

A short review by [sanmayce@sanmayce.com](mailto:sanmayce@sanmayce.com), 2021-Dec-02

A black and white photograph of a Shinkansen (bullet train) stopped at a station platform. The train is white with dark stripes and is positioned on a raised track. The platform has a concrete wall and a roof structure.

[illegible]

```
#ifndef FinishWithInsertionSort
int InsertionSortTHRESHOLD = 13;
#else
int InsertionSortTHRESHOLD = 0;
#endif
int64_t Jndx, 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 dx, 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         seth 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::
000f3 49 ff c9           dec r9
; Prefs .B7.10 .B7.11
.B7.13::
000f6 4d 3b f1          cmp r14, r9
000f9 0f 8c 74 ff ff     jl .B7.5
ff
; '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.gb64.org/forum/index.php?topic=3518.msg137645#msg137645
// https://www.linuxquestions.org/questions/programming-9/quicksort-vs-%27magnetica%27-quicksort-4175703333/#post6299782

```

#### 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 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,	Windows 10,	Fedora 33,	Windows 10,	Fedora 33,	Windows 10,	Fedora 33,
Compiler, -O3	Intel v15.0	GCC 10.2.1	Intel v15.0	GCC 10.2.1	Intel v15.0	GCC 10.2.1
qsort	58 seconds	360 seconds	343 seconds	535 seconds	444 seconds	519 seconds
Magnetica r.2	<b>35</b> seconds	<b>35</b> seconds	214 seconds	<b>222</b> seconds	<b>271</b> seconds	<b>286</b> seconds
Bentley-McIlroy	39 seconds	41 seconds	<b>203</b> 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.

#### Add-on, 2021-Dec-02:

Now, after so much instrumental runs/revisions, it's time for the best Quicksort function known to me - Magnetica r.4, some highlights:

- Open-source and 100% FREE, licenseless;
- 113 lines of scalar C, resulting in 380 lines of scalar Disassembly;
- Most excellent (due to the Magnetica partitioning scheme) when most/many array's elements are repetitive;
- Unbreakable! Iterative with a fail-safe, never overflows the user-defined stack (fallbacks to Insertionsort when stack is full), take heed, NOT the runtime stack;
- Robust against nastiest quadratic  $O(N^2)$  case, at gunpoint would confess rather  $O((N/4)^2)$  because of median of 7 Pivot - each time we secure 4 elements in either direction;
- The first Quicksort with a ... GearBox - 2 gears: Pivot (either median of 3 or median of 7) chosen branchlessly, depending on the two partitions ratio;
- Ideal for benchmarking since the code is tight and accesses RAM and the caches, a lot;
- Benchmarked as it should, on Linux with the best hardware counter reader - 'perf stat -d' - covering the three major sort scenarios - FEW/MANY/ALL distinct keys, 2+ BILLION of total keys;
- A nifty playground for further PRACTICAL ... playoffs.

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

Performer/Keys	FEW distinct		MANY distinct		"ALL" distinct		"ALLmore distinct"
Operating System,	Windows 10,	Fedora 33,	Windows 10,	Fedora 33,	Windows 10,	Fedora 33,	Windows 10, Fedora 33,
Compiler, -O3	Intel v15.0	GCC 10.2.1	Intel v15.0	GCC 10.2.1	Intel v15.0	GCC 10.2.1	Intel v15.0   GCC 10.2.1
qsort	58 seconds	364 seconds	339 seconds	536 seconds	441 seconds	520 seconds	862 seconds   1019 seconds
Magnetica r.4	<b>36</b> seconds	<b>36</b> seconds	220 seconds	232 seconds	<b>275</b> seconds	<b>295</b> seconds	<b>542</b> seconds   <b>581</b> seconds
Bentley-McIlroy	38 seconds	40 seconds	<b>204</b> seconds	<b>230</b> seconds	276 seconds	313 seconds	544 seconds   607 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

"ALLmore" = 3,803,483,825 keys, of them distinct = 3,346,259,533

Notes:

- The whole package (except the 3rd and 4th datasets) is downloadable at:

[www.samayce.com/Quicksort\\_says\\_rev8.zip](http://www.samayce.com/Quicksort_says_rev8.zip)

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

- To reproduce the roster, run on Windows or Linux:

- BENCH\_ICL.BAT

- sh bench gcc.sh

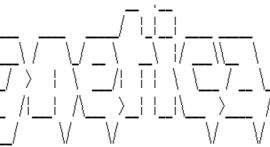
- 3rd dataset is downloadable at:

[https://download.fedoraproject.org/pub/fedora/linux/releases/35/Workstation/x86\\_64/iso/Fedora-Workstation-Live-x86\\_64-35-1.2.iso](https://download.fedoraproject.org/pub/fedora/linux/releases/35/Workstation/x86_64/iso/Fedora-Workstation-Live-x86_64-35-1.2.iso)

