Quicksort 'Magnetica' - FASTEST Quicksort?!

A short review by sanmayce@sanmayce.com, 2021-Nov-27

```
// If below define is commented then we have "middle Pivot standalone Magnetica", otherwise "median of 5 Pivot InsertionSort Magnetica".
#define FinishWithInsertionSort
// Uncomment either 2 (fastest so far) or 3:
#define revision2
//#define revision3
// Thus we have 4 variants:
// Variant #1: "middle Pivot standalone Magnetica r.2"
// Variant #2: "middle Pivot standalone Magnetica r.3"
// Variant #3: "median of 5 Pivot InsertionSort Magnetica r.2"
// Variant #4: "median of 5 Pivot InsertionSort Magnetica r.3"
void swapUnconditional(uint64_t *a, uint64_t *b) { uint64_t t = *a; *a = *b; *b = t; }
// Many thanks go to all the coders at: https://stackoverflow.com/questions/2786899/fastest-sort-of-fixed-length-6-int-array
#define minSO(x, y) (y ^ ((x ^ y) & -(x < y)))
#define maxSO(x, y) (x ^ ((x ^ y) & -(x < y)))
//    _____                         __  .__               
//   /     \ _____     ____   _____/  |_|__| ____ _____   
//  /  \ /  \\__  \   / ___\ /    \_/ __ \   __\  |/ ___\\__  \  
// /    Y    \/ __ \_/ /_/  >   |  \  ___/|  | |  \  \___ / __ \_
// \____|__  (____  /\___  /|___|  /\___  >__| |__|\___  >____  /
//         \/     \/_____/      \/     \/             \/     \/ 
// Written by Sanmayce, 2021-Nov-27
// The indexes are signed, but the elements are unsigned, targeting 64bit pointers mostly.
void Quicksort_QB64_v6(uint64_t QWORDS[], int64_t Left, int64_t Right) {
    #ifdef FinishWithInsertionSort
    int InsertionsortTHRESHOLD = 13;
    #else
    int InsertionsortTHRESHOLD = 0;
    #endif
    int64_t Indx, Jndx, PL, PR;
    int64_t Stack[99999];
    int64_t StackPtr = 0;
    uint64_t Pivot;
    int64_t LeftBackup = Left;
    int64_t RightBackup = Right;
    register uint64_t x0,x1,x2,x3,x4,x5;
    int o0,o1,o2,o3,o4,o5;
    int64_t TheMiddleOfMiddle;
    int Stefan_Edelkamp_Armin_Weiss1, Stefan_Edelkamp_Armin_Weiss2, Stefan_Edelkamp_Armin_Weiss3;
    uint64_t tmp;
    StackPtr++; Stack[StackPtr] = Left;
    StackPtr++; Stack[StackPtr] = Right;
    do {
        Right = Stack[StackPtr];
        Left = Stack[StackPtr - 1];
        StackPtr = StackPtr - 2;
        //do {
        for(;(Left + InsertionsortTHRESHOLD < Right);) {
            Jndx = Right;
            PL = Left;
            PR = Left;
```

```
    #ifdef FinishWithInsertionSort
            TheMiddleOfMiddle = Left + ((Right-Left)>>2); // (4Left + Right - Left)/4
            x0 = QWORDS[TheMiddleOfMiddle+0];
            x1 = QWORDS[TheMiddleOfMiddle+1];
            x2 = QWORDS[TheMiddleOfMiddle+2];
            x3 = QWORDS[TheMiddleOfMiddle+3];
            x4 = QWORDS[TheMiddleOfMiddle+4];
            //x5 = QWORDS[TheMiddleOfMiddle+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 = 10-(o0+o1+o2+o3);
            //o4 = (x4>=x0)+(x4>=x1)+(x4>=x2)+(x4>=x3)+(x4>x5);
