Magnetica r.5a2p [
for (;PB < Jndx;) {
    PB = PB + 1;
    if (Pivot > *PB) {
        M13_swapUnconditional (PL, PB);
        PL = PL + 1;
    } else if (Pivot < *PB) {
        for (;Pivot < *Jndx;) {
            Jndx = Jndx - 1;
        }
        M13_swapUnconditional (PB, Jndx);
        Jndx = Jndx - 1;
        PB = PB - 1;
    }
}
//The main loop of Magnetica r.5a2p ]
//if (PB > Jndx) {
//    M13_swapUnconditional (&QWORDS[PB+1], &QWORDS[Jndx+1]);
//}
M13_SwapConditional_ifBv_BUGGY((uint64_t)Jndx, (uint64_t)PB, PB+1, Jndx+1); // Should be faster than "Akkodah" but
// buggy when wraparounding, inhere 'PB' and 'Jndx' are far below the dangerous values - no pitfall.
//M13_swapConditionalIXY_Akkodah((uint64_t)Jndx, (uint64_t)PB, PB+1, Jndx+1); // The safe one, but slower, isn't
// it?!
```

```
; mark_description "Intel(R) C++ Compiler XE for applications running on Intel(R) 64,
Version 15.0.0.108 Build 20140726";
; mark_description "-Fbcs -O3 -FeQ5_bench_r13_ICL15_0_64bit.exe -D_N_HIGH_PRIORITY";
; dd-9b-2-68 bytes, 4/1 conditional/unconditional jumps; 23 instructions
; 'Magnetica' partitioning, mainloop rev.5a2p[
.B183.7::
000b 48 83 c0 08    add rax, 8
000f 4c 8b 00       mov r8, QWORD PTR [rax]
00a2 49 3b f0       cmp rsi, r8
00a5 77 26         ja .B183.14
.B183.8::
00a7 73 31         jae .B183.15
.B183.9::
00a9 48 8b 2f       mov rbp, QWORD PTR [rdi]
00ac 48 3b f5       cmp rsi, rbp
00af 73 0c         jae .B183.13
.B183.11::
00b1 48 83 c7 f8     add rdi, -8
00b5 48 8b 2f       mov rbp, QWORD PTR [rdi]
00b8 48 3b f5       cmp rsi, rbp
00bb 72 f4         jb .B183.11
.B183.13::
00bd 48 89 28       mov QWORD PTR [rax], rbp
00c0 48 83 c0 f8     add rax, -8
00c4 4c 89 07       mov QWORD PTR [rdi], r8
00c7 48 83 c7 f8     add rdi, -8
00cb eb 0d         jmp .B183.15
.B183.14::
00cd 48 8b 29       mov rbp, QWORD PTR [rcx]
00d0 4c 89 01       mov QWORD PTR [rcx], r8
00d3 48 83 c1 08     add rcx, 8
00d7 48 89 28       mov QWORD PTR [rax], rbp
.B183.15::
00da 48 3b c7       cmp rax, rdi
00dd 72 bc         jb .B183.7
; 'Magnetica' partitioning, mainloop rev.5a2p]
```



The way so far: Quicksort (1962) by Sir Antony Hoare, Bentley-McIlroy 3-way partitioning (1992) by Dr. Jon Bentley (Bell Labs) and Dr. Douglas McIlroy (Bell Labs) to the Dr. Dennis Ritchie's right, Magnetica (2021) by Sarmayce

Quicksort Showdown - [quicksort Magnetica partitioning vs quicksort Bentley McIlroy 3way partitioning](#); for more updates: <https://twitter.com/Sarmayce>

```
// Magnetica_v14.h:
```



```
// Written by Sammayce, 2022-Mar-13
```

```
#define M14_InsertionSortTHRESHOLD 0
```

```
#define M14_StackEntries 18001 //(18001-1)/2=9000 pairs/recursions
```

```
void M14_swapUnconditional(uint64_t *a, uint64_t *b) { uint64_t t = *a; *a = *b; *b = t; }
```

```
#define MINkaze(x,y) (x<y?x:y)
```

```
#define MAXkaze(x,y) (x>y?x:y)
```

```
#define MINkaze50(x,y) (y ^ ((x ^ y) & -(x < y)))
```

```
#define MAXkaze50(x,y) (x ^ ((x ^ y) & -(x < y)))
```

```
int Scandum_UINT64v(uint64_t a, uint64_t b) { return (a > b) - (a < b); } // if a G b then 1
```

```
/*
```

```
int Scandum_UINT64v(uint64_t a, uint64_t b) { return (a > b) - (a < b); }
```

```
00000 45 33 c0 xor r8d, r8d
```

```
00003 33 c0 xor eax, eax
```

```
00005 48 3b ca cmp rcx, rdx
```

```
00008 0f 97 c0 seta al
```

```
0000b 41 0f 92 c0 setb r8b
```

```
0000f 41 2b c0 sub eax, r8d
```

```
00012 c3 ret
```

```
*/
```

```
int Scandum_UINT64p(uint64_t *a, uint64_t *b) { return (*a > *b) - (*a < *b); }
```

```
/*
```

```
int Scandum_UINT64p(uint64_t *a, uint64_t *b) { return (*a > *b) - (*a < *b); }
```

```
00000 45 33 c0 xor r8d, r8d
```

```
00003 48 8b 01 mov rax, QWORD PTR [rcx]
```

```
00006 48 3b 02 cmp rax, QWORD PTR [rdx]
```

```
00009 44 89 c0 mov eax, r8d
```

```
0000c 0f 97 c0 seta al
```

```
0000f 41 0f 92 c0 setb r8b
```

```
00013 41 2b c0 sub eax, r8d
```

```
00016 c3 ret
```

```
*/
```

```
void M14_swapconditional1(uint64_t *a, uint64_t *b) { uint64_t aOLD = *a; uint64_t bOLD = *b; *a = MINkaze(aOLD,bOLD); *b = MAXkaze(aOLD,bOLD); }
```

```
/*
```

```
void M14_swapconditional1(uint64_t *a, uint64_t *b) { uint64_t aOLD = *a; uint64_t bOLD = *b; *a = MINkaze(aOLD,bOLD); *b = MAXkaze(aOLD,bOLD); }
```

```
00000 4c 8b 09 mov r9, QWORD PTR [rcx]
```

```
00003 4c 8b 02 mov r8, QWORD PTR [rdx]
```

```
00006 4d 3b c8 cmp r9, r8
```

```
00009 4c 89 c0 mov rax, r8
```

```
0000c 49 0f 42 c1 cmovb rax, r9
```

```
00010 4d 3b c1 cmp r8, r9
```

```
00013 48 89 01 mov QWORD PTR [rcx], rax
```

```
00016 4d 0f 47 c8 cmova r9, r8
```

```
0001a 4c 89 0a mov QWORD PTR [rdx], r9
```

```
0001d c3 ret
```

```
*/
```

```
void M14_swapconditional2(uint64_t *a, uint64_t *b) { uint64_t aOLD = *a; uint64_t bOLD = *b; *a = MINkaze50(aOLD,bOLD); *b = MAXkaze50(aOLD,bOLD); }
```

```
/*
```

```
void M14_swapconditional2(uint64_t *a, uint64_t *b) { uint64_t aOLD = *a; uint64_t bOLD = *b; *a = MINkaze50(aOLD,bOLD); *b = MAXkaze50(aOLD,bOLD); }
```

```
00000 33 c0 xor eax, eax
```

```
00002 4c 8b 11 mov r10, QWORD PTR [rcx]
```

```
00005 4d 89 d1 mov r9, r10
```

```
00008 4c 8b 02 mov r8, QWORD PTR [rdx]
```

```
0000b 4d 3b d0 cmp r10, r8
```

```
0000e 0f 92 c0 setb al
```

```
00011 4d 33 c8 xor r9, r8
```

```
00014 f7 d8 neg eax
```

```
00016 48 63 c0 movsxd rax, eax
```

```
00019 4c 23 c8 and r9, rax
```

```
0001c 4d 33 c1 xor r8, r9
```

```
0001f 4d 33 d1 xor r10, r9
```

```
00022 4c 89 01 mov QWORD PTR [rcx], r8
```

```
00025 4c 89 12 mov QWORD PTR [rdx], r10
```

```
00028 c3 ret
```

```
*/
```

```
void M14_SwapConditional_ifXbY_BUGGY(int64_t X, int64_t Y, uint64_t *a, uint64_t *b) { int64_t XYsgnd = X-Y; int64_t ABsgnd = *a-*b; ABsgnd = ABsgnd&(XYsgnd >> 63); *a = *a-ABsgnd; *b = *b+ABsgnd; } // if X<Y then swap
```

```
/*
```

```
void M14_SwapConditional_ifXbY_BUGGY(int64_t X, int64_t Y, uint64_t *a, uint64_t *b) { int64_t XYsgnd = X-Y; int64_t ABsgnd = *a-*b; ABsgnd = ABsgnd&(XYsgnd >> 63); *a = *a-ABsgnd; *b = *b+ABsgnd; } // if X<Y then swap
```

```
00000 48 2b ca sub rcx, rdx
```

```
00003 4d 8b 11 mov r10, QWORD PTR [r9]
```

```
*/
```

Quicksort Showdown - quicksort Magnetica partitioning vs quicksort Bentley McIlroy 3way partitioning; for more updates: <https://twitter.com/Sammayce>

```

00006 49 f7 da      neg r10
00009 49 8b 00      mov rax, QWORD PTR [r8]
0000c 4c 03 d0      add r10, rax
0000f 48 c1 f9 3f    sar rcx, 63
00013 4c 23 d1      and r10, rcx
00016 49 2b c2      sub rax, r10
00019 49 89 00      mov QWORD PTR [r8], rax
0001c 4d 01 11      add QWORD PTR [r9], r10
0001f c3            ret
*/
void M14_swapConditionalXY_Akkodah(uint64_t X, uint64_t Y, uint64_t *a, uint64_t *b) { int64_t XYsgnd = -(X<Y); int64_t ABsgnd= *a-*b; ABsgnd = ABsgnd&XYsgnd; *a = *a-ABsgnd; *b = *b+ABsgnd; } // if X<Y then swap
/*
;;; void M14_swapConditionalXY_Akkodah(uint64_t X, uint64_t Y, uint64_t *a, uint64_t *b) { int64_t XYsgnd = -(X<Y); int64_t ABsgnd= *a-*b; ABsgnd = ABsgnd&XYsgnd; *a = *a-ABsgnd; *b = *b+ABsgnd; } // if X<Y then swap
swap
00000 4d 89 ca      mov r10, r9
00003 45 33 db      xor r11d, r11d
00006 48 3b ca      cmp rcx, rdx
00009 4d 8b 08      mov r9, QWORD PTR [r8]
0000c 41 0f 92 c3     setb r11b
00010 49 8b 02      mov rax, QWORD PTR [r10]
00013 41 f7 db      neg r11d
00016 48 f7 d8      neg rax
00019 4d 63 db      movsxd r11, r11d
0001c 49 03 c1      add rax, r9
0001f 49 23 c3      and rax, r11
00022 4c 2b c8      sub r9, rax
00025 4d 89 08      mov QWORD PTR [r8], r9
00028 49 01 02      add QWORD PTR [r10], rax
0002b c3            ret
*/
void Quicksort_QB64_v14(uint64_t QWORDS[], uint64_t NumOfEle) {
    uint64_t* Stack(M14_StackEntries);
    uint64_t* Left = QWORDS;
    uint64_t* Right = &QWORDS[NumOfEle-1];
    uint64_t* TheMiddleOfMiddle;
    uint64_t* LeftBackup = Left;
    uint64_t* RightBackup = Right;
    uint64_t *Idx, *Jndx, *PL, *PR;
    uint64_t StackPtr = 0;
    int StackNotOverflow;
    register uint64_t x0,x1,x2,x3,x4,x5,x6,x7,x8;
    int o0,o1,o2,o3,o4,o5,o6,o7,o8;
    uint64_t Pivot;
    int64_t Stefan_Edelkamp_Armin_Weiss;
    int64_t Stefan_Edelkamp_Armin_Weiss_MASK, Stefan_Edelkamp_Armin_Weiss_MASKm;
    uint64_t Temporary;
    StackPtr++; Stack[StackPtr] = Left; // M14_StackEntries minimum is 3 i.e. Stack[0]=unused,Stack[1],Stack[2] i.e. one entry/pair
    StackPtr++; Stack[StackPtr] = Right;
    do {
        Right = Stack[StackPtr];
        Left = Stack[StackPtr - 1];
        StackPtr = StackPtr - 2;
        for( (M14_InsertionsortTHRESHOLD < Right-Left); ) { // if not 1<Right-Left then Right=Left (i.e. 0), or adjacent (i.e. 1) ... or Left>Right
            switch (Right-Left) { // The name of the game "jmp rax"

/*
                                case 1:      // sorting 2 elements
                                        //M14_swapconditional1(&QWORDS[Left], &QWORDS[Right]);
                                        x0 = *(Left+0);
                                        x1 = *(Left+1);
                                        o0 = (x0>x1);
                                        o1 = (0+1)-(o0);
                                        *(Left+o0)=x0;
                                        *(Left+o1)=x1;
                                        goto DirtyBypass; //break;

*/
// 11 instructions!
/*
..1.200.0.TAG.01.0.200::
.B60.37::
00e52 48 8b 1a      mov rbx, QWORD PTR [rdx]
00e55 b8 01 00 00 00 mov eax, 1
00e5a 48 8b 72 08      mov rsi, QWORD PTR [8+rdx]
00e5e 33 ed            xor ebp, ebp
00e60 48 3b de        cmp rbx, rsi
00e63 48 0f 47 e8      cmova rbp, rax
00e67 48 c1 e5 03     shl rbp, 3
00e6b 48 89 1c 2a     mov QWORD PTR [rdx+rbp], rbx
00e6f 48 2b d5       sub rdx, rbp
00e72 48 89 72 08     mov QWORD PTR [8+rdx], rsi

```



```
00e76 e9 0a f3 ff ff jmp .B60.19
```

```
*/  
/*
```

```
case 2: // sorting 3 elements  
    x0 = *(Left+0);  
    x1 = *(Left+1);  
    x2 = *(Left+2);  
    o0 = (x0>x1)+(x0>x2);  
    o1 = (x1>x0)+(x1>x2);  
    o2 = (0+1+2)-(o0+o1);  
    *(Left+o0)=x0;  
    *(Left+o1)=x1;  
    *(Left+o2)=x2;  
    goto DirtyBypass; //break;
```

```
*/  
// 23 instructions!  
/*
```

```
..1.200.0.TAG.02.0.200::
```

```
.B60.35::  
00e09 48 8b 3a      mov rdi, QWORD PTR [rdx]  
00e0c 33 db          xor ebx, ebx  
00e0e 4c 8b 42 08     mov r8, QWORD PTR [8+rdx]  
00e12 49 3b f8        cmp rdi, r8  
00e15 4c 8b 4a 10     mov r9, QWORD PTR [16+rdx]  
00e19 0f 97 c3        seta bl  
00e1c 33 c0          xor eax, eax  
00e1e 49 3b f9        cmp rdi, r9  
00e21 0f 97 c0        seta al  
00e24 33 f6          xor esi, esi  
00e26 33 ed          xor ebp, ebp  
00e28 4c 3b c7        cmp r8, rdi  
00e2b 40 0f 93 c6     setae sil  
00e2f 4d 3b c1        cmp r8, r9  
00e32 40 0f 97 c5     seta bpl  
00e36 03 d8          add ebx, eax  
00e38 03 f5          add esi, ebp  
00e3a 48 89 3c da     mov QWORD PTR [rdx+rbx*8], rdi  
00e3e 48 03 de         add rbx, rsi  
00e41 48 f7 db        neg rbx  
00e44 4c 89 04 f2     mov QWORD PTR [rdx+rsi*8], r8  
00e48 4c 89 4c da 18  mov QWORD PTR [24+rdx+rbx*8], r9  
00e4d e9 33 f3 ff ff  jmp .B60.19
```

```
*/  
/*
```

```
case 3: // sorting 4 elements  
    x0 = *(Left+0);  
    x1 = *(Left+1);  
    x2 = *(Left+2);  
    x3 = *(Left+3);  
    o0 = (x0>x1)+(x0>x2)+(x0>x3);  
    o1 = (x1>x0)+(x1>x2)+(x1>x3);  
    o2 = (x2>x0)+(x2>x1)+(x2>x3);  
    o3 = (0+1+2+3)-(o0+o1+o2);  
    *(Left+o0)=x0;  
    *(Left+o1)=x1;  
    *(Left+o2)=x2;  
    *(Left+o3)=x3;  
    goto DirtyBypass; //break;
```

```
case 4: // sorting 5 elements  
    x0 = *(Left+0);  
    x1 = *(Left+1);  
    x2 = *(Left+2);  
    x3 = *(Left+3);  
    x4 = *(Left+4);  
    o0 = (x0>x1)+(x0>x2)+(x0>x3)+(x0>x4);  
    o1 = (x1>x0)+(x1>x2)+(x1>x3)+(x1>x4);  
    o2 = (x2>x0)+(x2>x1)+(x2>x3)+(x2>x4);  
    o3 = (x3>x0)+(x3>x1)+(x3>x2)+(x3>x4);  
    o4 = (0+1+2+3+4)-(o0+o1+o2+o3);  
    *(Left+o0)=x0;  
    *(Left+o1)=x1;  
    *(Left+o2)=x2;  
    *(Left+o3)=x3;  
    *(Left+o4)=x4;  
    goto DirtyBypass; //break;
```

