Lab 4
Isa Dzhumabaev

Save the output of the program in your lab report

```
$ ./a.out
validated
        quadsort: sorted  100000 i64s in 0.013857 seconds. (random order)
          qsort: sorted  100000 i64s in 0.012997 seconds. (random order)

        quadsort: sorted  100000 i32s in 0.013584 seconds. (random order)
          qsort: sorted  100000 i32s in 0.012799 seconds. (random order)

        quadsort: sorted  100000 i32s in 0.000263 seconds. (ascending)
          qsort: sorted  100000 i32s in 0.000528 seconds. (ascending)

        quadsort: sorted  100000 i32s in 0.002616 seconds. (ascending saw)
          qsort: sorted  100000 i32s in 0.007200 seconds. (ascending saw)

        quadsort: sorted  100000 i32s in 0.010146 seconds. (generic order)
          qsort: sorted  100000 i32s in 0.004536 seconds. (generic order)

        quadsort: sorted  100000 i32s in 0.000364 seconds. (descending order)
          qsort: sorted  100000 i32s in 0.002670 seconds. (descending order)

        quadsort: sorted  100000 i32s in 0.002515 seconds. (descending saw)
          qsort: sorted  100000 i32s in 0.005236 seconds. (descending saw)

        quadsort: sorted  100000 i32s in 0.003542 seconds. (random tail)
          qsort: sorted  100000 i32s in 0.006575 seconds. (random tail)

        quadsort: sorted  100000 i32s in 0.007406 seconds. (random half)
          qsort: sorted  100000 i32s in 0.011522 seconds. (random half)

        quadsort: sorted  100000 i32s in 0.005104 seconds. (wave order)
          qsort: sorted  100000 i32s in 0.006501 seconds. (wave order)

        quadsort: sorted  100000 i32s in 0.009738 seconds. (stable)
          qsort: sorted  100000 i32s in 0.005025 seconds. (unstable)

        quadsort: sorted       8 i32s in 0.000052 seconds. (random 1-8)
          qsort: sorted       8 i32s in 0.000035 seconds. (random 1-8)
```

```
quadsort: sorted      15 i32s in 0.000145 seconds. (random 9-15)
  qsort: sorted       15 i32s in 0.000175 seconds. (random 9-15)

quadsort: sorted      15 i64s in 0.000096 seconds. (random 1-15)
  qsort: sorted       15 i64s in 0.000106 seconds. (random 1-15)

quadsort: sorted      63 i32s in 0.000898 seconds. (random 16-63)
  qsort: sorted       63 i32s in 0.001303 seconds. (random 16-63)

quadsort: sorted     127 i32s in 0.002620 seconds. (random 64-127)
  qsort: sorted      127 i32s in 0.004788 seconds. (random 64-127)

quadsort: sorted     255 i32s in 0.005723 seconds. (random 128-255)
  qsort: sorted      255 i32s in 0.011384 seconds. (random 128-255)

quadsort: sorted     511 i32s in 0.014032 seconds. (random 256-511)
  qsort: sorted      511 i32s in 0.025942 seconds. (random 256-511)

quadsort: sorted    1023 i32s in 0.040460 seconds. (random 512-1023)
  qsort: sorted     1023 i32s in 0.058153 seconds. (random 512-1023)

quadsort: sorted    1023 i32s in 0.024502 seconds. (random 1-1023)
  qsort: sorted     1023 i32s in 0.036731 seconds. (random 1-1023)

quadsort: sorted    2047 i32s in 0.110264 seconds. (random 1024-2047)
  qsort: sorted     2047 i32s in 0.127463 seconds. (random 1024-2047)

quadsort: sorted    4095 i32s in 0.479185 seconds. (random 2048-4095)
  qsort: sorted     4095 i32s in 0.512234 seconds. (random 2048-4095)
```

What is the main advantage of this sort? Explain shortly in 4-5 sentences

As I understood the main advantage is quad swap that reduces a lot of redundant swaps. Another thing is detecting sorted parts in the array in order to skip sorting and merging, which would be useless. And also it does not check bounds of the array, instead it compares last elements of two arrays. But as it was said by author the main advantage is quad swap.

How is the program measuring the running time?

Time is measured using utime.h header. Just saving the starting time and end time then subtraction.

What about the space issues?

Answer is in readme file of author ☺

Cache

Because quadsort uses n / 2 swap memory its cache utilization is not as ideal as in-place sorts. However, in-place sorting of random data results in suboptimal swapping. Based on my benchmarks it appears that quadsort is always faster than in-place sorts for array sizes that do not exhaust the L1 cache, which can be up to 64KB on modern processors.

Can you increase to more than 4 (quad)?

I think that theoreticaly it is possible, but the quiestion is «do we need it?». I think that in most cases best solutions are created from totaly new ideas rather from evloving of old solution.