Assignment 4: Quicksort and Mergesort - Analysis Document

Muyuan Zhang

2022-11-21

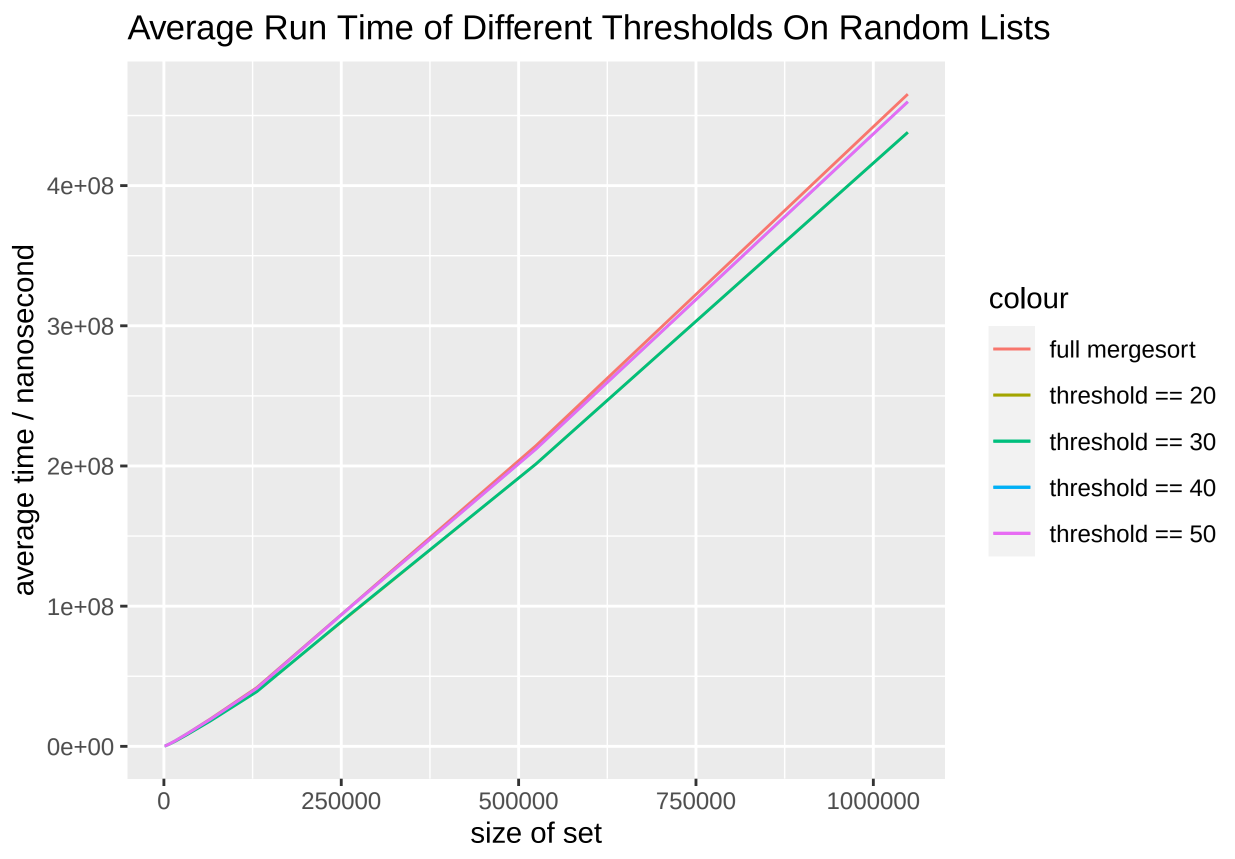
1. Who are your team members?