```
case 5: // sorting 6 elements  
    x0 = *(Left+0);  
    x1 = *(Left+1);  
    x2 = *(Left+2);
```

```

x3 = *(Left+3);
x4 = *(Left+4);
x5 = *(Left+5);
o0 = (x0>x1)+(x0>x2)+(x0>x3)+(x0>x4)+(x0>x5);
o1 = (x1>x0)+(x1>x2)+(x1>x3)+(x1>x4)+(x1>x5);
o2 = (x2>x0)+(x2>x1)+(x2>x3)+(x2>x4)+(x2>x5);
o3 = (x3>x0)+(x3>x1)+(x3>x2)+(x3>x4)+(x3>x5);
o4 = (x4>x0)+(x4>x1)+(x4>x2)+(x4>x3)+(x4>x5);
o5 = (0+1+2+3+4+5)-(o0+o1+o2+o3+o4);
*(Left+o0)=x0;
*(Left+o1)=x1;
*(Left+o2)=x2;
*(Left+o3)=x3;
*(Left+o4)=x4;
*(Left+o5)=x5;
goto DirtyBypass; //break;
case 6: // sorting 7 elements
x0 = *(Left+0);
x1 = *(Left+1);
x2 = *(Left+2);
x3 = *(Left+3);
x4 = *(Left+4);
x5 = *(Left+5);
x6 = *(Left+6);
o0 = (x0>x1)+(x0>x2)+(x0>x3)+(x0>x4)+(x0>x5)+(x0>x6);
o1 = (x1>x0)+(x1>x2)+(x1>x3)+(x1>x4)+(x1>x5)+(x1>x6);
o2 = (x2>x0)+(x2>x1)+(x2>x3)+(x2>x4)+(x2>x5)+(x2>x6);
o3 = (x3>x0)+(x3>x1)+(x3>x2)+(x3>x4)+(x3>x5)+(x3>x6);
o4 = (x4>x0)+(x4>x1)+(x4>x2)+(x4>x3)+(x4>x5)+(x4>x6);
o5 = (x5>x0)+(x5>x1)+(x5>x2)+(x5>x3)+(x5>x4)+(x5>x6);
o6 = (0+1+2+3+4+5+6)-(o0+o1+o2+o3+o4+o5);
*(Left+o0)=x0;
*(Left+o1)=x1;
*(Left+o2)=x2;
*(Left+o3)=x3;
*(Left+o4)=x4;
*(Left+o5)=x5;
*(Left+o6)=x6;
goto DirtyBypass; //break;
*/

case 13: // sorting 14 elements
M14_swapconditional1((Left+0),(Left+1)); M14_swapconditional1((Left+1),(Left+2)); M14_swapconditional1((Left+2),(Left+3));
M14_swapconditional1((Left+3),(Left+4)); M14_swapconditional1((Left+4),(Left+5)); M14_swapconditional1((Left+5),(Left+6)); M14_swapconditional1((Left+6),(Left+7)); M14_swapconditional1((Left+7),(Left+8));
M14_swapconditional1((Left+8),(Left+9)); M14_swapconditional1((Left+9),(Left+10)); M14_swapconditional1((Left+10),(Left+11)); M14_swapconditional1((Left+11),(Left+12)); M14_swapconditional1((Left+12),(Left+13));
case 12: // sorting 13 elements
M14_swapconditional1((Left+0),(Left+1)); M14_swapconditional1((Left+1),(Left+2)); M14_swapconditional1((Left+2),(Left+3));
M14_swapconditional1((Left+3),(Left+4)); M14_swapconditional1((Left+4),(Left+5)); M14_swapconditional1((Left+5),(Left+6)); M14_swapconditional1((Left+6),(Left+7)); M14_swapconditional1((Left+7),(Left+8));
M14_swapconditional1((Left+8),(Left+9)); M14_swapconditional1((Left+9),(Left+10)); M14_swapconditional1((Left+10),(Left+11)); M14_swapconditional1((Left+11),(Left+12));
case 11: // sorting 12 elements
M14_swapconditional1((Left+0),(Left+1)); M14_swapconditional1((Left+1),(Left+2)); M14_swapconditional1((Left+2),(Left+3));
M14_swapconditional1((Left+3),(Left+4)); M14_swapconditional1((Left+4),(Left+5)); M14_swapconditional1((Left+5),(Left+6)); M14_swapconditional1((Left+6),(Left+7)); M14_swapconditional1((Left+7),(Left+8));
M14_swapconditional1((Left+8),(Left+9)); M14_swapconditional1((Left+9),(Left+10)); M14_swapconditional1((Left+10),(Left+11));
case 10: // sorting 11 elements
M14_swapconditional1((Left+0),(Left+1)); M14_swapconditional1((Left+1),(Left+2)); M14_swapconditional1((Left+2),(Left+3));
M14_swapconditional1((Left+3),(Left+4)); M14_swapconditional1((Left+4),(Left+5)); M14_swapconditional1((Left+5),(Left+6)); M14_swapconditional1((Left+6),(Left+7)); M14_swapconditional1((Left+7),(Left+8));
M14_swapconditional1((Left+8),(Left+9)); M14_swapconditional1((Left+9),(Left+10));
case 9: // sorting 10 elements
M14_swapconditional1((Left+0),(Left+1)); M14_swapconditional1((Left+1),(Left+2)); M14_swapconditional1((Left+2),(Left+3));
M14_swapconditional1((Left+3),(Left+4)); M14_swapconditional1((Left+4),(Left+5)); M14_swapconditional1((Left+5),(Left+6)); M14_swapconditional1((Left+6),(Left+7)); M14_swapconditional1((Left+7),(Left+8));
M14_swapconditional1((Left+8),(Left+9));
case 8: // sorting 9 elements
M14_swapconditional1((Left+0),(Left+1)); M14_swapconditional1((Left+1),(Left+2)); M14_swapconditional1((Left+2),(Left+3));
M14_swapconditional1((Left+3),(Left+4)); M14_swapconditional1((Left+4),(Left+5)); M14_swapconditional1((Left+5),(Left+6)); M14_swapconditional1((Left+6),(Left+7)); M14_swapconditional1((Left+7),(Left+8));
case 7: // sorting 8 elements
M14_swapconditional1((Left+0),(Left+1)); M14_swapconditional1((Left+1),(Left+2)); M14_swapconditional1((Left+2),(Left+3));
M14_swapconditional1((Left+3),(Left+4)); M14_swapconditional1((Left+4),(Left+5)); M14_swapconditional1((Left+5),(Left+6)); M14_swapconditional1((Left+6),(Left+7));
case 6: // sorting 7 elements
M14_swapconditional1((Left+0),(Left+1)); M14_swapconditional1((Left+1),(Left+2)); M14_swapconditional1((Left+2),(Left+3));
M14_swapconditional1((Left+3),(Left+4)); M14_swapconditional1((Left+4),(Left+5)); M14_swapconditional1((Left+5),(Left+6));
case 5: // sorting 6 elements
M14_swapconditional1((Left+0),(Left+1)); M14_swapconditional1((Left+1),(Left+2)); M14_swapconditional1((Left+2),(Left+3));
M14_swapconditional1((Left+3),(Left+4)); M14_swapconditional1((Left+4),(Left+5));
case 4: // sorting 5 elements
M14_swapconditional1((Left+0),(Left+1)); M14_swapconditional1((Left+1),(Left+2)); M14_swapconditional1((Left+2),(Left+3));
M14_swapconditional1((Left+3),(Left+4));
case 3: // sorting 4 elements
M14_swapconditional1((Left+0),(Left+1)); M14_swapconditional1((Left+1),(Left+2)); M14_swapconditional1((Left+2),(Left+3));
case 2: // sorting 3 elements
M14_swapconditional1((Left+0),(Left+1)); M14_swapconditional1((Left+1),(Left+2));

```

```

case 1: // sorting 2 elements
    M14_swapconditional1((Left+0),(Left+1));
    goto DirtyBypass; //break;

/*

case 7: // sorting 8 elements
    x0 = *(Left+0);
    x1 = *(Left+1);
    x2 = *(Left+2);
    x3 = *(Left+3);
    x4 = *(Left+4);
    x5 = *(Left+5);
    x6 = *(Left+6);
    x7 = *(Left+7);
    o0 = (x0>x1)+(x0>x2)+(x0>x3)+(x0>x4)+(x0>x5)+(x0>x6)+(x0>x7);
    o1 = (x1>x0)+(x1>x2)+(x1>x3)+(x1>x4)+(x1>x5)+(x1>x6)+(x1>x7);
    o2 = (x2>x0)+(x2>x1)+(x2>x3)+(x2>x4)+(x2>x5)+(x2>x6)+(x2>x7);
    o3 = (x3>x0)+(x3>x1)+(x3>x2)+(x3>x4)+(x3>x5)+(x3>x6)+(x3>x7);
    o4 = (x4>x0)+(x4>x1)+(x4>x2)+(x4>x3)+(x4>x5)+(x4>x6)+(x4>x7);
    o5 = (x5>x0)+(x5>x1)+(x5>x2)+(x5>x3)+(x5>x4)+(x5>x6)+(x5>x7);
    o6 = (x6>x0)+(x6>x1)+(x6>x2)+(x6>x3)+(x6>x4)+(x6>x5)+(x6>x7);
    o7 = (0+1+2+3+4+5+6+7)-(o0+o1+o2+o3+o4+o5+o6);
    *(Left+o0)=x0;
    *(Left+o1)=x1;
    *(Left+o2)=x2;
    *(Left+o3)=x3;
    *(Left+o4)=x4;
    *(Left+o5)=x5;
    *(Left+o6)=x6;
    *(Left+o7)=x7;
    goto DirtyBypass; //break;

case 8: // sorting 9 elements
    x0 = *(Left+0);
    x1 = *(Left+1);
    x2 = *(Left+2);
    x3 = *(Left+3);
    x4 = *(Left+4);
    x5 = *(Left+5);
    x6 = *(Left+6);
    x7 = *(Left+7);
    x8 = *(Left+8);
    o0 = (x0>x1)+(x0>x2)+(x0>x3)+(x0>x4)+(x0>x5)+(x0>x6)+(x0>x7)+(x0>x8);
    o1 = (x1>x0)+(x1>x2)+(x1>x3)+(x1>x4)+(x1>x5)+(x1>x6)+(x1>x7)+(x1>x8);
    o2 = (x2>x0)+(x2>x1)+(x2>x3)+(x2>x4)+(x2>x5)+(x2>x6)+(x2>x7)+(x2>x8);
    o3 = (x3>x0)+(x3>x1)+(x3>x2)+(x3>x4)+(x3>x5)+(x3>x6)+(x3>x7)+(x3>x8);
    o4 = (x4>x0)+(x4>x1)+(x4>x2)+(x4>x3)+(x4>x5)+(x4>x6)+(x4>x7)+(x4>x8);
    o5 = (x5>x0)+(x5>x1)+(x5>x2)+(x5>x3)+(x5>x4)+(x5>x6)+(x5>x7)+(x5>x8);
    o6 = (x6>x0)+(x6>x1)+(x6>x2)+(x6>x3)+(x6>x4)+(x6>x5)+(x6>x7)+(x6>x8);
    o7 = (x7>x0)+(x7>x1)+(x7>x2)+(x7>x3)+(x7>x4)+(x7>x5)+(x7>x6)+(x7>x8);
    o8 = (0+1+2+3+4+5+6+7+8)-(o0+o1+o2+o3+o4+o5+o6+o7);
    *(Left+o0)=x0;
    *(Left+o1)=x1;
    *(Left+o2)=x2;
    *(Left+o3)=x3;
    *(Left+o4)=x4;
    *(Left+o5)=x5;
    *(Left+o6)=x6;
    *(Left+o7)=x7;
    *(Left+o8)=x8;
    goto DirtyBypass; //break;

```

```

*/
// 289 instructions!
/*
..1.200.0.TAG.08.0.200::
.B60.23::
001c8 48 89 8c 24 a8      mov QWORD PTR [144040+rsp], rcx
      32 02 00
001d0 33 ed                xor ebp, ebp
001d2 48 8b 0a          mov rcx, QWORD PTR [rdx]
001d5 45 33 d2          xor r10d, r10d
001d8 48 8b 7a 18        mov rdi, QWORD PTR [24+rdx]
001dc 48 3b cf          cmp rcx, rdi
001df 4c 8b 5a 20        mov r11, QWORD PTR [32+rdx]
001e3 40 0f 97 c5       seta bpl
001e7 45 33 ed          xor r13d, r13d
001ea 49 3b cb          cmp rcx, r11
001ed 4c 8b 4a 28        mov r9, QWORD PTR [40+rdx]
001f1 41 0f 97 c2       seta r10b
001f5 45 33 f6          xor r14d, r14d
001f8 49 3b c9          cmp rcx, r9

```

```

001fb 4c 8b 62 30    mov r12, QWORD PTR [48+rdx]
001ff 41 0f 97 c5    seta r13b
00203 45 33 ff         xor r15d, r15d
00206 49 3b cc         cmp rcx, r12
00209 48 8b 5a 38    mov rbx, QWORD PTR [56+rdx]
0020d 41 0f 97 c6    seta r14b
00211 41 03 ea       add ebp, r10d
00214 45 03 ee       add r13d, r14d
00217 45 33 f6     xor r14d, r14d
0021a 41 03 ed       add ebp, r13d
0021d 45 33 ed     xor r13d, r13d
00220 48 3b cb       cmp rcx, rbx
00223 48 8b 42 40    mov rax, QWORD PTR [64+rdx]
00227 41 0f 97 c5    seta r13b
0022b 45 33 d2     xor r10d, r10d
0022e 48 3b c8       cmp rcx, rax
00231 48 8b 72 08    mov rsi, QWORD PTR [8+rdx]
00235 41 0f 97 c7    seta r15b
00239 48 3b ce       cmp rcx, rsi
0023c 4c 8b 42 10    mov r8, QWORD PTR [16+rdx]
00240 41 0f 97 c6    seta r14b
00244 49 3b c8       cmp rcx, r8
00247 48 89 84 24 b0
32 02 00          mov QWORD PTR [144048+rsp], rax
0024f 41 0f 97 c2    seta r10b
00253 45 03 ef       add r13d, r15d
00256 45 03 f2       add r14d, r10d
00259 45 33 d2     xor r10d, r10d
0025c 48 3b f1       cmp rsi, rcx
0025f 41 0f 93 c2    setae r10b
00263 45 33 ff     xor r15d, r15d
00266 48 3b f7       cmp rsi, rdi
00269 41 0f 97 c7    seta r15b
0026d 45 03 ee       add r13d, r14d
00270 41 03 ed       add ebp, r13d
00273 45 33 ed     xor r13d, r13d
00276 49 3b f3       cmp rsi, r11
00279 41 0f 97 c5    seta r13b
0027d 45 33 f6     xor r14d, r14d
00280 49 3b f1       cmp rsi, r9
00283 41 0f 97 c6    seta r14b
00287 45 03 d7       add r10d, r15d
0028a 45 33 ff     xor r15d, r15d
0028d 49 3b f4       cmp rsi, r12
00290 41 0f 97 c7    seta r15b
00294 45 03 ee       add r13d, r14d
00297 45 33 f6     xor r14d, r14d
0029a 48 3b f3       cmp rsi, rbx
0029d 48 89 0c ea    mov QWORD PTR [rdx+rbp*8], rcx
002a1 41 0f 97 c6    seta r14b
002a5 45 03 d5       add r10d, r13d
002a8 45 03 fe       add r15d, r14d
002ab 45 33 f6     xor r14d, r14d
002ae 48 3b f0       cmp rsi, rax
002b1 41 0f 97 c6    seta r14b
002b5 45 33 ed     xor r13d, r13d
002b8 49 3b f0       cmp rsi, r8
002bb 41 0f 97 c5    seta r13b
002bf 45 03 f5       add r14d, r13d
002c2 45 33 ed     xor r13d, r13d
002c5 45 03 fe       add r15d, r14d
002c8 45 33 f6     xor r14d, r14d
002cb 4c 3b c1       cmp r8, rcx
002ce 41 0f 93 c6    setae r14b
002d2 4c 3b c6       cmp r8, rsi
002d5 41 0f 93 c5    setae r13b
002d9 45 03 d7       add r10d, r15d
002dc 4c 3b c7       cmp r8, rdi
002df 4c 89 94 24 b8
32 02 00          mov QWORD PTR [144056+rsp], r10
002e7 41 ba 00 00 00
00          mov r10d, 0
002ed 41 0f 97 c2    seta r10b
002f1 45 33 ff     xor r15d, r15d
002f4 4d 3b c3       cmp r8, r11
002f7 41 0f 97 c7    seta r15b
002fb 45 03 f5       add r14d, r13d
002fe 45 03 d7       add r10d, r15d
00301 45 33 ed     xor r13d, r13d

```