            //o5 = 15-(o0+o1+o2+o3+o4);
            QWORDS[TheMiddleOfMiddle+o0]=x0; QWORDS[TheMiddleOfMiddle+o1]=x1; QWORDS[TheMiddleOfMiddle+o2]=x2; QWORDS[TheMiddleOfMiddle+o3]=x3; QWORDS[TheMiddleOfMiddle+o4]=x4; //QWORDS[TheMiddleOfMiddle+o5]=x5;
            swapUnconditional (&QWORDS[TheMiddleOfMiddle+2], &QWORDS[PR]);
            Pivot = QWORDS[PR];
    #else
            swapUnconditional (&QWORDS[(Left + Right)>>1], &QWORDS[PR]);
            Pivot = QWORDS[PR];
    #endif
    #ifdef revision2
            for (;PR < Jndx;) {
                PR = PR + 1;
                if (Pivot > QWORDS[PR]) {
                    swapUnconditional (&QWORDS[PL], &QWORDS[PR]);
                    PL = PL + 1;
                //} else if (Pivot == QWORDS[PR]) {
                } else if (Pivot < QWORDS[PR]) {
                    for (;Pivot < QWORDS[Jndx];) {
                        Jndx = Jndx - 1;
                    }
                    if (PR < Jndx) swapUnconditional (&QWORDS[PR], &QWORDS[Jndx]);
                    Jndx = Jndx - 1;
                    PR = PR - 1;
                }
            }
    #endif
/*
; mark_description "Intel(R) C++ Compiler XE for applications running on Intel(R) 64, Version 15.0.0.108 Build 20140726";
; ae-64+2=76 bytes i.e. (94-76=18 less), 5/1 conditional/unconditional jumps i.e. 2-1=1 less unconditional jump than r.1
; 'Magnetica' partitioning, mainloop rev.2[
.B4.5::
  00064 48 ff c5        inc rbp
  00067 48 8b 14 e9     mov rdx, QWORD PTR [rcx+rbp*8]
  0006b 4c 3b ea        cmp r13, rdx
  0006e 77 2c           ja .B4.14
.B4.6::
  00070 73 39           jae .B4.15
.B4.7::
  00072 4e 8b 3c d9     mov r15, QWORD PTR [rcx+r11*8]
  00076 4d 3b ef        cmp r13, r15
  00079 73 0c           jae .B4.11
.B4.9::
  0007b 49 ff cb        dec r11
  0007e 4e 8b 3c d9     mov r15, QWORD PTR [rcx+r11*8]
  00082 4d 3b ef        cmp r13, r15
  00085 72 f4           jb .B4.9
.B4.11::
  00087 49 3b eb        cmp rbp, r11
  0008a 7d 08           jge .B4.13
.B4.12::
  0008c 4c 89 3c e9     mov QWORD PTR [rcx+rbp*8], r15
  00090 4a 89 14 d9     mov QWORD PTR [rcx+r11*8], rdx
.B4.13::
  00094 49 ff cb        dec r11
  00097 48 ff cd        dec rbp
  0009a eb 0f           jmp .B4.15
.B4.14::
  0009c 4e 8b 3c f1     mov r15, QWORD PTR [rcx+r14*8]
  000a0 4a 89 14 f1     mov QWORD PTR [rcx+r14*8], rdx
  000a4 49 ff c6        inc r14
  000a7 4c 89 3c e9     mov QWORD PTR [rcx+rbp*8], r15
.B4.15::
  000ab 49 3b eb        cmp rbp, r11
  000ae 7c b4           jl .B4.5
; 'Magnetica' partitioning, mainloop rev.2]
*/
```

```
    #ifdef revision3
            for (;PR < Jndx;) {
                // Many thanks go to Orson Peters for sharing his Pattern-defeating quicksort (pdqsort) at GitHub.
                // This is due to a new technique described in "BlockQuicksort: How Branch Mispredictions don't affect Quicksort" by Stefan Edelkamp and Armin Weiss.