- Gloria Dukuzeyesu

2. Mergesort Threshold Experiment: Determine the best threshold value for which mergesort switches over to insertion sort. Your list sizes should cover a range of input sizes to make meaningful plots, and should be large enough to capture accurate running times. To ensure a fair comparison, use the same set of permuted -order lists for each threshold value. Keep in mind that you can't resort the same ArrayList over and over, as the second time the order will have changed. Create an initial input and copy it to a temporary ArrayList for each test (but make sure you subtract the copy time from your timing results!). Use the timing techniques we already demonstrated, and be sure to choose a large enough value of timesToLoop to get a reasonable average of running times. Note that the best threshold value may be a constant value or a fraction of the list size. Plot the running times of your threshold mergesort for five different threshold values on permuted-order lists (one line for each threshold value). In the five different threshold values, be sure to include the threshold value that simulates a full mergesort, i.e., never switching to insertion sort (and identify that line as such in your plot).

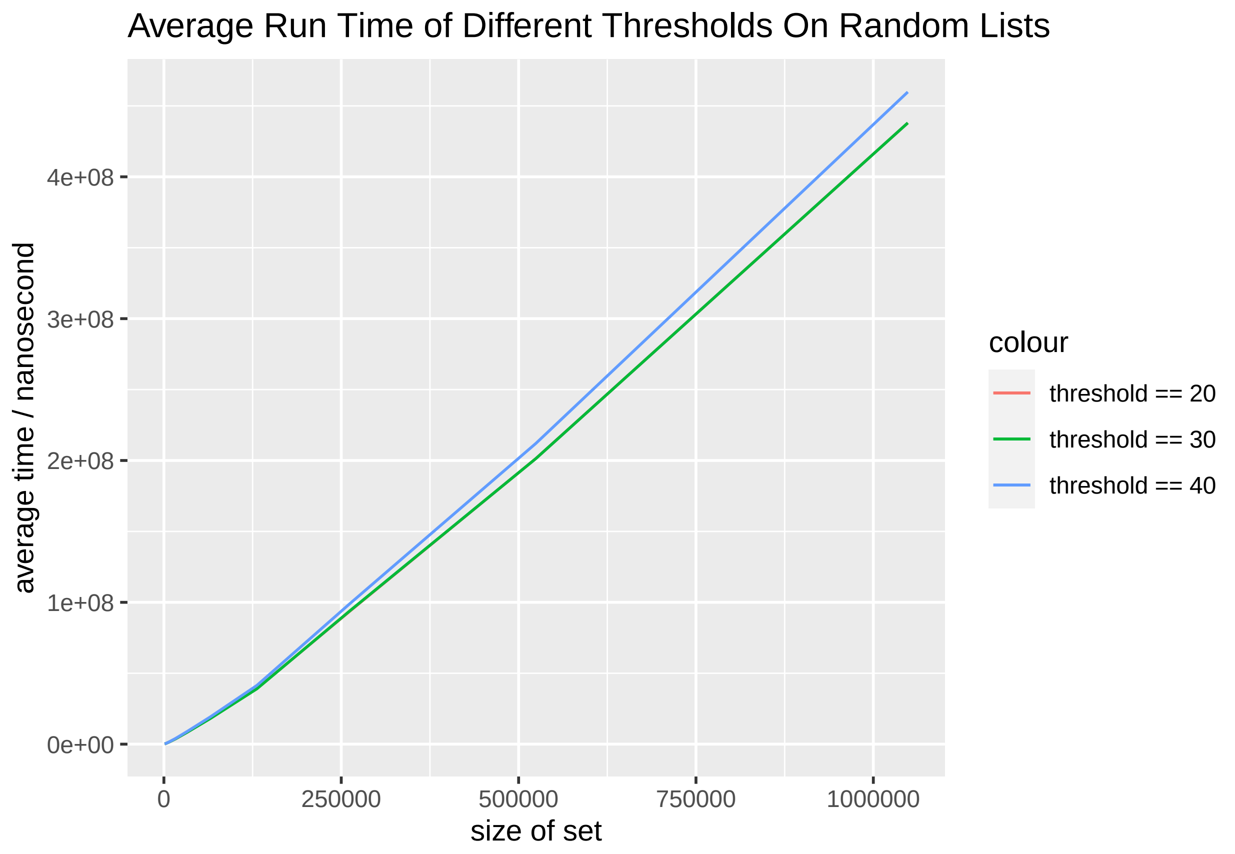
- It should take O(N logN) to mergesort the array list.
- Run the program and record the average time for `mergesort()` on 11 set sizes: 2^{10}, 2^{11}, 2^{12}… 2^{19}, 2^{20}.
- Set the threshold as 1 (full mergesort), 20, 30, 40, 50.
- Time `mergesort()` for 100 times for each set size and each threshold value, and take the average.
- Plot these times against the set size.

```
library(ggplot2)
size    <- c(1024, 2048, 4096, 8192, 16384, 32768, 65536, 131072, 262144,
             524288, 1048576)
avgtime_1 <- c(312702.03, 396769.15, 861541.32, 1883692.12, 4138983.75,
             9093584.2, 1.959862589E7, 4.197827383E7, 9.933893785E7,
             2.1443037631E8, 4.6525832293E8)
avgtime_20 <- c(186459.6, 341709.62, 755207.11, 1693997.88, 3763911.21,
             8397636.28, 1.826801625E7, 3.918377752E7, 9.373298667E7,
             2.0140159041E8, 4.380136717E8)
avgtime_30 <- c(175276.68, 329659.12, 743655.83, 1694994.96, 3752410.4,
             8376655.86, 1.810860712E7, 3.902084752E7, 9.401400879E7,
             2.013528958E8, 4.3801992545E8)
avgtime_40 <- c(196281.21, 357514.52, 807767.07, 1796253.32, 4011897.9,
             8949740.41, 1.925908375E7, 4.141884836E7, 9.908042919E7,
             2.1205283875E8, 4.5983821042E8)
avgtime_50 <- c(187207.08, 351055.81, 800228.68, 1790699.61, 4018087.02,
             8940900.36, 1.926321661E7, 4.139427668E7, 9.906480579E7,
             2.1202451748E8, 4.5983067496E8)
timeingdata <- data.frame(size, avgtime_1, avgtime_20, avgtime_30,
                          avgtime_40, avgtime_50)
ggplot() + geom_line(data = timeingdata, aes(x = size, y = avgtime_1,
    colour = "full mergesort")) + geom_point() +
    geom_line(data = timeingdata, aes(x = size, y = avgtime_20,
    colour = "threshold == 20")) + geom_point() +
    geom_line(data = timeingdata, aes(x = size, y = avgtime_30,
    colour = "threshold == 30")) + geom_point() +
    geom_line(data = timeingdata, aes(x = size, y = avgtime_40,
    colour = "threshold == 40")) + geom_point() +
    geom_line(data = timeingdata, aes(x = size, y = avgtime_50,
    colour = "threshold == 50")) + geom_point() + labs(
    x = "size of set",
    y = "average time / nanosecond",
    title = "Average Run Time of Different Thresholds On Random Lists")
```