```

00304 45 03 f2      add r14d, r10d
00307 45 33 d2      xor r10d, r10d
0030a 4d 3b c1      cmp r8, r9
0030d 41 0f 97 c2    seta r10b
00311 45 33 ff      xor r15d, r15d
00314 4d 3b c4      cmp r8, r12
00317 41 0f 97 c5    seta r13b
0031b 45 03 d5      add r10d, r13d
0031e 45 33 ed      xor r13d, r13d
00321 4c 3b c3      cmp r8, rbx
00324 41 0f 97 c5    seta r13b
00328 4c 3b c0      cmp r8, rax
0032b 41 0f 97 c7    seta r15b
0032f 45 03 ef      add r13d, r15d
00332 45 33 ff      xor r15d, r15d
00335 45 03 d5      add r10d, r13d
00338 45 33 ed      xor r13d, r13d
0033b 49 3b fb      cmp rdi, r11
0033e 41 0f 97 c5    seta r13b
00342 45 03 f2      add r14d, r10d
00345 45 33 d2      xor r10d, r10d
00348 49 3b f9      cmp rdi, r9
0034b 41 0f 97 c2    seta r10b
0034f 45 03 ea      add r13d, r10d
00352 45 33 d2      xor r10d, r10d
00355 49 3b fc      cmp rdi, r12
00358 41 0f 97 c2    seta r10b
0035c 48 3b fb      cmp rdi, rbx
0035f 4c 89 b4 24 c0
32 02 00          mov QWORD PTR [144064+rsp], r14
00367 41 0f 97 c7    seta r15b
0036b 45 33 f6      xor r14d, r14d
0036e 48 3b f8      cmp rdi, rax
00371 41 0f 97 c6    seta r14b
00375 45 03 d7      add r10d, r15d
00378 45 33 ff      xor r15d, r15d
0037b 48 3b f9      cmp rdi, rcx
0037e 41 0f 93 c7    setae r15b
00382 45 03 ea      add r13d, r10d
00385 45 03 f7      add r14d, r15d
00388 45 33 ff      xor r15d, r15d
0038b 48 3b fe      cmp rdi, rsi
0038e 41 0f 93 c7    setae r15b
00392 45 33 d2      xor r10d, r10d
00395 49 3b f8      cmp rdi, r8
00398 41 0f 93 c2    setae r10b
0039c 45 03 fa      add r15d, r10d
0039f 45 33 d2      xor r10d, r10d
003a2 4c 3b df      cmp r11, rdi
003a5 41 0f 93 c2    setae r10b
003a9 45 03 f7      add r14d, r15d
003ac 45 03 ee      add r13d, r14d
003af 45 33 f6      xor r14d, r14d
003b2 4d 3b d9      cmp r11, r9
003b5 41 0f 97 c6    seta r14b
003b9 45 33 ff      xor r15d, r15d
003bc 45 03 d6      add r10d, r14d
003bf 45 33 f6      xor r14d, r14d
003c2 4d 3b dc      cmp r11, r12
003c5 41 0f 97 c6    seta r14b
003c9 4c 3b db      cmp r11, rbx
003cc 41 0f 97 c7    seta r15b
003d0 45 03 f7      add r14d, r15d
003d3 45 33 ff      xor r15d, r15d
003d6 4c 3b d8      cmp r11, rax
003d9 41 0f 97 c7    seta r15b
003dd 4c 3b d9      cmp r11, rcx
003e0 4c 89 ac 24 c8
32 02 00          mov QWORD PTR [144072+rsp], r13
003e8 41 bd 00 00 00
00          mov r13d, 0
003ee 41 0f 93 c5    setae r13b
003f2 45 03 d6      add r10d, r14d
003f5 45 03 fd      add r15d, r13d
003f8 45 33 ed      xor r13d, r13d
003fb 4c 3b de      cmp r11, rsi
003fe 41 0f 93 c5    setae r13b
00402 45 33 f6      xor r14d, r14d
00405 4d 3b d8      cmp r11, r8

```

```

00408 41 0f 93 c6    setae r14b
0040c 45 03 ee        add r13d, r14d
0040f 45 33 f6        xor r14d, r14d
00412 45 03 fd        add r15d, r13d
00415 45 33 ed        xor r13d, r13d
00418 4c 3b cf        cmp r9, rdi
0041b 41 0f 93 c5    setae r13b
0041f 4d 3b cb        cmp r9, r11
00422 41 0f 93 c6    setae r14b
00426 45 03 d7        add r10d, r15d
00429 4d 3b cc        cmp r9, r12
0042c 4c 89 94 24 d0 32 02 00    mov QWORD PTR [144080+rsp], r10
00434 41 ba 00 00 00 00    mov r10d, 0
0043a 41 0f 97 c2      seta r10b
0043e 45 33 ff        xor r15d, r15d
00441 4c 3b cb        cmp r9, rbx
00444 41 0f 97 c7      seta r15b
00448 45 03 ee        add r13d, r14d
0044b 45 33 f6        xor r14d, r14d
0044e 4c 3b c8        cmp r9, rax
00451 41 0f 97 c6      seta r14b
00455 45 03 d7        add r10d, r15d
00458 45 33 ff        xor r15d, r15d
0045b 4c 3b c9        cmp r9, rcx
0045e 41 0f 93 c7    setae r15b
00462 45 03 ea        add r13d, r10d
00465 45 03 f7        add r14d, r15d
00468 45 33 ff        xor r15d, r15d
0046b 4c 3b ce        cmp r9, rsi
0046e 41 0f 93 c7    setae r15b
00472 45 33 d2        xor r10d, r10d
00475 4d 3b c8        cmp r9, r8
00478 41 0f 93 c2      setae r10b
0047c 45 03 fa        add r15d, r10d
0047f 45 33 d2        xor r10d, r10d
00482 4c 3b e7        cmp r12, rdi
00485 41 0f 93 c2      setae r10b
00489 45 03 f7        add r14d, r15d
0048c 45 03 ee        add r13d, r14d
0048f 45 33 f6        xor r14d, r14d
00492 4d 3b e3        cmp r12, r11
00495 41 0f 93 c6      setae r14b
00499 45 33 ff        xor r15d, r15d
0049c 4d 3b e1        cmp r12, r9
0049f 41 0f 93 c7      setae r15b
004a3 4c 3b e3        cmp r12, rbx
004a6 4c 89 ac 24 d8 32 02 00    mov QWORD PTR [144088+rsp], r13
004ae 41 bd 00 00 00 00    mov r13d, 0
004b4 41 0f 97 c5      seta r13b
004b8 45 03 d6        add r10d, r14d
004bb 45 03 fd        add r15d, r13d
004be 45 33 ed        xor r13d, r13d
004c1 4c 3b e0        cmp r12, rax
004c4 41 0f 97 c5      seta r13b
004c8 45 33 f6        xor r14d, r14d
004cb 4c 3b e1        cmp r12, rcx
004ce 41 0f 93 c6      setae r14b
004d2 45 03 d7        add r10d, r15d
004d5 45 03 ee        add r13d, r14d
004d8 45 33 f6        xor r14d, r14d
004db 4c 3b e6        cmp r12, rsi
004de 41 0f 93 c6      setae r14b
004e2 45 33 ff        xor r15d, r15d
004e5 4d 3b e0        cmp r12, r8
004e8 41 0f 93 c7      setae r15b
004ec 45 03 f7        add r14d, r15d
004ef 45 33 ff        xor r15d, r15d
004f2 45 03 ee        add r13d, r14d
004f5 45 33 f6        xor r14d, r14d
004f8 48 3b df        cmp rbx, rdi
004fb 41 0f 93 c6      setae r14b
004ff 45 03 d5        add r10d, r13d
00502 45 33 ed        xor r13d, r13d
00505 49 3b db        cmp rbx, r11
00508 41 0f 93 c5      setae r13b

```

```

0050c 45 03 f5      add r14d, r13d
0050f 45 33 ed      xor r13d, r13d
00512 49 3b d9      cmp rbx, r9
00515 41 0f 93 c5   setae r13b
00519 49 3b dc      cmp rbx, r12
0051c 41 0f 93 c7   setae r15b
00520 45 03 ef      add r13d, r15d
00523 45 33 ff      xor r15d, r15d
00526 48 3b d8      cmp rbx, rax
00529 41 0f 97 c7   seta r15b
0052d 33 c0        xor eax, eax
0052f 48 3b d9      cmp rbx, rcx
00532 48 8b 8c 24 c0
          32 02 00      mov rcx, QWORD PTR [144064+rsp]
0053a 0f 93 c0      setae al
0053d 45 03 f5      add r14d, r13d
00540 44 03 f8      add r15d, eax
00543 33 c0        xor eax, eax
00545 48 3b de      cmp rbx, rsi
00548 0f 93 c0      setae al
0054b 45 33 ed      xor r13d, r13d
0054e 49 3b d8      cmp rbx, r8
00551 41 0f 93 c5   setae r13b
00555 41 03 c5      add eax, r13d
00558 4c 8b ac 24 b8
          32 02 00      mov r13, QWORD PTR [144056+rsp]
00560 44 03 f8      add r15d, eax
00563 45 03 f7      add r14d, r15d
00566 49 03 ed      add rbp, r13
00569 4a 89 34 ea   mov QWORD PTR [rdx+r13*8], rsi
0056d 4c 89 04 ca   mov QWORD PTR [rdx+rcx*8], r8
00571 4c 8b 84 24 c8
          32 02 00      mov r8, QWORD PTR [144072+rsp]
00579 49 03 c8      add rcx, r8
0057c 48 8b b4 24 d0
          32 02 00      mov rsi, QWORD PTR [144080+rsp]
00584 48 03 e9      add rbp, rcx
00587 48 8b 8c 24 a8
          32 02 00      mov rcx, QWORD PTR [144040+rsp]
0058f 4a 89 3c c2   mov QWORD PTR [rdx+r8*8], rdi
00593 4c 89 1c f2   mov QWORD PTR [rdx+rsi*8], r11
00597 4c 8b 9c 24 d8
          32 02 00      mov r11, QWORD PTR [144088+rsp]
0059f 49 03 f3      add rsi, r11
005a2 4e 89 0c da   mov QWORD PTR [rdx+r11*8], r9
005a6 4e 89 24 d2   mov QWORD PTR [rdx+r10*8], r12
005aa 4d 03 d6      add r10, r14
005ad 49 03 f2      add rsi, r10
005b0 48 03 ee      add rbp, rsi
005b3 48 f7 dd      neg rbp
005b6 4a 89 1c f2   mov QWORD PTR [rdx+r14*8], rbx
005ba 48 8b 9c 24 b0
          32 02 00      mov rbx, QWORD PTR [144048+rsp]
005c2 48 89 9c ea 20
          01 00 00      mov QWORD PTR [288+rdx+rbp*8], rbx
005ca e9 b6 fb ff ff jmp .B60.19

```

*/

default:

Jndx = Right;

PL = Left+1;

PR = Left+1;

//TheMiddleOfMiddle = Left + ((Right-Left)>>2); // (4Left + Right - Left)/4; Caution: [TheMiddleOfMiddle*6] should be <= [Right] i.e. 6*(4Left + Right - Left)/4 <= Right

or 4*(4Left + Right - Left) <= 4Right or 4*(4Left + Right - Left) <= 4Right or (4*(4Left + Right - Left) <= 4Right) <= 4Right i.e. 8 <= Right-Left as a minimum condition to enter the 'for'.

M14_swapUnconditional (Left + ((Right-Left)>>2), Left+0);

M14_swapUnconditional (Left + ((Right-Left)>>1), Left+1);

x0 = *(Left+0);

x1 = *(Left+1);

x2 = *(Left+2);

o0 = (x0>x1)?(x0>x2);

o1 = (x1>x0)?(x1>x2);

o2 = (o1+1)-(o0+o1);

*(Left+o0)=x0;

*(Left+o1)=x1;

*(Left+o2)=x2;

PL = PL - (*(Left+1)==*(Left+0));

//PL = PL - (*(Left+2)==*(Left+1)); // trying to expand the Pivot Pool (i.e. PL..PR) is kinda not worth it, therefore commented

//PL = PL - (*(Left+2)==*(Left+0)); // trying to expand the Pivot Pool (i.e. PL..PR) is kinda not worth it, therefore commented

/*

```

000c9 45 33 c0      xor r8d, r8d

```

```

000cc 48 8b 42 10    mov rax, QWORD PTR [16+rdx]
000d0 48 3b 42 08    cmp rax, QWORD PTR [8+rdx]
000d4 45 0f 44 f7    cmovbe r14d, r15d
000d8 48 3b 02       cmp rax, QWORD PTR [rdx]
000db 45 0f 44 c7    cmovbe r8d, r15d
000df 49 f7 de       neg r14
000e2 4d 2b f0       sub r14, r8
*/
                                //(5-2)=0
                                //(2-5)=0
                                //(3-3)=1
                                //PL = PL - !(*Left+2)-(*Left+1); // trying to expand the Pivot Pool (i.e. PL..PR) is kinda not worth it, therefore commented
                                //PL = PL - !(*Left+2)-(*Left+0); // trying to expand the Pivot Pool (i.e. PL..PR) is kinda not worth it, therefore commented
/*
000c9 45 33 c0       xor r8d, r8d
000cc 48 8b 42 10    mov rax, QWORD PTR [16+rdx]
000d0 48 3b 42 08    cmp rax, QWORD PTR [8+rdx]
000d4 45 0f 44 f7    cmovbe r14d, r15d
000d8 48 3b 02       cmp rax, QWORD PTR [rdx]
000db 45 0f 44 c7    cmovbe r8d, r15d
000df 49 f7 de       neg r14
000e2 4d 2b f0       sub r14, r8
*/

//Before: x0,x1,x2,x3,x4,x5,x6 = 7,1,2,4,0,3,3
//Before: o0,o1,o2,o3,o4,o5,o6 = 6 1 2 5 0 3 4
//After : s0,s1,s2,s3,s4,s5,s6 = 0,1,2,3,3,4,7
//PL,PR = 1D980050,1D980050 // 50-50=00 i.e. 0/8=0 or 7-7=0

//Before: x0,x1,x2,x3,x4,x5,x6 = 7,7,7,4,0,3,3
//Before: o0,o1,o2,o3,o4,o5,o6 = 4 5 6 3 0 1 2
//After : s0,s1,s2,s3,s4,s5,s6 = 0,3,3,4,7,7,7
//PL,PR = 1D980040,1D980050 // 50-40=10 i.e. 16/8=2 or 7-5=2

//Before: x0,x1,x2,x3,x4,x5,x6 = 7,7,7,7,7,7,7
//Before: o0,o1,o2,o3,o4,o5,o6 = 0 1 2 3 4 5 6
//After : s0,s1,s2,s3,s4,s5,s6 = 7,7,7,7,7,7,7
//PL,PR = 1D980020,1D980050 // 50-20=30 i.e. 48/8=6 or 7-1=6

                                Pivot = *PR;
/*
; mark_description "Intel(R) C++ Compiler XE for applications running on Intel(R) 64, Version 15.0.0.108 Build 20140726";
; mark_description "-FACS -O3 -FeQS_bench_r13_ICL15_0_64bit.exe -D_N_HIGH_PRIORITY";
; 19f-f4+6=177 bytes, 2/0 conditional/unconditional jumps; 53 instructions
; 'Magnetica' partitioning, mainloop rev.6-[
.B60.7::
000f4 4c 8b 6d 08    mov r13, QWORD PTR [8+rbp]
000f8 4d 3b dd        cmp r11, r13
000fb 41 89 fa       mov r10d, edi
000fe 41 89 f8       mov r8d, edi
00101 41 0f 97 c2     seta r10b
00105 41 0f 92 c0     setb r8b
00109 4d 2b d0      sub r10, r8
0010c 4d 89 d0       mov r8, r10
0010f 49 f7 da      neg r10
00112 49 c1 f8 3f   sar r8, 63
00116 49 c1 fa 3f   sar r10, 63
0011a 4c 8b 36     mov r14, QWORD PTR [rsi]
0011d 4d 3b de     cmp r11, r14
00120 73 0c       jae .B60.11
.B60.9::
00122 48 83 c6 f8    add rsi, -8
00126 4c 8b 36     mov r14, QWORD PTR [rsi]
00129 4d 3b de     cmp r11, r14
0012c 72 f4       jbe .B60.9
.B60.11::
0012e 4d 89 c1     mov r9, r8
00131 4d 89 ef     mov r15, r13
00134 49 f7 d1     not r9
00137 4d 23 f0     and r14, r8
0013a 4d 23 f9     and r15, r9
0013d 4d 03 fe     add r15, r14
00140 4c 89 7d 08   mov QWORD PTR [8+rbp], r15
00144 4d 89 c7     mov r15, r8
00147 49 83 e0 01   and r8, 1
0014b 48 83 c5 08   add rbp, 8
0014f 49 c1 e0 03   shl r8, 3
00153 4d 23 fd     and r15, r13
00156 49 2b e8     sub rbp, r8

```

```

00159 4d 23 ea    and r13, r10
0015c 4c 8b 36    mov r14, QWORD PTR [rsi]
0015f 4d 23 f1    and r14, r9
00162 4d 03 fe    add r15, r14
00165 4d 89 d6    mov r14, r10
00168 4c 89 3e    mov QWORD PTR [rsi], r15
0016b 49 f7 d6    not r14
0016e 4d 8b 0c 24  mov r9, QWORD PTR [r12]
00172 49 2b f0    sub rsi, r8
00175 4c 8b 45 00  mov r8, QWORD PTR [rbp]
00179 4d 23 ca    and r9, r10
0017c 4d 23 c6    and r8, r14
0017f 49 83 e2 01  and r10, 1
00183 4d 03 c8    add r9, r8
00186 4c 89 4d 00  mov QWORD PTR [rbp], r9
0018a 4d 8b 3c 24  mov r15, QWORD PTR [r12]
0018e 4d 23 fe    and r15, r14
00191 4d 03 ef    add r13, r15
00194 4d 89 2c 24  mov QWORD PTR [r12], r13
00198 4f 8d 24 d4  lea r12, QWORD PTR [r12+r10*8]
0019c 48 3b ee    cmp rbp, rsi
0019f 0f 82 4f ff ff  jb .B60.7
; 'Magnetica' partitioning, mainloop rev.6-]
*/
// 'Magnetica' partitioning, mainloop rev.6- [ ! TOO SLOW !
//      for (;PR < Jndx;) {
//          PR = PR + 1;
//          Temporary=*PR;
//          Stefan_Edelkamp_Armin_Weiss = Scandum_UINT64v(Pivot, Temporary);
//          Stefan_Edelkamp_Armin_Weiss_MASK = (Stefan_Edelkamp_Armin_Weiss >> 63);
//          Stefan_Edelkamp_Armin_Weiss_MASKm = ((-Stefan_Edelkamp_Armin_Weiss) >> 63);
//          for (;Pivot < *Jndx;) {
//              Jndx = Jndx - 1;
//          }
//          //if (Stefan_Edelkamp_Armin_Weiss < 0) { //]
//              M14_swapUnconditional (PR, Jndx); //]-----+
//              Jndx = Jndx - 1; //]
//              PR = PR - 1; //]
//          } //]-----/
//          *PR = ( *Jndx&Stefan_Edelkamp_Armin_Weiss_MASK ) + ( *PR&(~Stefan_Edelkamp_Armin_Weiss_MASK) );
//          *Jndx = ( Temporary&Stefan_Edelkamp_Armin_Weiss_MASK ) + ( *Jndx&(~Stefan_Edelkamp_Armin_Weiss_MASK) );
//          Jndx = Jndx - ( 1&Stefan_Edelkamp_Armin_Weiss_MASK );
//          PR = PR - ( 1&Stefan_Edelkamp_Armin_Weiss_MASK );
//          //} else if (Stefan_Edelkamp_Armin_Weiss > 0) { //]
//              M14_swapUnconditional (PL, PR); //]-----+
//              PL = PL + 1; //]
//          } //]
//          //} else { //]-----/
//              *PR = ( *PL&Stefan_Edelkamp_Armin_Weiss_MASKm ) + ( *PR&(~Stefan_Edelkamp_Armin_Weiss_MASKm) );
//              *PL = ( Temporary&Stefan_Edelkamp_Armin_Weiss_MASKm ) + ( *PL&(~Stefan_Edelkamp_Armin_Weiss_MASKm) );
//              PL = PL + ( 1&Stefan_Edelkamp_Armin_Weiss_MASKm );
//          } //]
//      }
// 'Magnetica' partitioning, mainloop rev.6-] ! TOO SLOW !