                Stefan_Edelkamp_Armin_Weiss1 = (Pivot > QWORDS[PR + 1]);
                Stefan_Edelkamp_Armin_Weiss2 = (Pivot == QWORDS[PR + 1]);
                Stefan_Edelkamp_Armin_Weiss3 = (Pivot < QWORDS[PR + 1]);
                tmp = minSO(QWORDS[PL], QWORDS[PR + 1]); QWORDS[PR + 1] = maxSO(QWORDS[PL], QWORDS[PR + 1]); QWORDS[PL] = tmp; // Since Pivot is equal to QWORDS[PL]
                PL = PL + Stefan_Edelkamp_Armin_Weiss1;
                PR = PR + Stefan_Edelkamp_Armin_Weiss1;
                PR = PR + Stefan_Edelkamp_Armin_Weiss2;
                if (Stefan_Edelkamp_Armin_Weiss3) {
                    for (;Pivot < QWORDS[Jndx];) {
                        Jndx = Jndx - 1;
                    }
                    if (PR + 1 < Jndx) swapUnconditional (&QWORDS[PR + 1], &QWORDS[Jndx]);
                    Jndx = Jndx - 1;
                }
            }
    #endif
/*
; mark_description "Intel(R) C++ Compiler XE for applications running on Intel(R) 64, Version 15.0.0.108 Build 20140726";
; f9-73+6=140 bytes, 4/0 conditional/unconditional jumps
; 'Magnetica' partitioning, mainloop rev.3[
.B7.5::
  00073 4a 8b 5c f1 08   mov rbx, QWORD PTR [8+rcx+r14*8]
  00078 33 d2            xor edx, edx
  0007a 4c 3b eb         cmp r13, rbx
  0007d 4a 8b 2c c1      mov rbp, QWORD PTR [rcx+r8*8]
  00081 49 0f 47 d4      cmova rdx, r12
  00085 45 33 ff         xor r15d, r15d
  00088 48 3b eb         cmp rbp, rbx
  0008b 48 89 ee         mov rsi, rbp
  0008e 41 0f 92 c7      setb r15b
  00092 48 33 f3         xor rsi, rbx
  00095 41 f7 df         neg r15d
  00098 4d 63 ff         movsxd r15, r15d
  0009b 49 23 f7         and rsi, r15
  0009e 48 33 ee         xor rbp, rsi
  000a1 4c 3b eb         cmp r13, rbx
  000a4 4a 89 6c f1 08   mov QWORD PTR [8+rcx+r14*8], rbp
  000a9 bd 00 00 00 00   mov ebp, 0
  000ae 49 0f 44 ec      cmove rbp, r12
  000b2 48 33 f3         xor rsi, rbx
  000b5 4a 89 34 c1      mov QWORD PTR [rcx+r8*8], rsi
  000b9 4c 03 c2         add r8, rdx
  000bc 48 03 d5         add rdx, rbp
  000bf 4c 03 f2         add r14, rdx
  000c2 4c 3b eb         cmp r13, rbx
  000c5 73 2f            jae .B7.13
.B7.6::
  000c7 4a 8b 14 c9      mov rdx, QWORD PTR [rcx+r9*8]
  000cb 4c 3b ea         cmp r13, rdx
  000ce 73 0c            jae .B7.10
.B7.8::
  000d0 49 ff c9         dec r9
  000d3 4a 8b 14 c9      mov rdx, QWORD PTR [rcx+r9*8]
  000d7 4c 3b ea         cmp r13, rdx
  000da 72 f4            jb .B7.8
.B7.10::