- For a full mergesort (never switching to insertion sort), we have the slowest sort speed. `Threshold == 50` does not seem like a good choice either.
- Well, the lines of `Threshold == 20`, `Threshold == 30` and `Threshold == 40` overlap with each other. Let's exclude other cases and zoom in to see the differences.

```r
library(ggplot2)
size    <- c(1024, 2048, 4096, 8192, 16384, 32768, 65536, 131072, 262144,
             524288, 1048576)
avgtime_20 <- c(186459.6, 341709.62, 755207.11, 1693997.88, 3763911.21,
             8397636.28, 1.826801625E7, 3.918377752E7, 9.373298667E7,
             2.0140159041E8, 4.380136717E8)
avgtime_30 <- c(175276.68, 329659.12, 743655.83, 1694994.96, 3752410.4,
             8376655.86, 1.810860712E7, 3.902084752E7, 9.401400879E7,
             2.013528958E8, 4.3801992545E8)
avgtime_40 <- c(196281.21, 357514.52, 807767.07, 1796253.32, 4011897.9,
             8949740.41, 1.925908375E7, 4.141884836E7, 9.908042919E7,
             2.1205283875E8, 4.5983821042E8)
timeingdata <- data.frame(size, avgtime_20, avgtime_30, avgtime_40)
ggplot() + geom_line(data = timeingdata, aes(x = size, y = avgtime_20,
    colour = "threshold == 20")) + geom_point() +
    geom_line(data = timeingdata, aes(x = size, y = avgtime_30,
    colour = "threshold == 30")) + geom_point() +
    geom_line(data = timeingdata, aes(x = size, y = avgtime_40,
    colour = "threshold == 40")) + geom_point() + labs(
    x = "size of set",
    y = "average time / nanosecond",
    title = "Average Run Time of Different Thresholds On Random Lists")
```