- 4th dataset is downloadable at:

<https://download.fedoraproject.org/pub/fedora/linux/releases/35/Workstation/aarch64/images/Fedora-Workstation-35-1.2.aarch64.raw.xz>

#### Here comes the C source:

```
00,001 #define StackEntries 99999
00,002 #define FinishWithInsertionSort
00,003 #define revision4
00,004 void swapUnconditional(uint64_t *a, uint64_t *b) { uint64_t t = *a; *a = *b; *b = t; }
00,005 #define min50(x, y) (y ^ ((x ^ y) & -(x < y)))
00,006 #define max50(x, y) (x ^ ((x ^ y) & -(x < y)))
00,007 //
00,008 // 
00,009 //
00,010 //
00,011 //
00,012 //
00,013 // Written by Sammayce, 2021-Dec-02
00,014 void Quicksort_QB64_v7(uint64_t QWORDS[], int64_t Left, int64_t Right) {
00,015     #ifdef FinishWithInsertionSort
00,016         int InsertionsortTHRESHOLD = 17;
00,017     #else
00,018         int InsertionsortTHRESHOLD = 0;
00,019     #endif
00,020     int64_t Jndx, Jndx, PL, PR;
00,021     int64_t Stack[StackEntries];
00,022     int64_t StackPtr = 0;
00,023     uint64_t Pivot;
00,024     int64_t LeftBackup = Left;
00,025     int64_t RightBackup = Right;
00,026     register uint64_t x0,x1,x2,x3,x4,x5,x6;
00,027     int o0,o1,o2,o3,o4,o5,o6;
00,028     int64_t TheMiddleOfMiddle;
00,029     int GearBox = 0; // 0 is Median of 3; 1 is Median of 7; Shifting to higher gear if BiggerPartition:SmallerPartition ratio is > 64:1
00,030     StackPtr++; Stack[StackPtr] = Left;
00,031     StackPtr++; Stack[StackPtr] = Right;
00,032     do {
00,033         Right = Stack[StackPtr];
00,034         Left = Stack[StackPtr - 1];
00,035         StackPtr = StackPtr - 2;
00,036         for( (Left + InsertionsortTHRESHOLD < Right); ) { // For instance, 1 + 17 < 19 i.e. (19-1)/4+6 < 19
00,037             Jndx = Right;
00,038             PL = Left;
00,039             PR = Left;
00,040         #ifdef FinishWithInsertionSort
00,041             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*6*(4Left + Right - Left) <= 3Right-3Left or (4*6)/3 <= Right-Left i.e. 8 <= Right-Left as a minimum condition to enter the 'for'.
00,042             x0 = QWORDS[TheMiddleOfMiddle+0];
00,043             x1 = QWORDS[TheMiddleOfMiddle+1];
00,044             x2 = QWORDS[TheMiddleOfMiddle+2];
00,045         if (GearBox == 0) { // Median of 5 proves more effective than 7, however 7 betters better the nastiest N^2
00,046             o0 = (x0>x1)+(x0>x2);
00,047             o1 = (x1>x0)+(x1>x2);
00,048             o2 = (0+1+2)-(o0+o1);
00,049             QWORDS[TheMiddleOfMiddle+o0]=x0; QWORDS[TheMiddleOfMiddle+o1]=x1; QWORDS[TheMiddleOfMiddle+o2]=x2;
00,050             swapUnconditional (&QWORDS[TheMiddleOfMiddle+1], &QWORDS[PR]); //2nd
00,051         } else {
00,052             x3 = QWORDS[TheMiddleOfMiddle+3];
00,053             x4 = QWORDS[TheMiddleOfMiddle+4];
00,054             x5 = QWORDS[TheMiddleOfMiddle+5];
00,055             x6 = QWORDS[TheMiddleOfMiddle+6];
00,056             o0 = (x0>x1)+(x0>x2)+(x0>x3)+(x0>x4)+(x0>x5)+(x0>x6);
00,057             o1 = (x1>x0)+(x1>x2)+(x1>x3)+(x1>x4)+(x1>x5)+(x1>x6);
00,058             o2 = (x2>x0)+(x2>x1)+(x2>x3)+(x2>x4)+(x2>x5)+(x2>x6);
00,059             o3 = (x3>x0)+(x3>x1)+(x3>x2)+(x3>x4)+(x3>x5)+(x3>x6);
00,060             o4 = (x4>x0)+(x4>x1)+(x4>x2)+(x4>x3)+(x4>x5)+(x4>x6);
00,061             o5 = (x5>x0)+(x5>x1)+(x5>x2)+(x5>x3)+(x5>x4)+(x5>x6);
00,062             o6 = (0+1+2+3+4+5+6)-(o0+o1+o2+o3+o4+o5);
00,063             QWORDS[TheMiddleOfMiddle+o0]=x0; QWORDS[TheMiddleOfMiddle+o1]=x1; QWORDS[TheMiddleOfMiddle+o2]=x2; QWORDS[TheMiddleOfMiddle+o3]=x3; QWORDS[TheMiddleOfMiddle+o4]=x4;
00,064             QWORDS[TheMiddleOfMiddle+o5]=x5; QWORDS[TheMiddleOfMiddle+o6]=x6;
00,065             swapUnconditional (&QWORDS[TheMiddleOfMiddle+3], &QWORDS[PR]); //4th
00,066         }
00,067         Pivot = QWORDS[PR];
00,068     #else
00,069         swapUnconditional (&QWORDS[(Left + Right)>>1], &QWORDS[PR]);
00,070     #endif
00,071     } while (Left < Right);
00,072 }
```