/*
; mark_description "Intel(R) C++ Compiler XE for applications running on Intel(R) 64, Version 15.0.0.108 Build 20140726";
; mark_description "-FACS -O3 -FeQS_bench_r13_ICL15.0_64bit.exe -D_N_HIGH_PRIORITY";
; dd-9b+2=68 bytes, 4/1 conditional/unconditional jumps; 23 instructions
; 'Magnetica' partitioning, mainloop rev.5a2p[
.B183.7::
0009b 48 83 c0 08    add rax, 8
0009f 4c 8b 00    mov r8, QWORD PTR [rax]
000a2 49 3b f0    cmp rsi, r8
000a5 77 26    ja .B183.14
.B183.8::
000a7 73 31    jae .B183.15
.B183.9::
000a9 48 8b 2f    mov rbp, QWORD PTR [rdi]
000ac 48 3b f5    cmp rsi, rbp
000af 73 0c    jae .B183.13
.B183.11::
000b1 48 83 c7 f8    add rdi, -8
000b5 48 8b 2f    mov rbp, QWORD PTR [rdi]
000b8 48 3b f5    cmp rsi, rbp
000bb 72 f4    jb .B183.11
.B183.13::
000bd 48 89 28    mov QWORD PTR [rax], rbp

```

```

000c0 48 83 c0 f8    add rax, -8
000c4 4c 89 07    mov QWORD PTR [rdi], r8
000c7 48 83 c7 f8    add rdi, -8
000cb eb 0d      jmp .B183.15
.B183.14::
000cd 48 8b 29    mov rbp, QWORD PTR [rcx]
000d0 4c 89 01    mov QWORD PTR [rcx], r8
000d3 48 83 c1 08    add rcx, 8
000d7 48 89 28    mov QWORD PTR [rax], rbp
.B183.15::
000da 48 3b c7    cmp rax, rdi
000dd 72 bc      jb .B183.7
; 'Magnetica' partitioning, mainloop rev.5a2p]
*/
// 'Magnetica' partitioning, mainloop rev.5a2p[
    for (;PR < Jndx;) {
        PR = PR + 1;
        if (Pivot > *PR) {
            M14_swapUnconditional (PL, PR);
            PL = PL + 1;
        } else if (Pivot < *PR) {
            for (;Pivot < *PR;) {
                Jndx = Jndx - 1;
            }
            M14_swapUnconditional (PR, Jndx);
            Jndx = Jndx - 1;
            PR = PR - 1;
        }
    }
// 'Magnetica' partitioning, mainloop rev.5a2p]
    //if (PR > Jndx) { // ]
    //    M14_swapUnconditional (&QWORDS[PR+1], &QWORDS[Jndx+1]); // ]-----+
    // } // ]
    M14_SwapConditional_ifXbY_BUGGY((uint64_t)Jndx, (uint64_t)PR, PR+1, Jndx+1); //<-----/ Kinda 'Desperate Measures' - debranchifying the mainloop as much as
possible
    Jndx = PL - 1;
    Indx = PR + 1;
    //StackPtr = StackPtr + 2; ]
    //if ((Right-Indx) > (Jndx-Left)) { ]
    //    Stack[StackPtr - 1] = Indx; ]
    //    Stack[StackPtr] = Right; ]
    //    Right = Jndx; ]
    //} else { ]
    //    Stack[StackPtr - 1] = Left; ]
    //    Stack[StackPtr] = Jndx; ]
    //    Left = Indx; ]-----+
    // } ]
    //if (Indx + M14_InsertionsortTHRESHOLD < Right) { //<-----/ Operation 'Maximum Underkill', getting rid of conditional jumps as if there is no tomorrow
        StackPtr = StackPtr + 2;
        Stack[StackPtr - 1] = Indx;
        Stack[StackPtr] = Right;
    }
    //
    Right = Jndx;
    // Stack is full, bail out and finalize with Insertionsort [
/*
    StackNotOverflow = (StackPtr + 2 <= M14_StackEntries-1);
    StackPtr = StackPtr &(-StackNotOverflow); // StackPtr should be enforced FALSE; also, ensure the next 'push' above is sanctioned - the max index of Stack[] being
'StackEntries-1'
    Right = (uint64_t*)((uint64_t)(Right) &(-StackNotOverflow)); // Left + M14_InsertionsortTHRESHOLD < Right should be enforced FALSE:
*/
    // Stack is full, bail out and finalize with Insertionsort ]
    } //switch (Right-Left) {
} //for((M14_InsertionsortTHRESHOLD < Right-Left);) { // if not 1(Right-Left then Right=Left (i.e. 0), or adjacent (i.e. 1) ... or Left>Right
DirtyBypass;
    // Caution: Inhere 'Left' could be bigger than 'Right'.
// Too slow [[[
//
// for (Indx=Left+1; Indx <= Right; Indx++) { // Inhere, we must sort the [Left..Right] micro-partition [[[
//     Jndx = Indx;
//     for (;Jndx >= 1;) {
//         if (QWORDS[Jndx-1] > QWORDS[Jndx]) M14_swapUnconditional (&QWORDS[Jndx-1], &QWORDS[Jndx]); else break;
//         Jndx = Jndx - 1;
//     }
//     } // Inhere, we must sort the [Left..Right] micro-partition ]]]
// Too slow ]]]
    } while ( (StackPtr != 0) );
// Slow [[[
/*
if (!StackNotOverflow)
    for (Indx=LeftBackup+1; Indx <= RightBackup; Indx++) {

```



```

        Jndx = Jndx;
        for (;Jndx >= LeftBackup*1;) {
            if (*(Jndx-1) > *Jndx) M14_swapUnconditional (Jndx-1, Jndx); else break;
            Jndx = Jndx - 1;
        }
    }

*/
// Slow )))
}

// This is the initial variant, without any optimizations:
/*
    // 'Magnetica' partitioning [
    Jndx = Right;
    PL = Left;
    PR = Left;
    swap (&QWORDS[(Left + Right)>>1], &QWORDS[PR]);
    Pivot = QWORDS[PR];
    for (;PR < Jndx;) {
        if (Pivot > QWORDS[PR + 1]) {
            swap (&QWORDS[PL], &QWORDS[PR + 1]);
            PL = PL + 1;
            PR = PR + 1;
        } else if (Pivot == QWORDS[PR + 1]) {
            PR = PR + 1;
        } else {
            for (;Pivot < QWORDS[Jndx];) {
                Jndx = Jndx - 1;
            }
            if (PR + 1 < Jndx) swap (&QWORDS[PR + 1], &QWORDS[Jndx]);
            Jndx = Jndx - 1;
        }
    }
    Jndx = PL - 1;
    Indx = PR + 1;
    // 'Magnetica' partitioning ]
*/

/*
// https://stackoverflow.com/questions/2061245/how-to-subtract-two-unsigned-ints-with-wrap-around-or-overflow
// /*
// Here's a little more detail of why it 'just works' when you subtract the 'smaller' from the 'larger'.
//
// A couple of things going into this..
// 1. In hardware, subtraction uses addition: The appropriate operand is simply negated before being added.
// 2. In two's complement (which pretty much everything uses), an integer is negated by inverting all the bits then adding 1.
//
// Hardware does this more efficiently than it sounds from the above description, but that's the basic algorithm for subtraction (even when values are unsigned).
//
// So, lets figure 2 - 250 using 8bit unsigned integers. In binary we have
//
// 0 0 0 0 0 0 1 0
// - 1 1 1 1 1 0 1 0
// We negate the operand being subtracted and then add. Recall that to negate we invert all the bits then add 1. After inverting the bits of the second operand we have
//
// 0 0 0 0 0 1 0 1
// Then after adding 1 we have
//
// 0 0 0 0 0 1 1 0
// Now we perform addition...
//
// 0 0 0 0 0 0 1 0
// + 0 0 0 0 0 1 1 0
//
// = 0 0 0 0 1 0 0 0 = 8, which is the result we wanted from 2 - 250
// */
//
// https://stackoverflow.com/questions/7221409/is-unsigned-integer-subtraction-defined-behavior?noredirect=1
//
// #include <stdint.h>
// #include <stdio.h>
//
// int main()
// {
//
//     uint64_t SGNdmax = 1LL<<63;
//     uint64_t UNSGNdmax = -1;
//     //[[[
//     uint8_t x = 0xff;
//     uint8_t y = x + 20;

```

```

//# int8_t res1;
//# int8_t res2;
//# //]]]
//#
//# //Well, an unsigned integer subtraction has defined behavior, also it is a tricky thing.
//# //When you subtract two unsigned integers, result is promoted to higher type int if result (lvalue) type is not specified explicitly.
//# //In the latter case, for example, int8_t result = a - b; (where a and b have int8_t type) you can obtain very weird behavior.
//# //I mean you may lose transitivity property (i.e. if a > b and b > c it is true that a > c).
//# //The loss of transitivity can destroy a tree-type data structure work.
//# //Care must be taken not to provide comparison function for sorting, searching, tree building that uses unsigned integer subtraction to deduce which key is higher or lower.
//# //See example below.
//#
//# uint8_t a = 255;
//# uint8_t b = 100;
//# uint8_t c = 150;
//#
//# //In Linux it is %llu and in Windows it is %I64u
//# printf("uint64_t SGNdmax = 1LL<<63 = %llu\n", SGNdmax);
//# printf("uint64_t UNSGNdmax = -1 = %llu\n", UNSGNdmax);
//# printf("uint64_t SGNdmax = 1LL<<63 = 0x%llx\n", SGNdmax);
//# printf("uint64_t UNSGNdmax = -1 = 0x%llx\n", UNSGNdmax);
//#
//# //[[[
//# printf("(uint8_t)x = %d\n", (uint8_t)x);
//# printf("(uint8_t)y = %d\n", (uint8_t)y);
//#
//# printf("(int8_t)x = %d\n", (int8_t)x);
//# printf("(int8_t)y = %d\n", (int8_t)y);
//#
//# res1 = (int8_t)x - y;
//# res2 = (int8_t)y - x;
//# printf("(int8_t)x - y = %d\n", res1);
//# printf("(int8_t)y - x = %d\n", res2);
//#
//# res1 = (uint8_t)x - (uint8_t)y;
//# res2 = (uint8_t)y - (uint8_t)x;
//# printf("x and y: %d and %d\n", x, y);
//# printf("(uint8_t)x - (uint8_t)y = %d\n", res1);
//# printf("(uint8_t)y - (uint8_t)x = %d\n", res2);
//# //]]]
//#
//# printf("uint8_t a = %d, b = %d, c = %d\n\n", a, b, c);
//#
//# printf("      b - a = %d\tpromotion to int type\n"
//#      "(int8_t)(b - a) = %d\n\n"
//#      "      b + a = %d\tpromotion to int type\n"
//#      "(uint8_t)(b + a) = %d\tmodular arithmetic\n"
//#      "      b + a %% %d = %d\n\n",
//#      b - a, (int8_t)(b - a),
//#      b + a, (uint8_t)(b + a),
//#      UINT8_MAX + 1,
//#      (b + a) % (UINT8_MAX + 1));
//#
//# // Modified by Kaze:
//# printf("b %s c (b - c = %d), b %s a (b - a = %d), AND c %s a (c - a = %d)\n",
//#      (int8_t)(b - c) < 0 ? "<" : ">", (int8_t)(b - c),
//#      (int8_t)(b - a) < 0 ? "<" : ">", (int8_t)(b - a),
//#      (int8_t)(c - a) < 0 ? "<" : ">", (int8_t)(c - a));
//# return 0;
//# }
//#
//# /*
//# F:\Quicksort_says_rev8++>notepad e.c
//#
//# F:\Quicksort_says_rev8++>icl e.c
//# Intel(R) C++ Compiler XE for applications running on IA-32, Version 15.0.0.108 Build 20140726
//# Copyright (C) 1985-2014 Intel Corporation. All rights reserved.
//#
//# e.c
//# Microsoft (R) Incremental Linker Version 10.00.40219.01
//# Copyright (C) Microsoft Corporation. All rights reserved.
//#
//# -out:e.exe
//# e.obj
//#
//# F:\Quicksort_says_rev8++>e
//# uint64_t SGNdmax = 1LL<<63 = 9223372036854775808
//# uint64_t UNSGNdmax = -1 = 18446744073709551615
//#

```

```

//# uint64_t 95Nmax = 1LL<<63 = 0x8000000000000000
//# uint64_t 95Nmin = -1 = 0xFFFFFFFFFFFFFFFF
//#
//# (uint8_t)x = 255
//# (uint8_t)y = 19
//# (int8_t)x = -1
//# (int8_t)y = 19
//# (int8_t)x - y = -20
//# (int8_t)y - x = 20
//# x and y: 255 and 19
//# (uint8_t)x - (uint8_t)y = -20
//# (uint8_t)y - (uint8_t)x = 20
//#
//# uint8_t a = +255, b = +100, c = +150
//#
//#      b - a = -155 promotion to int type
//# (int8_t)(b - a) = +101
//#
//#      b + a = +355 promotion to int type
//# (uint8_t)(b + a) = +99 modular arithmetic
//#      b + a % 256 = +99
//#
//# b < c (b - c = -50), b > a (b - a = 101), AND c < a (c - a = -105)
//#
//# F:\Quicksort_says_rev8++
//# */

```

```

// Test run: 2022-Mar-08:
// Laptop "Compressionette", Intel 'Kaby Lake' i5-7200U 3.1GHz max turbo, 36GB DDR4 2133MHz:

```

Performer/Keys	#1, FEW distinct	#2, MANY distinct	#3, MANYmore distinct	#4, ALL distinct	#5, ALLmore distinct	#6, ALLmax distinct
Operating System,	Windows 10,	Fedora 35,	Windows 10,	Fedora 35,	Windows 10,	Fedora 35,
Compiler, -O3	Intel v15.0 GCC 11.2.1	Intel v15.0 GCC 11.2.1	Intel v15.0 GCC 11.2.1	Intel v15.0 GCC 11.2.1	Intel v15.0 GCC 11.2.1	Intel v15.0 GCC 11.2.1
qsort	59 seconds	377 seconds	336 seconds	541 seconds	157 seconds	195 seconds
Magnetica v.13	31 seconds	30 seconds	202 seconds	196 seconds	88 seconds	85 seconds
Bentley-McIlroy	38 seconds	36 seconds	205 seconds	208 seconds	92 seconds	94 seconds
Crumsort	30 seconds	32 seconds	132 seconds	150 seconds	64 seconds	70 seconds
Best Time (bare						
bone in-place QS):	30s PARITY	132s for Crumsort	64s for Crumsort	184s for Crumsort	357s for Crumsort	N.A.

Performer/Keys	#1, FEW distinct	#2, MANY distinct	#3, MANYmore distinct	#4, ALL distinct	#5, ALLmore distinct	#6, ALLmax distinct
Operating System,	Fedora 35, GCC 11.2.1	Fedora 35, GCC 11.2.1	Fedora 35, GCC 11.2.1	Fedora 35, GCC 11.2.1	Fedora 35, GCC 11.2.1	Fedora 35, GCC 11.2.1
Compiler, -O3	instructions; IPC	instructions; IPC	instructions; IPC	instructions; IPC	instructions; IPC	instructions; IPC
qsort	3,302,993,934,921; 2.75	2,983,579,082,155; 1.75	886,263,153,476; 1.41	2,352,769,705,563; 1.38	4,527,367,288,814; 1.39	N.A.
Magnetica v.13	131,873,917,282; 1.00	658,478,895,600; 1.02	309,149,594,748; 1.08	884,297,729,161; 1.06	1,726,931,029,634; 1.08	N.A.
Bentley-McIlroy	164,915,835,204; 1.09	681,956,155,364; 1.00	322,584,009,352; 1.04	944,038,959,690; 1.02	1,719,825,062,847; 0.97	N.A.
Crumsort	312,328,497,447; 2.29	1,295,551,817,497; 2.57	603,276,911,007; 2.53	1,597,685,291,532; 2.46	3,091,001,982,856; 2.51	N.A.