  000dc 49 8d 5e 01      lea rbx, QWORD PTR [1+r14]
  000e0 4c 3b cb         cmp r9, rbx
  000e3 7e 0e            jle .B7.12
.B7.11::
  000e5 4a 8b 5c f1 08   mov rbx, QWORD PTR [8+rcx+r14*8]
  000ea 4a 89 54 f1 08   mov QWORD PTR [8+rcx+r14*8], rdx
  000ef 4a 89 1c c9      mov QWORD PTR [rcx+r9*8], rbx
.B7.12::                        ; Preds .B7.10 .B7.11
  000f3 49 ff c9         dec r9
.B7.13::
  000f6 4d 3b f1         cmp r14, r9
  000f9 0f 8c 74 ff ff
        ff               jl .B7.5
; 'Magnetica' partitioning, mainloop rev.3]
*/
            Jndx = PL - 1;
            Indx = PR + 1;
/*
            // '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 ]
*/
/*
; mark_description "Intel(R) C++ Compiler XE for applications running on Intel(R) 64, Version 15.0.0.108 Build 20140726";
; c6-6a+2=94 bytes, 5/2 conditional/unconditional jumps
; 'Magnetica' partitioning, mainloop [
.B4.5::
  0006a 4e 3b 5c e9 08   cmp r11, QWORD PTR [8+rcx+r13*8]
  0006f 77 37            ja .B4.15
.B4.6::
  00071 75 08            jne .B4.8
.B4.7::
  00073 49 89 d5         mov r13, rdx
  00076 48 ff c2         inc rdx
  00079 eb 48            jmp .B4.16
.B4.8::
  0007b 4e 8b 34 c1      mov r14, QWORD PTR [rcx+r8*8]
  0007f 4d 3b de         cmp r11, r14
  00082 73 0c            jae .B4.12
.B4.10::
  00084 49 ff c8         dec r8
  00087 4e 8b 34 c1      mov r14, QWORD PTR [rcx+r8*8]
  0008b 4d 3b de         cmp r11, r14
  0008e 72 f4            jb .B4.10
.B4.12::
  00090 4c 3b c2         cmp r8, rdx
  00093 7e 0e            jle .B4.14
.B4.13::
  00095 4e 8b 7c e9 08   mov r15, QWORD PTR [8+rcx+r13*8]
  0009a 4e 89 74 e9 08   mov QWORD PTR [8+rcx+r13*8], r14
  0009f 4e 89 3c c1      mov QWORD PTR [rcx+r8*8], r15
.B4.14::
  000a3 49 ff c8         dec r8
  000a6 eb 1b            jmp .B4.16
.B4.15::
  000a8 4e 8b 74 e9 08   mov r14, QWORD PTR [8+rcx+r13*8]
  000ad 4e 8b 3c e1      mov r15, QWORD PTR [rcx+r12*8]
  000b1 4e 89 34 e1      mov QWORD PTR [rcx+r12*8], r14
  000b5 49 ff c4         inc r12
  000b8 4e 89 7c e9 08   mov QWORD PTR [8+rcx+r13*8], r15
  000bd 49 89 d5         mov r13, rdx
  000c0 48 ff c2         inc rdx
.B4.16::
  000c3 4d 3b e8         cmp r13, r8
  000c6 7c a2            jl .B4.5
; 'Magnetica' partitioning, mainloop ]
*/

            if (Indx + InsertionsortTHRESHOLD < Right) {
                StackPtr = StackPtr + 2;
                Stack[StackPtr - 1] = Indx;
                Stack[StackPtr] = Right;
            }
            Right = Jndx;