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

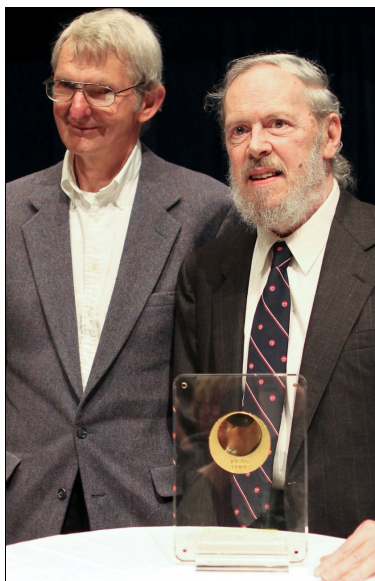
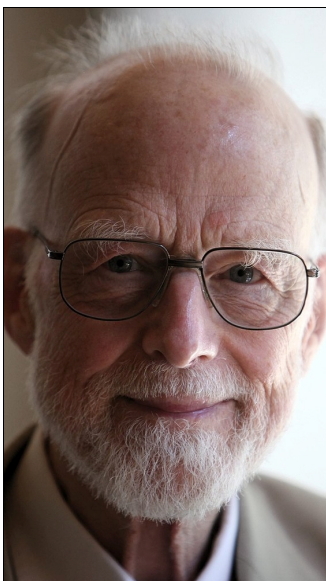


```

00,069     Pivot = QWORDS[PR];
00,070 #endif
00,071 #ifdef revision4
00,072     for (;PR < Jndx;) {
00,073         PR = PR + 1;
00,074         if (Pivot > QWORDS[PR]) {
00,075             swapUnconditional (&QWORDS[PL], &QWORDS[PR]);
00,076             PL = PL + 1;
00,077         } else if (Pivot < QWORDS[PR]) {
00,078             for (;Pivot < QWORDS[Jndx];) {
00,079                 Jndx = Jndx - 1;
00,080             }
00,081             if (PR < Jndx) swapUnconditional (&QWORDS[PR], &QWORDS[Jndx]);
00,082             Jndx = Jndx - 1;
00,083             PR = PR - 1;
00,084         }
00,085     }
00,086 #endif
00,087     Jndx = PL - 1;
00,088     Indx = PR + 1;
00,089 #ifdef FinishWithInsertionSort
00,090     GearBox = ( maxSO((Right-Indx), (Jndx-Left)) > minSO((Right-Indx), (Jndx-Left))<<6 ); // same as MAX/MIN > 64, i.e. Gear=1 if we need Median of 7
00,091 #endif
00,092     if (Indx + InsertionsortTHRESHOLD < Right) { // still refusing to go (always) with the smaller partition first...
00,093         StackPtr = StackPtr + 2;
00,094         Stack[StackPtr - 1] = Indx;
00,095         Stack[StackPtr] = Right;
00,096 #ifdef FinishWithInsertionSort
00,097         StackPtr = StackPtr*(StackPtr + 2 <= StackEntries-1); // StackPtr should be enforced FALSE; also, ensure the next 'push' above is sanctioned - the max index of Stack[] being 'StackEntries-1'
00,098         Right = Right*(StackPtr + 2 <= StackEntries-1); // Left + InsertionsortTHRESHOLD < Right should be enforced FALSE;
00,099 #endif
00,100     }
00,101     Right = Jndx;
00,102 } //while (Left + InsertionsortTHRESHOLD < Right);
00,103 } while ( (StackPtr != 0) );
00,104 #ifdef FinishWithInsertionSort
00,105 for (Indx=LeftBackup+1; Indx <= RightBackup; Indx++) {
00,106     Jndx = Indx;
00,107     for (;Jndx >= 1;) {
00,108         if (QWORDS[Jndx-1] > QWORDS[Jndx]) swapUnconditional (&QWORDS[Jndx-1], &QWORDS[Jndx]); else break;
00,109         Jndx = Jndx - 1;
00,110     }
00,111 }
00,112 #endif
00,113 }

```

This very code is now in the benchmark.  
Can we do better?!



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