```

// Speed Roster, (the base speed 1.00x is GLIBC's qsort):
// Rank #1: 2683/820= 3.27x = 32*150+ 70*192+ 376= 820 seconds for Crumsort
// Rank #2: 2683/1054= 2.54x = 30*196+ 85*250+ 493= 1054 seconds for Magnetica v.13
// Rank #3: 2683/1172= 2.28x = 36*208+ 94*281+ 553= 1172 seconds for Bentley-McIlroy
// Rank #4: 2683/2683= 1.00x = 377*541+195*534+1036= 2683 seconds for GLIBC's qsort

```

```

// Test run: 2022-Mar-09:
// Laptop "Brutalitto", AMD 'Renoir' 4800H 4.3GHz max turbo, 64GB DDR4 3200MHz:

```

Performer/Keys	#1, FEW distinct	#2, MANY distinct	#3, MANYmore distinct	#4, ALL distinct	#5, ALLmore distinct	#6, ALLmax distinct
Operating System,	Windows 10,	Windows 10,	Windows 10,	Windows 10,	Windows 10,	Windows 10,
Compiler, -O3	Intel v15.0 GCC 11.2.1	Intel v15.0 GCC 11.2.1	Intel v15.0 GCC 11.2.1	Intel v15.0 GCC 11.2.1	Intel v15.0 GCC 11.2.1	Intel v15.0 GCC 11.2.1
qsort	42 seconds	45 seconds	242 seconds	280 seconds	113 seconds	131 seconds
Magnetica v.13	22 seconds	21 seconds	135 seconds	134 seconds	60 seconds	59 seconds
Bentley-McIlroy	24 seconds	24 seconds	146 seconds	142 seconds	66 seconds	64 seconds
Crumsort	20 seconds	19 seconds	91 seconds	81 seconds	44 seconds	38 seconds
Best Time (bare						
bone in-place QS):	19s for Crumsort	81s for Crumsort	38s for Crumsort	109s for Crumsort	211s for Crumsort	479s for Crumsort

```
// Speed Roster, (the base speed 1.00x is GLIBC's qsort):
// Rank #1: 2943/937= 3.14x = 19+ 81+ 38+109+211+ 479= 937 seconds for Crumsort
// Rank #2: 2943/1462= 2.01x = 21+134+ 59+177+349+ 722= 1462 seconds for Magnetica v.13
// Rank #3: 2943/1602= 1.83x = 24+142+ 64+193+376+ 803= 1602 seconds for Bentley-McIlroy
// Rank #4: 2943/2943= 1.00x = 45+280+131+354+695+1438= 2943 seconds for GLIBC's qsort
//
// Test run: 2022-Mar-14:
// Laptop "Compressionette", Intel 'Kaby Lake' i5-7200U 3.1GHz max turbo, 36GB DDR4 2133MHz:
//
// +-----+
// | Performer/Keys | #1, FEW distinct | #2, MANY distinct | #3, MANYmore distinct | #4, ALL distinct | #5, ALLmore distinct | #6, ALLmax distinct |
// +-----+
// | Operating System, | Fedora 35, GCC 11.2.1 | Fedora 35, GCC 11.2.1 | Fedora 35, GCC 11.2.1 | Fedora 35, GCC 11.2.1 | Fedora 35, GCC 11.2.1 | Fedora 35, GCC 11.2.1 |
// | Compiler, -O3 | instructions; IPC | instructions; IPC | instructions; IPC | instructions; IPC | instructions; IPC | instructions; IPC |
// +-----+
// | qsort | 385/342 seconds | 527/234 seconds | 165/55 seconds | 516/128 seconds | 999/252 seconds | N.A. |
// | | 6,448,450,744,497; 2.82 | 4,921,980,445,033; 2.06 | 1,369,822,972,984; 1.94 | 3,250,596,357,540; 1.58 | 6,250,299,799,611; 1.59 | N.A. |
// +-----+
// | Magnetica v.14 | 29/6 seconds | 198/47 seconds | 86/22 seconds | 241/66 seconds | 472/134 seconds | N.A. |
// | | 171,452,642,615; 1.14 | 936,751,248,598; 1.17 | 443,844,553,192; 1.25 | 1,291,624,832,269; 1.28 | 2,503,942,225,085; 1.28 | N.A. |
// +-----+
// | Bentley-McIlroy | 38/13 seconds | 223/47 seconds | 100/26 seconds | 304/65 seconds | 597/139 seconds | N.A. |
// | | 246,587,334,436; 1.23 | 1,105,253,285,645; 1.25 | 507,140,963,617; 1.23 | 1,460,132,564,676; 1.22 | 2,848,210,143,410; 1.22 | N.A. |
// +-----+
// | Crumsort 1.1.5.3 | 29/4 seconds | 129/4 seconds | 61/2 seconds | 173/3 seconds | 339/7 seconds | N.A. |
// | | 351,332,884,902; 2.43 | 1,284,147,329,900; 2.80 | 603,065,127,518; 2.77 | 1,605,371,408,284; 2.64 | 3,102,283,088,926; 2.71 | N.A. |
// +-----+
//
// Legend (The time is exactly the Sort process time, first values is for unsorted, second one is for sorted):
// #1,FEW = 2,233,861,800 keys, of them distinct = 10; 178,708,944 bytes 22338618.QWORDS.bin; elements = 178,708,944/8 *100; // Keys are 100 times duplicated
// #2,MANY = 2,482,300,900 keys, of them distinct = 2,847,531; 24,823,016 bytes mbythesaurus.txt; elements = 24823016 -8+1; // BuildingBlocks are size-order+1, they are 100 times duplicated
// #3,MANYmore = 1,137,582,073 keys, of them distinct = 77,275,994; 1,137,582,080 bytes linux-5.15.25.tar; elements = 1137582080 -8+1; // BuildingBlocks are size-order+1
// #4,ALL = 2,009,333,753 keys, of them distinct = 1,912,608,132; 2,009,333,760 bytes Fedora-Workstation-Live-x86_64-35-1.2.iso; elements = 2009333760 -8+1; // BuildingBlocks are size-order+1
// #5,ALLmore = 3,803,483,825 keys, of them distinct = 3,346,259,533; 3,803,483,832 bytes Fedora-Workstation-35-1.2.aarch64.raw.xz; elements = 3803483832 -8+1; // BuildingBlocks are size-order+1
// #6,ALLmax = 7,798,235,435 keys, of them distinct = 6,770,144,405; 7,798,235,442 bytes math.stackexchange.com_en_all_2019-02.zim; elements = 7798235442 -8+1; // BuildingBlocks are size-order+1
//
// Notes:
// - Scandum's Crumsort is the FASTEST in-place sorter, known to me, hail Scandum!
// - All the runs were in "Current priority class is REALTIME_PRIORITY_CLASS" for Windows and "Current priority is -20." for Linux;
// - All the runs were executed by Core 1 (i.e. SetProcessAffinityMask(GetCurrentProcess(), 1);) in Windows, not in Linux (the task was migrating between the cores);
// To see more stats (the tables were deriving from) see 'log_i5-7200U_MAR14.txt';
// - Benchmark needs 32GB RAM, and 64GB for the 6th testset;
// - The whole package (except the 3rd, 4th, 5th and 6th datasets) is downloadable at:
// www.sammyce.com/Q5.showdown_r14.7z
// https://forum.qb64.org/index.php?topic=3518.msg141225#msg141225
// - To reproduce the roster, run on Windows or Linux:
// - BENCH_ICL32GB.BAT
// - BENCH_ICL64GB.BAT
// - sh bench_gcc32GB.sh
// - sh bench_gcc64GB.sh
// - 3rd dataset is downloadable at:
// https://cdn.kernel.org/pub/linux/kernel/v5.x/linux-5.15.25.tar.xz
// - 4th dataset is downloadable at:
// https://download.fedoraproject.org/pub/fedora/linux/releases/35/Workstation/x86_64/iso/Fedora-Workstation-Live-x86_64-35-1.2.iso
// - 5th dataset is downloadable at:
// https://download.fedoraproject.org/pub/fedora/linux/releases/35/Workstation/aarch64/images/Fedora-Workstation-35-1.2.aarch64.raw.xz
// - 6th dataset is downloadable at:
// https://download.kiwix.org/zim/stack_exchange/math.stackexchange.com_en_all_2019-02.zim
// - Managed to reduce the mainloop down to 68 bytes (from 71 in r.12), the gain comes from switching to pointers, had to do that long time ago, wanted to keep the etude ARRAY-syntax friendly. Now, r.12 uses arrays,
// whereas r.13 uses pointers. Another turning point, GCC now beats ICL in all tests.
// - In order to switch to "Benchmark mode":
// cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor
// echo performance | tee /sys/devices/system/cpu/cpu*/cpufreq/scaling_governor
//
/*
void quicksort_Bentley_McIlroy_3way_partitioning(uint64_t a[], int64_t l, int64_t r)
{
    int64_t i = l-1, j = r, p = l-1, q = r;
    int64_t k;
    uint64_t v;
    if (r <= l) return;
    swap(&a[r], &a[(r+1)>>1]); // Modified by Kaze
    v = a[r];
    for (;;)
    {
        while (a[++i] < v) ;
        while (v < a[--j]) if (j == l) break;
        if (i >= j) break;
        swap(&a[i], &a[j]);
        if (a[i] == v) { p++; swap(&a[p], &a[i]); }
    }
}
```

```

        if (v == a[j]) { q--; swap(&a[j], &a[q]); }
    }
    swap(&a[i], &a[r]);
    j = i-1; i = i+1;
    for (k = l; k < p; k++, j--) swap(&a[k], &a[j]);
    for (k = r-1; k > q; k--, i++) swap(&a[i], &a[k]);
    quicksort_Bentley_McIlroy_3way_partitioning(a, l, j);
    quicksort_Bentley_McIlroy_3way_partitioning(a, i, r);
}
*/

```

// And the Assembly code generated by GCC 11.2.1 for Magnetica_v13:

```

// 001 Quicksort_QB64_v13:
// 002 .LFB214:
// 003 .cfi_startproc
// 004     pushq    %r15
// 005     .cfi_def_cfa_offset 16
// 006     .cfi_offset 15, -16
// 007     leaq     -8(%rdi,%rsi,8), %r10
// 008     movq     %rdi, %xmm0
// 009     movq     %rdi, %rax
// 010     pushq    %r14
// 011     .cfi_def_cfa_offset 24
// 012     .cfi_offset 14, -24
// 013     movq     %r10, %xmm2
// 014     movl     $2, %r15d
// 015     pushq    %r13
// 016     .cfi_def_cfa_offset 32
// 017     .cfi_offset 13, -32
// 018     punpckldq %xmm2, %xmm0
// 019     pushq    %r12
// 020     .cfi_def_cfa_offset 40
// 021     .cfi_offset 12, -40
// 022     pushq    %rbp
// 023     .cfi_def_cfa_offset 48
// 024     .cfi_offset 6, -48
// 025     pushq    %rbx
// 026     .cfi_def_cfa_offset 56
// 027     .cfi_offset 3, -56
// 028     subq     $143920, %rsp
// 029     .cfi_def_cfa_offset 143976
// 030     movups   %xmm0, -96(%rsp)
// 031     p2align 4,,10
// 032     p2align 3
// 033 .L5115:
// 034     movq     %r10, %rcx
// 035     subq     $2, %r15
// 036     subq     %rax, %rcx
// 037     testq    %rcx, %rcx
// 038     jle     .L5096
// 039     leaq     8(%rax), %rbx
// 040 .L5113:
// 041     movq     %rcx, %rsi
// 042     movq     (%rax), %rdx
// 043     sarq     $3, %rsi
// 044     cmpeq    $55, %rcx
// 045     ja      .L5097
// 046     jmp     *.L5099(,%rsi,8)
// 047     .section .rodata
// 048     align 8
// 049     align 4
// 050 .L5099:
// 051     .quad     .L5097
// 052     .quad     .L5104
// 053     .quad     .L5103
// 054     .quad     .L5102
// 055     .quad     .L5101
// 056     .quad     .L5100
// 057     .quad     .L5098
// 058     .text
// 059     p2align 4,,10
// 060     p2align 3
// 061 .L5098:
// 062     movq     8(%rax), %rbp
// 063     movq     16(%rax), %rbx
// 064     xorl     %ecx, %ecx
// 065     movq     24(%rax), %r11
// 066     movq     32(%rax), %r10
// 067     cmpeq    %rdx, %rbp

```

```

// 068 movq 40(%rax), %r9
// 069 movq 48(%rax), %r12
// 070 setb %cl
// 071 cmpeq %rdx, %rbx
// 072 adcl $0, %ecx
// 073 cmpeq %rdx, %r11
// 074 adcl $0, %ecx
// 075 cmpeq %rdx, %r10
// 076 adcl $0, %ecx
// 077 cmpeq %rdx, %r9
// 078 adcl $0, %ecx
// 079 cmpeq %rdx, %r12
// 080 adcl $0, %ecx
// 081 xorl %r8d, %r8d
// 082 cmpeq %rdx, %rbp
// 083 movl %ecx, -100(%rsp)
// 084 setnb %r8b
// 085 xorl %ecx, %ecx
// 086 cmpeq %rbx, %rbp
// 087 seta %cl
// 088 addl %ecx, %r8d
// 089 xorl %ecx, %ecx
// 090 cmpeq %r11, %rbp
// 091 seta %cl
// 092 addl %ecx, %r8d
// 093 xorl %ecx, %ecx
// 094 cmpeq %r10, %rbp
// 095 seta %cl
// 096 addl %ecx, %r8d
// 097 xorl %ecx, %ecx
// 098 cmpeq %r9, %rbp
// 099 seta %cl
// 100 addl %ecx, %r8d
// 101 xorl %ecx, %ecx
// 102 cmpeq %r12, %rbp
// 103 seta %cl
// 104 xorl %edi, %edi
// 105 addl %ecx, %r8d
// 106 cmpeq %rdx, %rbx
// 107 setnb %dil
// 108 xorl %ecx, %ecx
// 109 cmpeq %rbx, %rbp
// 110 setbe %cl
// 111 addl %ecx, %edi
// 112 xorl %ecx, %ecx
// 113 cmpeq %r11, %rbx
// 114 seta %cl
// 115 addl %ecx, %edi
// 116 xorl %ecx, %ecx
// 117 cmpeq %r10, %rbx
// 118 seta %cl
// 119 addl %ecx, %edi
// 120 xorl %ecx, %ecx
// 121 cmpeq %r9, %rbx
// 122 seta %cl
// 123 addl %ecx, %edi
// 124 xorl %ecx, %ecx
// 125 cmpeq %r12, %rbx
// 126 seta %cl
// 127 xorl %esi, %esi
// 128 addl %ecx, %edi
// 129 cmpeq %rdx, %r11
// 130 setnb %sil
// 131 xorl %ecx, %ecx
// 132 cmpeq %r11, %rbp
// 133 setbe %cl
// 134 addl %ecx, %esi
// 135 xorl %ecx, %ecx
// 136 cmpeq %r11, %rbx
// 137 setbe %cl
// 138 addl %ecx, %esi
// 139 xorl %ecx, %ecx
// 140 cmpeq %r10, %r11
// 141 seta %cl
// 142 addl %ecx, %esi
// 143 xorl %ecx, %ecx
// 144 cmpeq %r9, %r11
// 145 seta %cl
// 146 addl %ecx, %esi

```