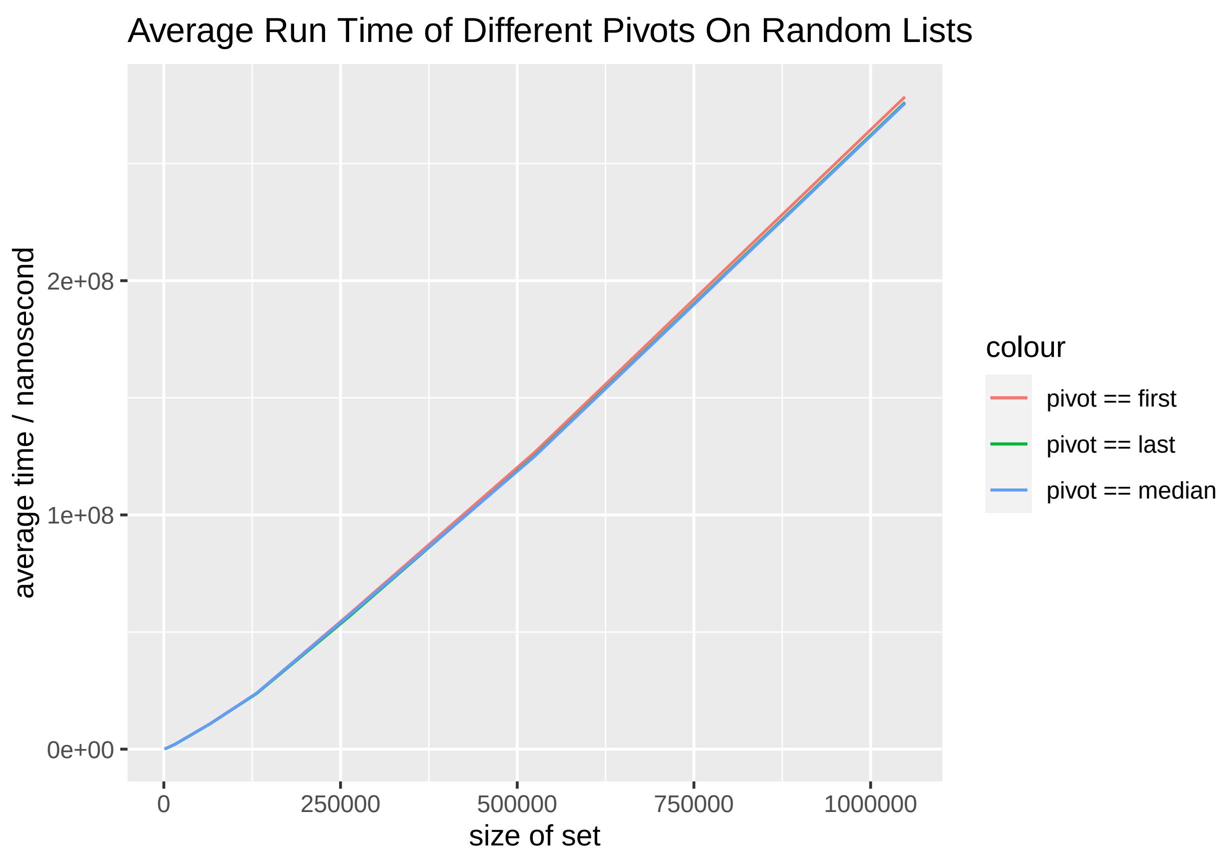


- `Threshold == 20` and `Threshold == 30` overlap again. But it's fairly clear that `Threshold == 40` is out of the game. Therefore, the best threshold should between 20 and 30.