        } //while (Left + InsertionsortTHRESHOLD < Right);
    } while (StackPtr != 0);
    #ifdef FinishWithInsertionSort
```

```
    for (Indx=LeftBackup+1; Indx <= RightBackup; Indx++) {
        Jndx = Indx;
        for (;Jndx >= 1;) {
            if (QWORDS[Jndx-1] > QWORDS[Jndx]) swapUnconditional (&QWORDS[Jndx-1], &QWORDS[Jndx]); else break;
            Jndx = Jndx - 1;
        }
    }
    #endif
}
// 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.qb64.org/forum/index.php?topic=3518.msg137645#msg137645
// https://www.linuxquestions.org/questions/programming-9/qsort-vs-%27magnetica%27-quicksort-4175703333/#post6239782
```

Benchmarking:

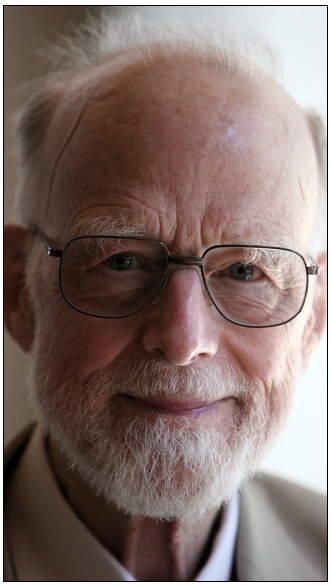
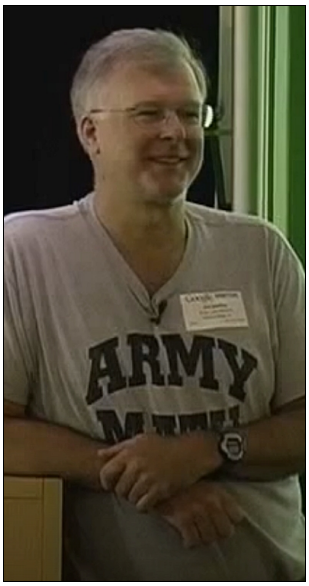
Files being sorted (taking 8 bytes as an element back-to-back i.e. filesize/8 elements in total):
10/30/2021 15:29 178,708,944 22338618_QWORDS.bin **! used as testdataset 'few' !**
Files being sorted (taking 8 bytes as an element at each position i.e. filesize-8+1 elements in total):
03/26/2014 08:14 24,823,016 mobythesaurus.txt **! used as testdataset 'many' !**
11/12/2021 14:52 2,009,333,760 Fedora-Workstation-Live-x86_64-35-1.2.iso **! used as testdataset 'ALL' !**

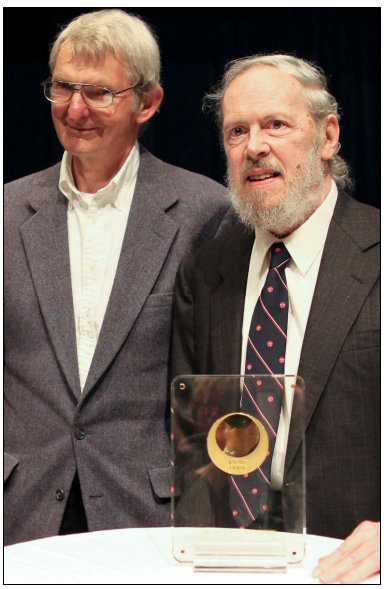
Laptop Intel i5-7200U 3.1GHz max turbo, 36GB DDR4 2133MHz:

Performer/Keys	FEW distinct		MANY distinct		~ALL distinct	
Operating System, Compiler, -O3	Windows 10, Intel v15.0	Fedora 33, GCC 10.2.1	Windows 10, Intel v15.0	Fedora 33, GCC 10.2.1	Windows 10, Intel v15.0	Fedora 33, GCC 10.2.1
qsort	58 seconds	360 seconds	343 seconds	535 seconds	444 seconds	519 seconds
Magnetica r.2	**35** seconds	**35** seconds	214 seconds	**222** seconds	**271** seconds	**286** seconds
Bentley-McIlroy	39 seconds	41 seconds	**203** seconds	230 seconds	275 seconds	313 seconds

Legend (The time is exactly the Sort process time):
FEW = 2,233,861,800 keys, of them distinct = 10;
MANY = 2,482,300,900 keys, of them distinct = 2,847,531;
~ALL = 2,009,333,753 keys, of them distinct = 1,912,608,132
Note: The results in bold are fastest, for some reason (perhaps due to the different pivot choosing) Magnetica lags behind in MANY scenario.

Quick highlights:
- GLIBC's qsort is just trash;
- Intel's qsort is trashy;
- Bentley-McIlroy is good, however the disassembly (-O3) shows many weird vector instructions used, not good for a scalar code;
- Magnetica r.2 is nice, the disassembly (-O3) gladdens the eyes.




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**