```

// 147 xorl    %ecx, %ecx
// 148 cmpq    %r12, %r11
// 149 seta    %cl
// 150 addl    %ecx, %esi
// 151 xorl    %ecx, %ecx
// 152 cmpq    %rdx, %r10
// 153 setnb   %cl
// 154 xorl    %r13d, %r13d
// 155 cmpq    %r10, %rbp
// 156 setbe   %r13b
// 157 addl    %r13d, %ecx
// 158 xorl    %r13d, %r13d
// 159 cmpq    %r10, %rbx
// 160 setbe   %r13b
// 161 addl    %r13d, %ecx
// 162 xorl    %r13d, %r13d
// 163 cmpq    %r10, %r11
// 164 setbe   %r13b
// 165 addl    %r13d, %ecx
// 166 xorl    %r13d, %r13d
// 167 cmpq    %r9, %r10
// 168 seta    %r13b
// 169 addl    %r13d, %ecx
// 170 xorl    %r13d, %r13d
// 171 cmpq    %r12, %r10
// 172 seta    %r13b
// 173 xorl    %r14d, %r14d
// 174 addl    %r13d, %ecx
// 175 cmpq    %rdx, %r9
// 176 setnb   %r14b
// 177 xorl    %r13d, %r13d
// 178 cmpq    %r9, %rbp
// 179 setbe   %r13b
// 180 addl    %r14d, %r13d
// 181 xorl    %r14d, %r14d
// 182 cmpq    %r9, %rbx
// 183 setbe   %r14b
// 184 addl    %r13d, %r14d
// 185 xorl    %r13d, %r13d
// 186 cmpq    %r9, %r11
// 187 setbe   %r13b
// 188 addl    %r14d, %r13d
// 189 xorl    %r14d, %r14d
// 190 cmpq    %r9, %r10
// 191 setbe   %r14b
// 192 addl    %r13d, %r14d
// 193 xorl    %r13d, %r13d
// 194 cmpq    %r12, %r9
// 195 seta    %r13b
// 196 addl    %r14d, %r13d
// 197 movslq   -108(%rsp), %r14
// 198 movq    %rdx, (%rax,%r14,8)
// 199 movslq   %r8d, %rdx
// 200 movl    -108(%rsp), %r14d
// 201 movq    %rbp, (%rax,%rdx,8)
// 202 movslq   %edi, %rdx
// 203 movq    %rbx, (%rax,%rdx,8)
// 204 movslq   %esi, %rdx
// 205 movq    %r11, (%rax,%rdx,8)
// 206 movslq   %ecx, %rdx
// 207 movq    %r10, (%rax,%rdx,8)
// 208 movslq   %r13d, %rdx
// 209 movq    %r9, (%rax,%rdx,8)
// 210 leal    (%r14,%r8), %edx
// 211 addl    %edi, %edx
// 212 addl    %esi, %edx
// 213 addl    %ecx, %edx
// 214 movl    $21, %ecx
// 215 addl    %r13d, %edx
// 216 subl    %edx, %ecx
// 217 movslq   %ecx, %rdx
// 218 movq    %r12, (%rax,%rdx,8)
// 219 .p2align 4,,10
// 220 .p2align 3
// 221 .L5096:
// 222 testq    %r15, %r15
// 223 je       .L5095
// 224 .L5121:
// 225 movq    -104(%rsp,%r15,8), %r10

```

```

// 226 movq    -112(%rsp,%r15,8), %rax
// 227 jmp     .L5115
// 228 .p2align 4,,10
// 229 .p2align 3
// 230 .L5103:
// 231 movq    8(%rax), %rdi
// 232 movq    16(%rax), %r8
// 233 xorl    %ecx, %ecx
// 234 cmpq    %rdx, %rdi
// 235 setb    %cl
// 236 cmpq    %rdx, %r8
// 237 adcl    $0, %ecx
// 238 xorl    %esi, %esi
// 239 cmpq    %rdx, %rdi
// 240 setnb    %sil
// 241 xorl    %r9d, %r9d
// 242 cmpq    %r8, %rdi
// 243 seta    %r9b
// 244 addl    %r9d, %esi
// 245 movslq   %ecx, %r9
// 246 movq    %rdx, (%rax,%r9,8)
// 247 movslq   %esi, %rdx
// 248 addl    %esi, %ecx
// 249 movq    %rdi, (%rax,%rdx,8)
// 250 movl    $3, %edx
// 251 subl    %ecx, %edx
// 252 movslq   %edx, %rdx
// 253 movq    %r8, (%rax,%rdx,8)
// 254 testq   %r15, %r15
// 255 jne     .L5121
// 256 .L5095:
// 257 addq    $143920, %rsp
// 258 .cfi_remember_state
// 259 .cfi_def_cfa_offset 56
// 260 popq    %rbx
// 261 .cfi_def_cfa_offset 48
// 262 popq    %rbp
// 263 .cfi_def_cfa_offset 40
// 264 popq    %r12
// 265 .cfi_def_cfa_offset 32
// 266 popq    %r13
// 267 .cfi_def_cfa_offset 24
// 268 popq    %r14
// 269 .cfi_def_cfa_offset 16
// 270 popq    %r15
// 271 .cfi_def_cfa_offset 8
// 272 ret
// 273 .p2align 4,,10
// 274 .p2align 3
// 275 .L5104:
// 276 .cfi_restore_state
// 277 movq    8(%rax), %rcx
// 278 cmpq    %rdx, %rcx
// 279 shbq    %rsi, %rsi
// 280 andl    $0, %esi
// 281 cmpq    %rdx, %rcx
// 282 movq    %rdx, (%rax,%rsi)
// 283 movl    $1, %edx
// 284 shbl    $0, %edx
// 285 movslq   %edx, %rdx
// 286 movq    %rcx, (%rax,%rdx,8)
// 287 jmp     .L5096
// 288 .p2align 4,,10
// 289 .p2align 3
// 290 .L5101:
// 291 movq    8(%rax), %r10
// 292 movq    16(%rax), %r9
// 293 xorl    %ecx, %ecx
// 294 movq    24(%rax), %r8
// 295 movq    32(%rax), %rdi
// 296 cmpq    %rdx, %r10
// 297 setb    %cl
// 298 cmpq    %rdx, %r9
// 299 adcl    $0, %ecx
// 300 cmpq    %rdx, %r8
// 301 adcl    $0, %ecx
// 302 cmpq    %rdx, %rdi
// 303 adcl    $0, %ecx
// 304 xorl    %esi, %esi

```

```

// 305    cmpq    %rdx, %r10
// 306    setnb   %sil
// 307    xorl    %r11d, %r11d
// 308    cmpq    %r9, %r10
// 309    seta    %r11b
// 310    addl    %r11d, %esi
// 311    xorl    %r11d, %r11d
// 312    cmpq    %r8, %r10
// 313    seta    %r11b
// 314    addl    %r11d, %esi
// 315    xorl    %r11d, %r11d
// 316    cmpq    %rdi, %r10
// 317    seta    %r11b
// 318    addl    %esi, %r11d
// 319    xorl    %esi, %esi
// 320    cmpq    %rdx, %r9
// 321    setnb   %sil
// 322    xorl    %ebx, %ebx
// 323    cmpq    %r9, %r10
// 324    setbe   %bl
// 325    addl    %ebx, %esi
// 326    xorl    %ebx, %ebx
// 327    cmpq    %r8, %r9
// 328    seta    %bl
// 329    addl    %ebx, %esi
// 330    xorl    %ebx, %ebx
// 331    cmpq    %rdi, %r9
// 332    seta    %bl
// 333    addl    %esi, %ebx
// 334    xorl    %esi, %esi
// 335    cmpq    %rdx, %r8
// 336    setnb   %sil
// 337    xorl    %ebp, %ebp
// 338    cmpq    %r8, %r10
// 339    setbe   %bpl
// 340    addl    %ebp, %esi
// 341    xorl    %ebp, %ebp
// 342    cmpq    %r8, %r9
// 343    setbe   %bpl
// 344    addl    %ebp, %esi
// 345    xorl    %ebp, %ebp
// 346    cmpq    %rdi, %r8
// 347    seta    %bpl
// 348    addl    %ebp, %esi
// 349    movslq   %ecx, %rbp
// 350    addl    %r11d, %ecx
// 351    movq     %rdx, (%rax,%rbp,8)
// 352    movslq   %r11d, %rdx
// 353    addl    %ebx, %ecx
// 354    movq     %r10, (%rax,%rdx,8)
// 355    movslq   %ebx, %rdx
// 356    addl    %esi, %ecx
// 357    movq     %r9, (%rax,%rdx,8)
// 358    movslq   %esi, %rdx
// 359    movq     %r8, (%rax,%rdx,8)
// 360    movl     $10, %edx
// 361    subl     %ecx, %edx
// 362    movslq   %edx, %rdx
// 363    movq     %rdi, (%rax,%rdx,8)
// 364    jmp      .L5096
// 365    .p2align 4,,10
// 366    .p2align 3
// 367 .L5102:
// 368    movq     8(%rax), %r10
// 369    movq     16(%rax), %r9
// 370    xorl     %ecx, %ecx
// 371    movq     24(%rax), %r8
// 372    cmpq    %rdx, %r10
// 373    setb     %cl
// 374    cmpq    %rdx, %r9
// 375    adcl     $0, %ecx
// 376    cmpq    %rdx, %r8
// 377    adcl     $0, %ecx
// 378    xorl     %edi, %edi
// 379    cmpq    %rdx, %r10
// 380    setnb    %dil
// 381    xorl     %esi, %esi
// 382    cmpq    %r9, %r10
// 383    seta     %sil

```

```

// 384 addl %esi, %edi
// 385 xorl %esi, %esi
// 386 cmpq %r8, %r10
// 387 seta %sil
// 388 addl %esi, %edi
// 389 xorl %esi, %esi
// 390 cmpq %rdx, %r9
// 391 setnb %sil
// 392 xorl %r11d, %r11d
// 393 cmpq %r9, %r10
// 394 setbe %r11b
// 395 addl %r11d, %esi
// 396 xorl %r11d, %r11d
// 397 cmpq %r8, %r9
// 398 seta %r11b
// 399 addl %r11d, %esi
// 400 movslq %ecx, %r11
// 401 addl %edi, %ecx
// 402 movq %rdx, (%rax, %r11, 8)
// 403 movslq %edi, %rdx
// 404 addl %esi, %ecx
// 405 movq %r10, (%rax, %rdx, 8)
// 406 movslq %esi, %rdx
// 407 movq %r9, (%rax, %rdx, 8)
// 408 movl $6, %edx
// 409 subl %ecx, %edx
// 410 movslq %edx, %rdx
// 411 movq %r8, (%rax, %rdx, 8)
// 412 jmp .L5096
// 413 .p2align 4,,10
// 414 .p2align 3
// 415 .L5100:
// 416 movq 8(%rax), %r12
// 417 movq 16(%rax), %rbp
// 418 xorl %ecx, %ecx
// 419 movq 24(%rax), %r11
// 420 movq 32(%rax), %r10
// 421 cmpq %rdx, %r12
// 422 movq 40(%rax), %rbx
// 423 setb %cl
// 424 cmpq %rdx, %rbp
// 425 adcl $0, %ecx
// 426 cmpq %rdx, %r11
// 427 adcl $0, %ecx
// 428 cmpq %rdx, %r10
// 429 adcl $0, %ecx
// 430 cmpq %rdx, %rbx
// 431 adcl $0, %ecx
// 432 xorl %r9d, %r9d
// 433 cmpq %rdx, %r12
// 434 setnb %r9b
// 435 xorl %esi, %esi
// 436 cmpq %rbp, %r12
// 437 seta %sil
// 438 addl %esi, %r9d
// 439 xorl %esi, %esi
// 440 cmpq %r11, %r12
// 441 seta %sil
// 442 addl %esi, %r9d
// 443 xorl %esi, %esi
// 444 cmpq %r10, %r12
// 445 seta %sil
// 446 addl %esi, %r9d
// 447 xorl %esi, %esi
// 448 cmpq %rbx, %r12
// 449 seta %sil
// 450 xorl %r8d, %r8d
// 451 addl %esi, %r9d
// 452 cmpq %rdx, %rbp
// 453 setnb %r8b
// 454 xorl %esi, %esi
// 455 cmpq %rbp, %r12
// 456 setbe %sil
// 457 addl %esi, %r8d
// 458 xorl %esi, %esi
// 459 cmpq %r11, %rbp
// 460 seta %sil
// 461 addl %esi, %r8d
// 462 xorl %esi, %esi

```

```

// 463    cmpq    %r10, %rbp
// 464    seta    %sil
// 465    addl    %esi, %r8d
// 466    xorl    %esi, %esi
// 467    cmpq    %rbx, %rbp
// 468    seta    %sil
// 469    xorl    %edi, %edi
// 470    addl    %esi, %r8d
// 471    cmpq    %rdx, %r11
// 472    setnb  %dil
// 473    xorl    %esi, %esi
// 474    cmpq    %r11, %r12
// 475    setbe  %sil
// 476    addl    %esi, %edi
// 477    xorl    %esi, %esi
// 478    cmpq    %r11, %rbp
// 479    setbe  %sil
// 480    addl    %esi, %edi
// 481    xorl    %esi, %esi
// 482    cmpq    %r10, %r11
// 483    seta    %sil
// 484    addl    %esi, %edi
// 485    xorl    %esi, %esi
// 486    cmpq    %rbx, %r11
// 487    seta    %sil
// 488    addl    %esi, %edi
// 489    xorl    %esi, %esi
// 490    cmpq    %rdx, %r10
// 491    setnb  %sil
// 492    xorl    %r13d, %r13d
// 493    cmpq    %r10, %r12
// 494    setbe  %r13b
// 495    addl    %r13d, %esi
// 496    xorl    %r13d, %r13d
// 497    cmpq    %r10, %rbp
// 498    setbe  %r13b
// 499    addl    %r13d, %esi
// 500    xorl    %r13d, %r13d
// 501    cmpq    %r10, %r11
// 502    setbe  %r13b
// 503    addl    %r13d, %esi
// 504    xorl    %r13d, %r13d
// 505    cmpq    %rbx, %r10
// 506    seta    %r13b
// 507    addl    %r13d, %esi
// 508    movslq  %ecx, %r13
// 509    movq    %rdx, (%rax,%r13,8)
// 510    movslq  %r9d, %rdx
// 511    movq    %r12, (%rax,%rdx,8)
// 512    movslq  %r8d, %rdx
// 513    movq    %rbp, (%rax,%rdx,8)
// 514    movslq  %edi, %rdx
// 515    movq    %r11, (%rax,%rdx,8)
// 516    movslq  %esi, %rdx
// 517    movq    %r10, (%rax,%rdx,8)
// 518    leal    (%rcx,%r9), %edx
// 519    movl    $15, %ecx
// 520    addl    %r8d, %edx
// 521    addl    %edi, %edx
// 522    addl    %esi, %edx
// 523    subl    %edx, %ecx
// 524    movslq  %ecx, %rdx
// 525    movq    %rbx, (%rax,%rdx,8)
// 526    jmp     .L5096
// 527 .L5097:
// 528    sarq     $2, %rsi
// 529    leaq     (%rax,%rsi,8), %rcx
// 530    movq     (%rcx), %r8
// 531    movq     %rdx, (%rcx)
// 532    movq     %r8, (%rax)
// 533    cmpq     %rax, %r10
// 534    jbe     .L5105
// 535    movq     %rax, %rdx
// 536    movq     %rax, %r9
// 537    movq     %r10, %rcx
// 538    jmp     .L5111
// 539    .p2align 4,,10
// 540    .p2align 3
// 541 .L5122:

```

```

// 542    movq    (%r9), %rdi
// 543    movq    %rsi, (%r9)
// 544    leaq    16(%rdx), %rsi
// 545    addq    $8, %r9
// 546    movq    %rdi, 8(%rdx)
// 547    movq    %r11, %rdx
// 548 .L5107:
// 549    cmpq    %rdx, %rcx
// 550    jbe     .L5112
// 551 .L5111:
// 552    movq    8(%rdx), %rsi
// 553    leaq    8(%rdx), %r11
// 554    cmpq    %r8, %rsi
// 555    jb     .L5122
// 556    ja     .L5120
// 557    leaq    16(%rdx), %rsi
// 558    movq    %r11, %rdx
// 559    cmpq    %rdx, %rcx
// 560    ja     .L5111
// 561 .L5112:
// 562    movq    8(%rdx), %rdi
// 563    movq    %rsi, %xmm0
// 564    movq    %rcx, %rsi
// 565    movq    %r10, %xmm1
// 566    subq    %rdx, %rsi
// 567    leaq    -8(%r9), %r10
// 568    punpcklqdq %xmm1, %xmm0
// 569    addq    $2, %r15
// 570    movq    %rdi, %r8
// 571    sarq    $63, %rsi
// 572    subq    8(%rcx), %r8
// 573    movups   %xmm0, -112(%rsp, %r15, 8)
// 574    andq    %r8, %rsi
// 575    subq    %rsi, %rdi
// 576    movq    %rdi, 8(%rdx)
// 577    addq    %rsi, 8(%rcx)
// 578    movq    %r10, %rcx
// 579    subq    %rax, %rcx
// 580    testq   %rcx, %rcx
// 581    jg      .L5113
// 582    jmp     .L5096
// 583    .p2align 4,,10
// 584    .p2align 3
// 585 .L5110:
// 586    subq    $8, %rcx
// 587 .L5120:
// 588    movq    (%rcx), %rdi
// 589    cmpq    %r8, %rdi
// 590    ja     .L5110
// 591    movq    %rdi, 8(%rdx)
// 592    subq    $8, %rcx
// 593    movq    %rsi, 8(%rcx)
// 594    movq    %r11, %rsi
// 595    jmp     .L5107
// 596    .p2align 4,,10
// 597    .p2align 3
// 598 .L5105:
// 599    movq    %rbx, %rsi
// 600    movq    %rax, %rdx
// 601    movq    %rax, %r9
// 602    movq    %r10, %rcx
// 603    jmp     .L5112
// 604    .cfi_endproc

```

// And the Assembly code generated by GCC 11.2.1 for Magnetica_v14:

```

// 001 Quicksort_QB64_v14:
// 002 .LFB212:
// 003    .cfi_startproc
// 004    pushq    %r15
// 005    .cfi_def_cfa_offset 16
// 006    .cfi_offset 15, -16
// 007    leaq    -8(%rdi,%rsi,8), %rcx
// 008    movq    %rdi, %xmm0
// 009    movl    $2, %r15d
// 010    pushq    %r14
// 011    .cfi_def_cfa_offset 24
// 012    .cfi_offset 14, -24
// 013    movq    %rcx, %xmm2
// 014    pushq    %r13

```