3. Quicksort Pivot Experiment: Determine the best pivot-choosing strategy for quicksort. (As in #2, use large list sizes, the same set of permuted-order lists for each strategy, and the timing techniques demonstrated before.) Plot the running times of your quicksort for three different pivot-choosing strategies on permuted-order lists (one line for each strategy).

- It should take O(N logN) to quicksort the array list.
- Run the program and record the average time for `quicksort()` on 11 set sizes: 2^{10}, 2^{11}, 2^{12}... 2^{19}, 2^{20}.
- Set the pivot strategy as the first element, the last element, and the median of three (first, last, middle).
- Time `quicksort()` for 100 times for each set size and each pivot strategy, and take the average.
- Plot these times against the set size.

```r
library(ggplot2)
size    <- c(1024, 2048, 4096, 8192, 16384, 32768, 65536, 131072, 262144,
             524288, 1048576)
avgtime_first <- c(180962.54, 227091.26, 449151.21, 1010127.45, 2193714.64,
             5121959.2, 1.090826759E7, 2.385734706E7, 5.759937917E7,
             1.2663214458E8, 2.7849013168E8)
avgtime_last <- c(166683.76, 221047.97, 441597.88, 1003401.31, 2187239.6,
             5049110.44, 1.082289491E7, 2.368753417E7, 5.657234917E7,
             1.2514324586E8, 2.7612353507E8)
avgtime_median <- c(157866.7, 233812.91, 438650.07, 996833.4, 2182417.07,
             5090274.98, 1.08179262E7, 2.386381586E7, 5.716442377E7,
             1.2486477619E8, 2.7571253544E8)
timeingdata <- data.frame(size, avgtime_first, avgtime_last, avgtime_median)
ggplot() + geom_line(data = timeingdata, aes(x = size, y = avgtime_first,
    colour = "pivot == first")) + geom_point() +
    geom_line(data = timeingdata, aes(x = size, y = avgtime_last,
    colour = "pivot == last")) + geom_point() +
    geom_line(data = timeingdata, aes(x = size, y = avgtime_median,
    colour = "pivot == median")) + geom_point() + labs(
    x = "size of set",
    y = "average time / nanosecond",
    title = "Average Run Time of Different Pivots On Random Lists")
```