```

// 015 .cfi_def_cfa_offset 32
// 016 .cfi_offset 13, -32
// 017 punpcklqdq %xmm2, %xmm0
// 018 pushq %r12
// 019 .cfi_def_cfa_offset 40
// 020 .cfi_offset 12, -40
// 021 movq %rdi, %r12
// 022 pushq %rbp
// 023 .cfi_def_cfa_offset 48
// 024 .cfi_offset 6, -48
// 025 pushq %rbx
// 026 .cfi_def_cfa_offset 56
// 027 .cfi_offset 3, -56
// 028 subq $143936, %rsp
// 029 .cfi_def_cfa_offset 143992
// 030 movups %xmm0, -80(%rsp)
// 031 .p2align 4,,10
// 032 .p2align 3
// 033 .L5166:
// 034 movq %rcx, %rdx
// 035 subq $2, %r15
// 036 subq %r12, %rdx
// 037 testq %rdx, %rdx
// 038 jle .L5127
// 039 leaq 8(%r12), %r10
// 040 leaq 16(%r12), %r11
// 041 .L5164:
// 042 movq %rdx, %rsi
// 043 movq (%r12), %rax
// 044 sarq $3, %rsi
// 045 cmpq $111, %rdx
// 046 ja .L5128
// 047 jmp *L5130(%rsi,8)
// 048 .section .rodata
// 049 .align 8
// 050 .align 4
// 051 .L5130:
// 052 .quad .L5128
// 053 .quad .L5142
// 054 .quad .L5141
// 055 .quad .L5140
// 056 .quad .L5139
// 057 .quad .L5138
// 058 .quad .L5137
// 059 .quad .L5136
// 060 .quad .L5135
// 061 .quad .L5134
// 062 .quad .L5133
// 063 .quad .L5132
// 064 .quad .L5131
// 065 .quad .L5129
// 066 .text
// 067 .p2align 4,,10
// 068 .p2align 3
// 069 .L5129:
// 070 movq 8(%r12), %rcx
// 071 movq %rax, %rdx
// 072 movq 72(%r12), %rsi
// 073 movq 88(%r12), %r13
// 074 cmpq %rax, %rcx
// 075 cmovbe %rcx, %rdx
// 076 cmovb %rax, %rcx
// 077 movq 16(%r12), %rax
// 078 movq %rcx, %rbp
// 079 cmpq %rcx, %rax
// 080 cmovbe %rax, %rbp
// 081 cmovb %rcx, %rax
// 082 movq 24(%r12), %rcx
// 083 movq %rax, %rbx
// 084 cmpq %rax, %rcx
// 085 cmovbe %rcx, %rbx
// 086 cmovb %rax, %rcx
// 087 movq 32(%r12), %rax
// 088 movq %rcx, %r11
// 089 cmpq %rcx, %rax
// 090 movq %rbx, %xmm0
// 091 cmovbe %rax, %r11
// 092 cmovb %rcx, %rax
// 093 movq 40(%r12), %rcx

```

```

// 094    movq    %rax, %r10
// 095    cmpq    %rax, %rcx
// 096    movq    %r11, %xmm3
// 097    cmovbe  %rcx, %r10
// 098    cmovb  %rax, %rcx
// 099    movq    48(%r12), %rax
// 100    punpckldq %xmm3, %xmm0
// 101    movups  %xmm0, 16(%r12)
// 102    cmpq    %rcx, %rax
// 103    movq    %rcx, %r9
// 104    movq    %r10, %xmm0
// 105    cmovbe  %rax, %r9
// 106    cmovb  %rcx, %rax
// 107    movq    56(%r12), %rcx
// 108    movq    %rax, %r8
// 109    cmpq    %rax, %rcx
// 110    movq    %r9, %xmm4
// 111    cmovbe  %rcx, %r8
// 112    cmovb  %rax, %rcx
// 113    movq    64(%r12), %rax
// 114    punpckldq %xmm4, %xmm0
// 115    movups  %xmm0, 32(%r12)
// 116    cmpq    %rcx, %rax
// 117    movq    %rcx, %rdi
// 118    movq    %r8, %xmm0
// 119    cmovbe  %rax, %rdi
// 120    cmovb  %rcx, %rax
// 121    cmpq    %rax, %rsi
// 122    movq    %rax, %rcx
// 123    movq    %rdi, %xmm5
// 124    cmovbe  %rsi, %rcx
// 125    cmovb  %rax, %rsi
// 126    movq    80(%r12), %rax
// 127    punpckldq %xmm5, %xmm0
// 128    movups  %xmm0, 48(%r12)
// 129    cmpq    %rsi, %rax
// 130    movq    %rsi, %r14
// 131    movq    %rcx, %xmm0
// 132    cmovbe  %rax, %r14
// 133    cmovb  %rsi, %rax
// 134    cmpq    %rax, %r13
// 135    movq    %rax, %rsi
// 136    movq    %r14, -96(%rsp)
// 137    cmovbe  %r13, %rsi
// 138    cmovb  %rax, %r13
// 139    movq    96(%r12), %rax
// 140    movq    %r13, %r14
// 141    cmpq    %r13, %rax
// 142    movhps  -96(%rsp), %xmm0
// 143    cmovbe  %rax, %r14
// 144    cmovb  %r13, %rax
// 145    movq    104(%r12), %r13
// 146    movups  %xmm0, 64(%r12)
// 147    movq    %rsi, %xmm0
// 148    movq    %r14, -104(%rsp)
// 149    cmpq    %rax, %r13
// 150    movq    %rax, %r14
// 151    cmovbe  %r13, %r14
// 152    cmovb  %rax, %r13
// 153    movq    %r14, -112(%rsp)
// 154    movhps  -104(%rsp), %xmm0
// 155    movq    %r13, 104(%r12)
// 156    movups  %xmm0, 80(%r12)
// 157 .L5143:
// 158    cmpq    %rbp, %rdx
// 159    movq    %rbp, %rax
// 160    movq    %r11, %r14
// 161    movq    %r10, %r13
// 162    cmovnb  %rdx, %rbp
// 163    cmovbe  %rdx, %rax
// 164    movq    %rbx, %rdx
// 165    cmpq    %rbx, %rbp
// 166    cmovnb  %rbp, %rbx
// 167    cmovbe  %rbp, %rdx
// 168    movq    %r9, %rbp
// 169    cmpq    %r11, %rbx
// 170    cmovnb  %rbx, %r11
// 171    cmovbe  %rbx, %r14
// 172    movq    %r8, %rbx

```

```

// 173    cmpq    %r10, %r11
// 174    movq    %r14, %xmm0
// 175    cmovnb  %r11, %r10
// 176    cmovbe  %r11, %r13
// 177    movq    %rdi, %r11
// 178    cmpq    %r9, %r10
// 179    movq    %r13, %xmm3
// 180    cmovnb  %r10, %r9
// 181    cmovbe  %r10, %rbp
// 182    punpckldq %xmm3, %xmm0
// 183    movq    %rcx, %r10
// 184    movups   %xmm0, 16(%r12)
// 185    cmpq    %r8, %r9
// 186    movq    %rbp, %xmm0
// 187    cmovnb  %r9, %r8
// 188    cmovbe  %r9, %rax
// 189    cmpq    %rdi, %r8
// 190    movq    %rax, %xmm4
// 191    cmovnb  %r8, %rdi
// 192    cmovbe  %r8, %r11
// 193    punpckldq %xmm4, %xmm0
// 194    movq    %rsi, %r8
// 195    movups   %xmm0, 32(%r12)
// 196    cmpq    %rcx, %rdi
// 197    movq    %r11, %xmm0
// 198    cmovnb  %rdi, %rcx
// 199    cmovbe  %rdi, %r10
// 200    movq    -96(%rsp), %rdi
// 201    cmpq    %rdi, %rcx
// 202    movq    %rdi, %r9
// 203    movq    %r10, %xmm5
// 204    cmovbe  %rcx, %r9
// 205    cmovb   %rdi, %rcx
// 206    punpckldq %xmm5, %xmm0
// 207    movups   %xmm0, 48(%r12)
// 208    cmpq    %rsi, %rcx
// 209    movq    %r9, %xmm0
// 210    cmovbe  %rcx, %r8
// 211    cmovb   %rsi, %rcx
// 212    movq    -104(%rsp), %rsi
// 213    cmpq    %rsi, %rcx
// 214    movq    %rsi, %rdi
// 215    movq    %r8, %xmm6
// 216    cmovbe  %rcx, %rdi
// 217    cmovb   %rsi, %rcx
// 218    movq    -112(%rsp), %rsi
// 219    punpckldq %xmm6, %xmm0
// 220    movups   %xmm0, 64(%r12)
// 221    cmpq    %rsi, %rcx
// 222    movq    %rdi, %xmm0
// 223    cmovbe  %rcx, %rsi
// 224    cmovb   -112(%rsp), %rcx
// 225    movq    %rsi, %xmm7
// 226    movq    %rcx, 96(%r12)
// 227    punpckldq %xmm7, %xmm0
// 228    movups   %xmm0, 80(%r12)
// 229 .L5144:
// 230    cmpq    %rdx, %rax
// 231    movq    %rdx, %rcx
// 232    cmovbe  %rax, %rcx
// 233    cmovb   %rdx, %rax
// 234    movq    %r14, %rdx
// 235    cmpq    %r14, %rax
// 236    cmovbe  %rax, %rdx
// 237    cmovb   %r14, %rax
// 238    movq    %r13, %r14
// 239    cmpq    %r13, %rax
// 240    cmovbe  %rax, %r14
// 241    cmovb   %r13, %rax
// 242    movq    %rbp, %r13
// 243    cmpq    %rbp, %rax
// 244    movq    %r14, %xmm0
// 245    cmovbe  %rax, %r13
// 246    cmovb   %rbp, %rax
// 247    movq    %rbx, %rbp
// 248    cmpq    %rbx, %rax
// 249    movq    %r13, %xmm3
// 250    cmovbe  %rax, %rbp
// 251    cmovb   %rbx, %rax

```

```

// 252 punpcklqdq %xmm3, %xmm0
// 253 movq %r11, %rbx
// 254 movups %xmm0, 16(%r12)
// 255 cmpq %r11, %rax
// 256 movq %rbp, %xmm0
// 257 cmovbe %rax, %rbx
// 258 cmovb %r11, %rax
// 259 movq %r10, %r11
// 260 cmpq %r10, %rax
// 261 movq %rbx, %xmm4
// 262 cmovbe %rax, %r11
// 263 cmovb %r10, %rax
// 264 punpcklqdq %xmm4, %xmm0
// 265 movq %r9, %r10
// 266 movups %xmm0, 32(%r12)
// 267 cmpq %r9, %rax
// 268 movq %r11, %xmm0
// 269 cmovbe %rax, %r10
// 270 cmovb %r9, %rax
// 271 movq %r8, %r9
// 272 cmpq %r8, %rax
// 273 movq %r10, %xmm2
// 274 cmovbe %rax, %r9
// 275 cmovb %r8, %rax
// 276 punpcklqdq %xmm2, %xmm0
// 277 movq %rdi, %r8
// 278 movups %xmm0, 48(%r12)
// 279 cmpq %rdi, %rax
// 280 movq %r9, %xmm0
// 281 cmovbe %rax, %r8
// 282 cmovb %rdi, %rax
// 283 movq %rsi, %rdi
// 284 cmpq %rsi, %rax
// 285 movq %r8, %xmm1
// 286 cmovbe %rax, %rdi
// 287 cmovb %rsi, %rax
// 288 punpcklqdq %xmm1, %xmm0
// 289 movups %xmm0, 64(%r12)
// 290 movq %rax, 88(%r12)
// 291 .L5145:
// 292 cmpq %rdx, %rcx
// 293 movq %rdx, %rax
// 294 movq %r14, %rsi
// 295 cmovnb %rcx, %rdx
// 296 cmovbe %rcx, %rax
// 297 movq %r11, %rcx
// 298 cmpq %r14, %rdx
// 299 cmovbe %rdx, %rsi
// 300 cmovb %r14, %rdx
// 301 movq %r13, %r14
// 302 cmpq %r13, %rdx
// 303 cmovbe %rdx, %r14
// 304 cmovb %r13, %rdx
// 305 movq %rbp, %r13
// 306 cmpq %rbp, %rdx
// 307 movq %r14, %xmm0
// 308 cmovbe %rdx, %r13
// 309 cmovb %rbp, %rdx
// 310 movq %rbx, %rbp
// 311 cmpq %rbx, %rdx
// 312 movq %r13, %xmm1
// 313 cmovbe %rdx, %rbp
// 314 cmovb %rbx, %rdx
// 315 punpcklqdq %xmm1, %xmm0
// 316 movups %xmm0, 16(%r12)
// 317 cmpq %r11, %rdx
// 318 movq %rbp, %xmm0
// 319 cmovbe %rdx, %rcx
// 320 cmovb %r11, %rdx
// 321 movq %r10, %r11
// 322 cmpq %r10, %rdx
// 323 movq %rcx, %xmm5
// 324 cmovbe %rdx, %r11
// 325 cmovb %r10, %rdx
// 326 punpcklqdq %xmm5, %xmm0
// 327 movq %r9, %r10
// 328 movups %xmm0, 32(%r12)
// 329 cmpq %r9, %rdx
// 330 movq %r11, %xmm0

```

```

// 331 cmovbe %rdx, %r10
// 332 cmovb %r9, %rdx
// 333 movq %r8, %r9
// 334 cmpq %r8, %rdx
// 335 movq %r10, %xmm6
// 336 cmovbe %rdx, %r9
// 337 cmovb %r8, %rdx
// 338 punpcklqdq %xmm6, %xmm0
// 339 movq %rdi, %r8
// 340 movups %xmm0, 48(%r12)
// 341 cmpq %rdi, %rdx
// 342 movq %r9, %xmm0
// 343 cmovbe %rdx, %r8
// 344 cmovb %rdi, %rdx
// 345 movq %r8, %xmm7
// 346 movq %rdx, 80(%r12)
// 347 punpcklqdq %xmm7, %xmm0
// 348 movups %xmm0, 64(%r12)
// 349 .L5146:
// 350 cmpq %rsi, %rax
// 351 movq %rsi, %rdx
// 352 movq %r14, %rbx
// 353 movq %r13, %rdi
// 354 cmovbe %rax, %rdx
// 355 cmovb %rsi, %rax
// 356 movq %r11, %rsi
// 357 cmpq %r14, %rax
// 358 cmovbe %rax, %rbx
// 359 cmovb %r14, %rax
// 360 cmpq %r13, %rax
// 361 cmovbe %rax, %rdi
// 362 cmovb %r13, %rax
// 363 movq %rbp, %r13
// 364 cmpq %rbp, %rax
// 365 movq %rdi, %xmm0
// 366 cmovbe %rax, %r13
// 367 cmovb %rbp, %rax
// 368 movq %rcx, %rbp
// 369 cmpq %rcx, %rax
// 370 movq %r13, %xmm5
// 371 cmovbe %rax, %rbp
// 372 cmovb %rcx, %rax
// 373 punpcklqdq %xmm5, %xmm0
// 374 movq %r9, %rcx
// 375 movups %xmm0, 16(%r12)
// 376 cmpq %r11, %rax
// 377 movq %rbp, %xmm0
// 378 cmovbe %rax, %rsi
// 379 cmovb %r11, %rax
// 380 movq %r10, %r11
// 381 cmpq %r10, %rax
// 382 movq %rsi, %xmm3
// 383 cmovbe %rax, %r11
// 384 cmovb %r10, %rax
// 385 punpcklqdq %xmm3, %xmm0
// 386 movups %xmm0, 32(%r12)
// 387 cmpq %r9, %rax
// 388 movq %r11, %xmm0
// 389 cmovbe %rax, %rcx
// 390 cmovb %r9, %rax
// 391 movq %r8, %r9
// 392 cmpq %r8, %rax
// 393 movq %rcx, %xmm2
// 394 cmovbe %rax, %r9
// 395 cmovb %r8, %rax
// 396 punpcklqdq %xmm2, %xmm0
// 397 movups %xmm0, 48(%r12)
// 398 movq %rax, 72(%r12)
// 399 .L5147:
// 400 cmpq %rbx, %rdx
// 401 movq %rbx, %rax
// 402 movq %r13, %r10
// 403 movq %rsi, %r8
// 404 cmovnb %rdx, %rbx
// 405 cmovbe %rdx, %rax
// 406 movq %rdi, %rdx
// 407 cmpq %rdi, %rbx
// 408 cmovnb %rbx, %rdi
// 409 cmovbe %rbx, %rdx

```