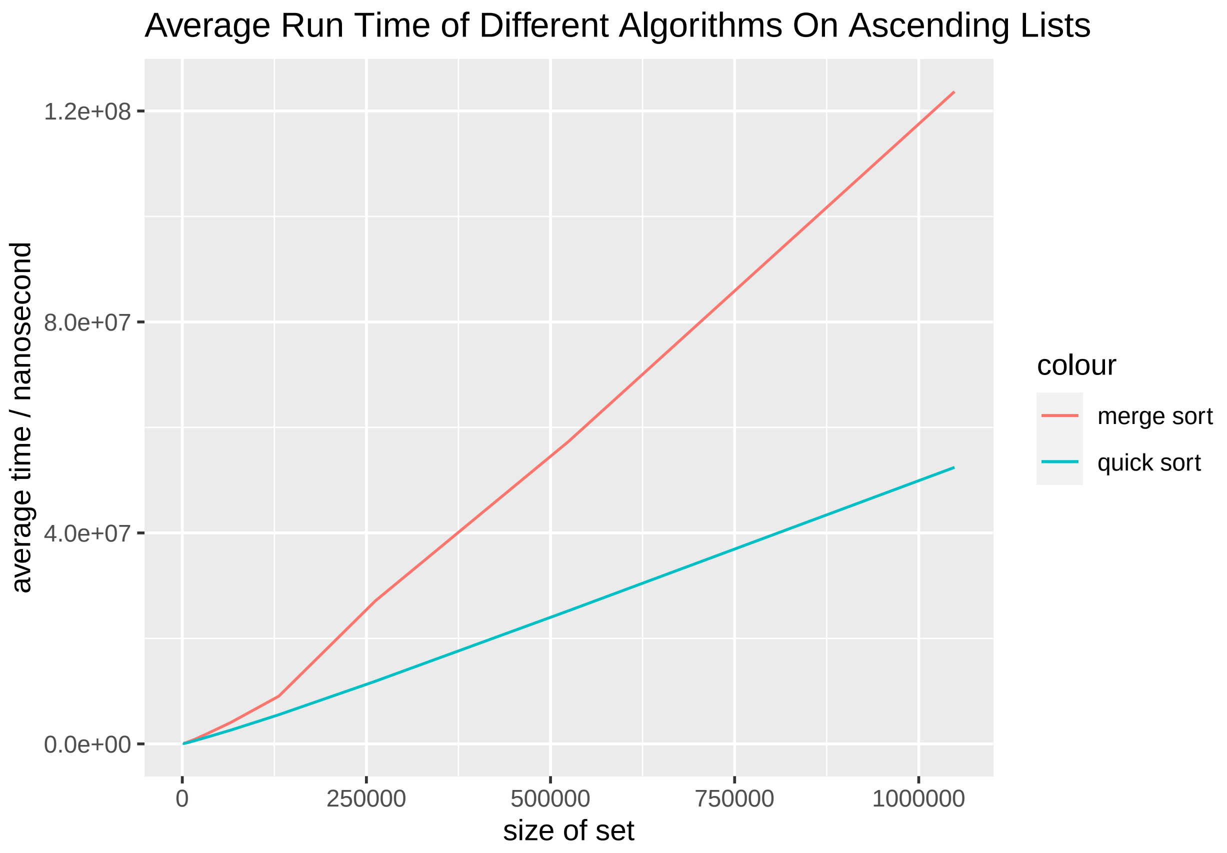


- I do apologize for them overlapping again. Maybe R Markdown is not a good choice for generating graphs, or maybe different pivots work roughly the same on random lists. But zoom in the graph or take a glance at the data itself - we can say median of three is the best pivot strategy.

4. Mergesort vs. Quicksort Experiment: Determine the best sorting algorithm for each of the three categories of lists (best-, average-, and worst-case). For the mergesort, use the threshold value that you determined to be the best. For the quicksort, use the pivot-choosing strategy that you determined to be the best. Note that the best pivot strategy on permuted lists may lead to O(N^2) performance on best/worst case lists. If this is the case, use a different pivot for this part. As in #2, use large list sizes, the same list sizes for each category and sort, and the timing techniques demonstrated before. Plot the running times of your sorts for the three categories of lists. You may plot all six lines at once or create three plots (one for each category of lists).

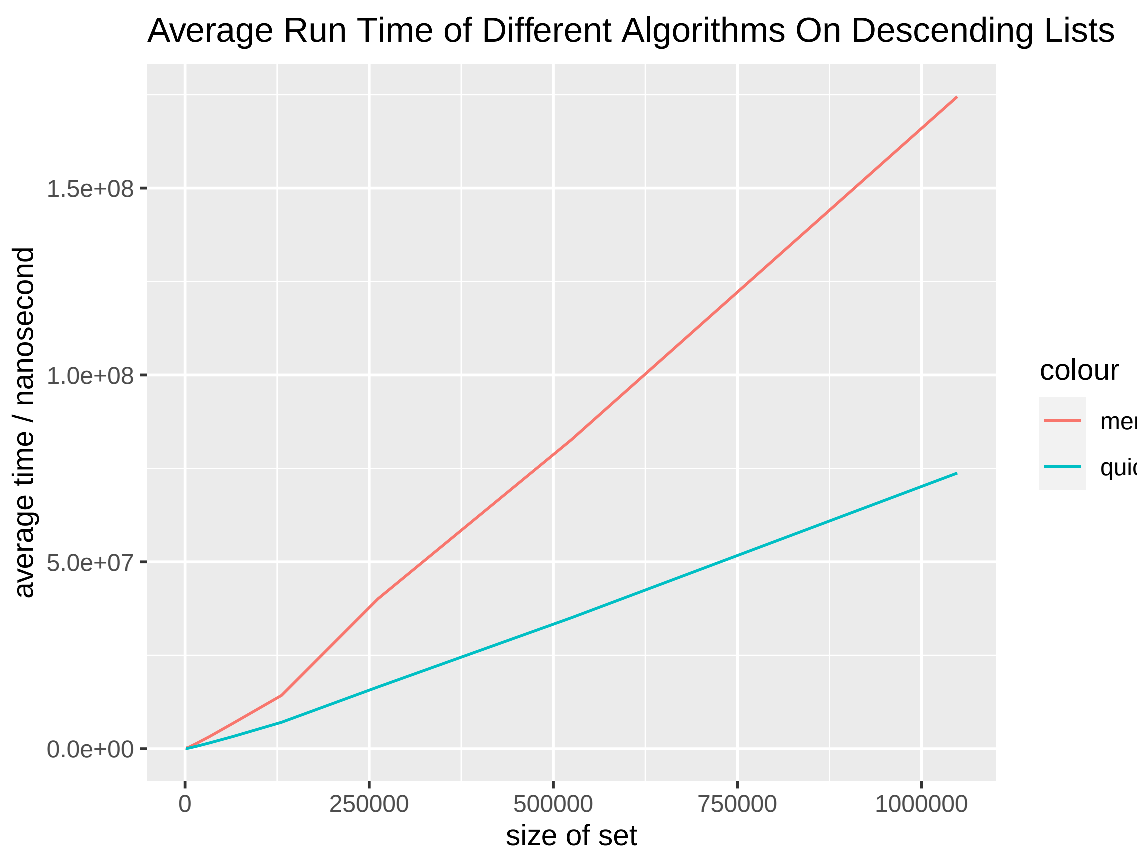
- For the best case, set the threshold as 30 and the pivot as median of three.

```
library(ggplot2)
size    <- c(1024, 2048, 4096, 8192, 16384, 32768, 65536, 131072, 262144,
             524288, 1048576)
avgtime_merge <- c(218015, 130626, 169868, 394005, 832479, 1873035,
             4034988, 9044164, 27110080, 57314365, 123674684)
avgtime_quick <- c(110316, 64469, 133595, 277145, 590060, 1246258,
             2599951, 5531112, 11880008, 25233195, 52425443)
timeingdata <- data.frame(size, avgtime_merge, avgtime_quick)
ggplot() + geom_line(data = timeingdata, aes(x = size, y = avgtime_merge,
    colour = "merge sort")) + geom_point() +
    geom_line(data = timeingdata, aes(x = size, y = avgtime_quick,
    colour = "quick sort")) + geom_point() + labs(
    x = "size of set",
    y = "average time / nanosecond",
    title = "Average Run Time of Different Algorithms On Ascending Lists")
```