```

// 410 movq %rbp, %rbx
// 411 cmpq %r13, %rdi
// 412 cmovbe %rdi, %r10
// 413 cmovb %r13, %rdi
// 414 cmpq %rbp, %rdi
// 415 movq %r10, %xmm0
// 416 cmovbe %rdi, %rbx
// 417 cmovb %rbp, %rdi
// 418 cmpq %rsi, %rdi
// 419 movq %rbx, %xmm6
// 420 cmovnb %rdi, %rsi
// 421 cmovbe %rdi, %r8
// 422 punpcklqdq %xmm6, %xmm0
// 423 movq %r11, %rdi
// 424 movups %xmm0, 16(%r12)
// 425 cmpq %r11, %rsi
// 426 movq %r8, %xmm0
// 427 cmovnb %rsi, %r11
// 428 cmovbe %rsi, %rdi
// 429 movq %rcx, %rsi
// 430 cmpq %rcx, %r11
// 431 movq %rdi, %xmm7
// 432 cmovbe %r11, %rsi
// 433 cmovb %rcx, %r11
// 434 punpcklqdq %xmm7, %xmm0
// 435 movq %r9, %rcx
// 436 movups %xmm0, 32(%r12)
// 437 cmpq %r9, %r11
// 438 movq %rsi, %xmm0
// 439 cmovbe %r11, %rcx
// 440 cmovb %r9, %r11
// 441 movq %rcx, %xmm4
// 442 movq %r11, 64(%r12)
// 443 punpcklqdq %xmm4, %xmm0
// 444 movups %xmm0, 48(%r12)
// 445 .L5148:
// 446 cmpq %rdx, %rax
// 447 movq %rdx, %r9
// 448 movq %r10, %r11
// 449 cmovbe %rax, %r9
// 450 cmovb %rdx, %rax
// 451 movq %r8, %rdx
// 452 cmpq %r10, %rax
// 453 cmovbe %rax, %r11
// 454 cmovb %r10, %rax
// 455 movq %rbx, %r10
// 456 cmpq %rbx, %rax
// 457 cmovbe %rax, %r10
// 458 cmovb %rbx, %rax
// 459 cmpq %r8, %rax
// 460 movq %r10, %xmm0
// 461 cmovbe %rax, %rdx
// 462 cmovb %r8, %rax
// 463 movq %rdi, %r8
// 464 cmpq %rdi, %rax
// 465 movq %rdx, %xmm6
// 466 cmovbe %rax, %r8
// 467 cmovb %rdi, %rax
// 468 punpcklqdq %xmm6, %xmm0
// 469 movq %rsi, %rdi
// 470 movups %xmm0, 16(%r12)
// 471 cmpq %rsi, %rax
// 472 movq %r8, %xmm0
// 473 cmovbe %rax, %rdi
// 474 cmovb %rsi, %rax
// 475 movq %rcx, %rsi
// 476 cmpq %rcx, %rax
// 477 movq %rdi, %xmm7
// 478 cmovbe %rax, %rsi
// 479 cmovb %rcx, %rax
// 480 punpcklqdq %xmm7, %xmm0
// 481 movups %xmm0, 32(%r12)
// 482 movq %rax, 56(%r12)
// 483 .L5149:
// 484 cmpq %r11, %r9
// 485 movq %r11, %rax
// 486 movq %r10, %rcx
// 487 cmovbe %r9, %rax
// 488 cmovb %r11, %r9

```



```

// 489    cmpq    %r10, %r9
// 490    cmovbe %r9, %rcx
// 491    cmovb  %r10, %r9
// 492    movq   %rdx, %r10
// 493    cmpq   %rdx, %r9
// 494    cmovnb %r9, %rdx
// 495    cmovbe %r9, %r10
// 496    movq   %r8, %r9
// 497    cmpq   %r8, %rdx
// 498    movq   %r10, %xmm0
// 499    cmovbe %rdx, %r9
// 500    cmovb  %r8, %rdx
// 501    movq   %rdi, %r8
// 502    cmpq   %rdi, %rdx
// 503    movq   %r9, %xmm4
// 504    cmovbe %rdx, %r8
// 505    cmovb  %rdi, %rdx
// 506    punpcklqdq %xmm4, %xmm0
// 507    movq   %rsi, %rdi
// 508    movups %xmm0, 16(%r12)
// 509    cmpq   %rsi, %rdx
// 510    movq   %r8, %xmm0
// 511    cmovbe %rdx, %rdi
// 512    cmovb  %rsi, %rdx
// 513    movq   %rdi, %xmm5
// 514    movq   %rdx, 48(%r12)
// 515    punpcklqdq %xmm5, %xmm0
// 516    movups %xmm0, 32(%r12)
// 517 .L5150:
// 518    cmpq   %rcx, %rax
// 519    movq   %rcx, %rdx
// 520    movq   %r9, %rsi
// 521    cmovbe %rax, %rdx
// 522    cmovb  %rcx, %rax
// 523    movq   %r10, %rcx
// 524    cmpq   %r10, %rax
// 525    cmovbe %rax, %rcx
// 526    cmovb  %r10, %rax
// 527    cmpq   %r9, %rax
// 528    cmovbe %rax, %rsi
// 529    cmovb  %r9, %rax
// 530    movq   %r8, %r9
// 531    cmpq   %r8, %rax
// 532    movq   %rsi, %xmm0
// 533    cmovbe %rax, %r9
// 534    cmovb  %r8, %rax
// 535    movq   %rdi, %r8
// 536    cmpq   %rdi, %rax
// 537    movq   %r9, %xmm3
// 538    cmovbe %rax, %r8
// 539    cmovb  %rdi, %rax
// 540    punpcklqdq %xmm3, %xmm0
// 541    movups %xmm0, 16(%r12)
// 542    movq   %rax, 40(%r12)
// 543 .L5151:
// 544    cmpq   %rcx, %rdx
// 545    movq   %rcx, %rax
// 546    movq   %r9, %rdi
// 547    cmovbe %rdx, %rax
// 548    cmovb  %rcx, %rdx
// 549    movq   %rsi, %rcx
// 550    cmpq   %rsi, %rdx
// 551    cmovbe %rdx, %rcx
// 552    cmovb  %rsi, %rdx
// 553    movq   %r8, %rsi
// 554    cmpq   %r9, %rdx
// 555    cmovbe %rdx, %rdi
// 556    cmovb  %r9, %rdx
// 557    cmpq   %r8, %rdx
// 558    movq   %rdi, %xmm0
// 559    cmovbe %rdx, %rsi
// 560    cmovb  %r8, %rdx
// 561    movq   %rsi, %xmm2
// 562    movq   %rdx, 32(%r12)
// 563    punpcklqdq %xmm2, %xmm0
// 564    movups %xmm0, 16(%r12)
// 565 .L5152:
// 566    cmpq   %rcx, %rax
// 567    movq   %rcx, %rdx

```

```

// 568 cmovbe %rax, %rdx
// 569 cmovb %rcx, %rax
// 570 movq %rdi, %rcx
// 571 cmpq %rdi, %rax
// 572 cmovbe %rax, %rcx
// 573 cmovb %rdi, %rax
// 574 movq %rsi, %rdi
// 575 cmpq %rsi, %rax
// 576 cmovbe %rax, %rdi
// 577 cmovb %rsi, %rax
// 578 movq %rax, 24(%r12)
// 579 .L5153:
// 580 cmpq %rcx, %rdx
// 581 movq %rcx, %rax
// 582 cmovbe %rdx, %rax
// 583 cmovb %rcx, %rdx
// 584 movq %rdi, %rcx
// 585 cmpq %rdi, %rdx
// 586 cmovbe %rdx, %rcx
// 587 cmovb %rdi, %rdx
// 588 movq %rdx, 16(%r12)
// 589 .L5154:
// 590 cmpq %rcx, %rax
// 591 movq %rcx, %rbx
// 592 cmovbe %rax, %rbx
// 593 cmovb %rcx, %rax
// 594 movq %rbx, %xmm0
// 595 movq %rax, %xmm1
// 596 punpckldq %xmm1, %xmm0
// 597 movups %xmm0, (%r12)
// 598 .L5127:
// 599 testq %r15, %r15
// 600 je .L5126
// 601 movq -88(%rsp,%r15,8), %rcx
// 602 movq -96(%rsp,%r15,8), %r12
// 603 jmp .L5166
// 604 .p2align 4,,10
// 605 .p2align 3
// 606 .L5131:
// 607 movq 96(%r12), %rbx
// 608 movq 80(%r12), %rsi
// 609 movq %rax, %rdx
// 610 movq 64(%r12), %rcx
// 611 movq 56(%r12), %rdi
// 612 movq 48(%r12), %r8
// 613 movq 40(%r12), %r9
// 614 movq %rbx, -112(%rsp)
// 615 movq 88(%r12), %rbx
// 616 movq 32(%r12), %r10
// 617 movq 24(%r12), %r11
// 618 movq 8(%r12), %rbp
// 619 movq %rbx, -104(%rsp)
// 620 movq 72(%r12), %rbx
// 621 movq %rbx, -96(%rsp)
// 622 movq 16(%r12), %rbx
// 623 jmp .L5143
// 624 .p2align 4,,10
// 625 .p2align 3
// 626 .L5132:
// 627 movq 88(%r12), %rsi
// 628 movq 80(%r12), %rdi
// 629 movq 72(%r12), %r8
// 630 movq 64(%r12), %r9
// 631 movq 56(%r12), %r10
// 632 movq 48(%r12), %r11
// 633 movq 40(%r12), %rbx
// 634 movq 32(%r12), %rbp
// 635 movq 24(%r12), %r13
// 636 movq 16(%r12), %r14
// 637 movq 8(%r12), %rdx
// 638 jmp .L5144
// 639 .p2align 4,,10
// 640 .p2align 3
// 641 .L5133:
// 642 movq 80(%r12), %rdi
// 643 movq 72(%r12), %r8
// 644 movq %rax, %rcx
// 645 movq 64(%r12), %r9
// 646 movq 56(%r12), %r10

```

```

// 647    movq    48(%r12), %r11
// 648    movq    40(%r12), %rbx
// 649    movq    32(%r12), %rbp
// 650    movq    24(%r12), %r13
// 651    movq    16(%r12), %r14
// 652    movq    8(%r12), %rdx
// 653    jmp     .L5145
// 654    .p2align 4,,10
// 655    .p2align 3
// 656 .L5134:
// 657    movq    72(%r12), %r8
// 658    movq    64(%r12), %r9
// 659    movq    56(%r12), %r10
// 660    movq    48(%r12), %r11
// 661    movq    40(%r12), %rcx
// 662    movq    32(%r12), %rbp
// 663    movq    24(%r12), %r13
// 664    movq    16(%r12), %r14
// 665    movq    8(%r12), %rsi
// 666    jmp     .L5146
// 667    .p2align 4,,10
// 668    .p2align 3
// 669 .L5135:
// 670    movq    64(%r12), %r9
// 671    movq    56(%r12), %rcx
// 672    movq    %rax, %rdx
// 673    movq    48(%r12), %r11
// 674    movq    40(%r12), %rsi
// 675    movq    32(%r12), %rbp
// 676    movq    24(%r12), %r13
// 677    movq    16(%r12), %rdi
// 678    movq    8(%r12), %rbx
// 679    jmp     .L5147
// 680    .p2align 4,,10
// 681    .p2align 3
// 682 .L5136:
// 683    movq    56(%r12), %rcx
// 684    movq    48(%r12), %rsi
// 685    movq    40(%r12), %rdi
// 686    movq    32(%r12), %r8
// 687    movq    24(%r12), %rbx
// 688    movq    16(%r12), %r10
// 689    movq    8(%r12), %rdx
// 690    jmp     .L5148
// 691    .p2align 4,,10
// 692    .p2align 3
// 693 .L5137:
// 694    movq    48(%r12), %rsi
// 695    movq    40(%r12), %rdi
// 696    movq    %rax, %r9
// 697    movq    32(%r12), %r8
// 698    movq    24(%r12), %rdx
// 699    movq    16(%r12), %r10
// 700    movq    8(%r12), %r11
// 701    jmp     .L5149
// 702    .p2align 4,,10
// 703    .p2align 3
// 704 .L5138:
// 705    movq    40(%r12), %rdi
// 706    movq    32(%r12), %r8
// 707    movq    24(%r12), %r9
// 708    movq    16(%r12), %r10
// 709    movq    8(%r12), %rcx
// 710    jmp     .L5150
// 711    .p2align 4,,10
// 712    .p2align 3
// 713 .L5141:
// 714    movq    16(%r12), %rdi
// 715    movq    8(%r12), %rcx
// 716    movq    %rax, %rdx
// 717    jmp     .L5153
// 718    .p2align 4,,10
// 719    .p2align 3
// 720 .L5142:
// 721    movq    8(%r12), %rcx
// 722    jmp     .L5154
// 723    .p2align 4,,10
// 724    .p2align 3
// 725 .L5139:

```

```

// 726 movq 32(%r12), %r8
// 727 movq 24(%r12), %r9
// 728 movq %rax, %rdx
// 729 movq 16(%r12), %rsi
// 730 movq 8(%r12), %rcx
// 731 jmp .L5151
// 732 .p2align 4,,10
// 733 .p2align 3
// 734 .L5140:
// 735 movq 24(%r12), %rsi
// 736 movq 16(%r12), %rdi
// 737 movq 8(%r12), %rcx
// 738 jmp .L5152
// 739 .p2align 4,,10
// 740 .p2align 3
// 741 .L5126:
// 742 addq $143936, %rsp
// 743 .cfi_remember_state
// 744 .cfi_def_cfa_offset 56
// 745 popq %rbx
// 746 .cfi_def_cfa_offset 48
// 747 popq %rbp
// 748 .cfi_def_cfa_offset 40
// 749 popq %r12
// 750 .cfi_def_cfa_offset 32
// 751 popq %r13
// 752 .cfi_def_cfa_offset 24
// 753 popq %r14
// 754 .cfi_def_cfa_offset 16
// 755 popq %r15
// 756 .cfi_def_cfa_offset 8
// 757 ret
// 758 .L5128:
// 759 .cfi_restore_state
// 760 movq %rsi, %rdi
// 761 sarq %rsi
// 762 movq %r10, %rdx
// 763 sarq $2, %rdi
// 764 leaq (%r12,%rdi,8), %rdi
// 765 movq (%rdi), %r8
// 766 movq %rax, (%rdi)
// 767 leaq (%r12,%rsi,8), %rax
// 768 movq 8(%r12), %rdi
// 769 movq %r8, (%r12)
// 770 movq (%rax), %rsi
// 771 movq %rdi, (%rax)
// 772 movq (%r12), %r8
// 773 xorl %eax, %eax
// 774 movq 16(%r12), %r9
// 775 movq %rsi, 8(%r12)
// 776 cmpq %rsi, %r8
// 777 seta %al
// 778 xorl %edi, %edi
// 779 cmpq %r9, %r8
// 780 seta %dil
// 781 addl %edi, %eax
// 782 xorl %edi, %edi
// 783 cmpq %rsi, %r8
// 784 setbe %dil
// 785 cmpq %rsi, %r9
// 786 movslq %eax, %rbx
// 787 adcl $0, %edi
// 788 movq %r8, (%r12,%rbx,8)
// 789 movslq %edi, %r8
// 790 addl %edi, %eax
// 791 movq %rsi, (%r12,%r8,8)
// 792 movl $3, %esi
// 793 subl %eax, %esi
// 794 movslq %esi, %rax
// 795 movq %r9, (%r12,%rax,8)
// 796 movq %r10, %r9
// 797 movq 8(%r12), %r8
// 798 cmpq (%r12), %r8
// 799 cmovbe %r12, %r9
// 800 cmpq %r10, %rcx
// 801 jbe .L5156
// 802 movq %rcx, %rax
// 803 jmp .L5162
// 804 .p2align 4,,10

```

```

// 805      .p2align 3
// 806 .L5173:
// 807      movq      (%r9), %rdi
// 808      movq      %rsi, (%r9)
// 809      leaq      16(%rdx), %rsi
// 810      addq      $8, %r9
// 811      movq      %rdi, 8(%rdx)
// 812      movq      %rbx, %rdx
// 813 .L5158:
// 814      cmpq      %rdx, %rax
// 815      jbe      .L5163
// 816 .L5162:
// 817      movq      8(%rdx), %rsi
// 818      leaq      8(%rdx), %rbx
// 819      cmpq      %rsi, %r8
// 820      ja      .L5173
// 821      jb      .L5172
// 822      leaq      16(%rdx), %rsi
// 823      movq      %rbx, %rdx
// 824      cmpq      %rdx, %rax
// 825      ja      .L5162
// 826 .L5163:
// 827      movq      %rsi, %xmm0
// 828      movq      %rcx, %xmm6
// 829      movq      8(%rdx), %rsi
// 830      movq      %rax, %rcx
// 831      subq      %rdx, %rcx
// 832      punpckldq %xmm6, %xmm0
// 833      addq      $2, %r15
// 834      movq      %rsi, %rdi
// 835      sarq      $63, %rcx
// 836      subq      8(%rax), %rdi
// 837      movups   %xmm0, -96(%rsp,%r15,8)
// 838      andq      %rdi, %rcx
// 839      subq      %rcx, %rsi
// 840      movq      %rsi, 8(%rdx)
// 841      addq      %rcx, 8(%rax)
// 842      leaq      -8(%r9), %rcx
// 843      movq      %rcx, %rdx
// 844      subq      %r12, %rdx
// 845      testq     %rdx, %rdx
// 846      jg      .L5164
// 847      jmp      .L5127
// 848      .p2align 4,,10
// 849      .p2align 3
// 850 .L5161:
// 851      subq      $8, %rax
// 852 .L5172:
// 853      movq      (%rax), %rdi
// 854      cmpq      %rdi, %r8
// 855      jb      .L5161
// 856      movq      %rdi, 8(%rdx)
// 857      subq      $8, %rax
// 858      movq      %rsi, 8(%rax)
// 859      movq      %rbx, %rsi
// 860      jmp      .L5158
// 861 .L5156:
// 862      movq      %r11, %rsi
// 863      movq      %rcx, %rax
// 864      jmp      .L5163
// 865      .cfi_endproc

```