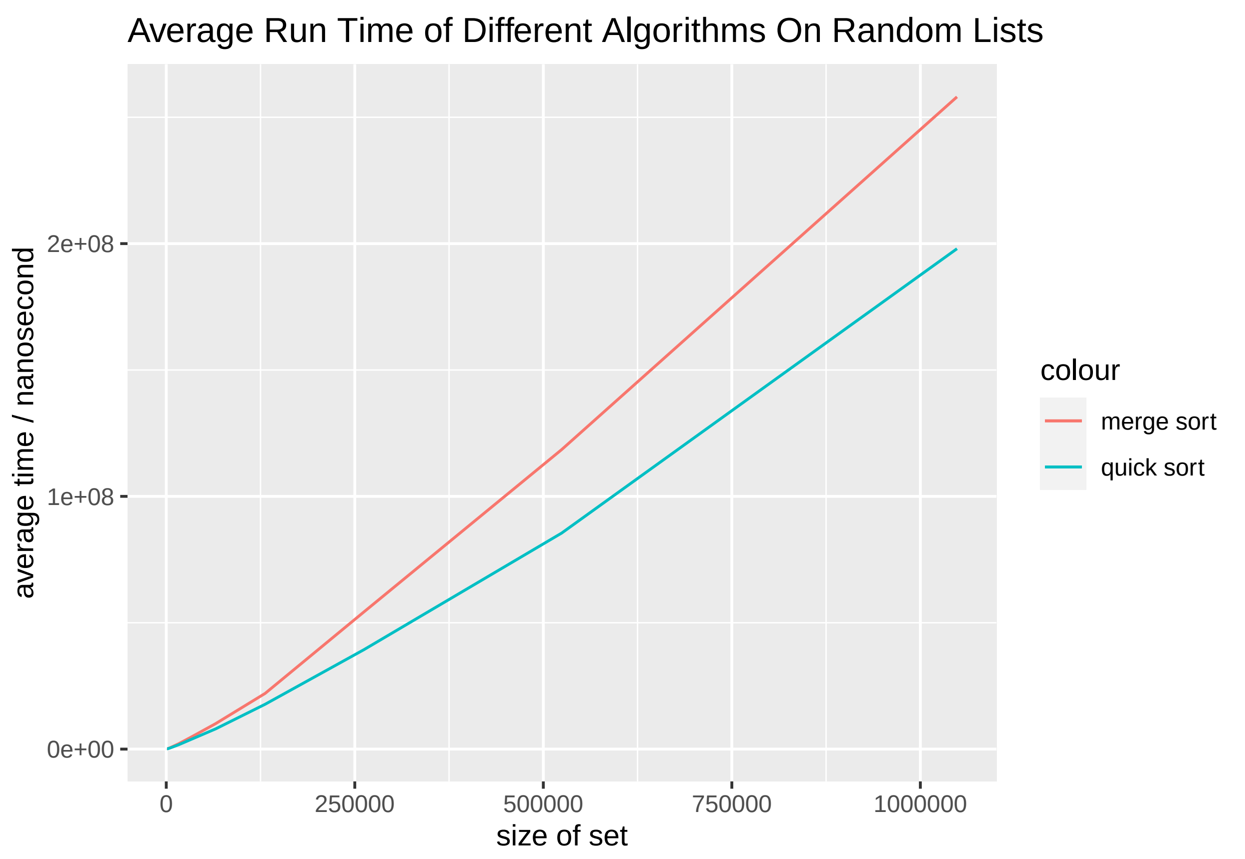
- Quick sort wins this turn.
- For the so-called worst case, set the threshold as 30 and the pivot as median of three.

```
    library(ggplot2)
    size    <- c(1024, 2048, 4096, 8192, 16384, 32768, 65536, 131072, 262144,
                 524288, 1048576)
    avgtime_merge <- c(274863, 217064, 328352, 722887, 1530134, 3220844,
                 6871584, 14298520, 40150209, 82627375, 174472783)
    avgtime_quick <- c(120106, 64221, 149244, 329562, 715769, 1549540,
                 3311641, 7099313, 16551887, 35038971, 73761289)
    timeingdata <- data.frame(size, avgtime_merge, avgtime_quick)
    ggplot() + geom_line(data = timeingdata, aes(x = size, y = avgtime_merge,
        colour = "merge sort")) + geom_point() +
        geom_line(data = timeingdata, aes(x = size, y = avgtime_quick,
        colour = "quick sort")) + geom_point() + labs(
        x = "size of set",
        y = "average time / nanosecond",
        title = "Average Run Time of Different Algorithms On Descending Lists")
```



- Again, quick sort wins this turn.
- For the average case, set the threshold as 30 and the pivot as median of three.

```
library(ggplot2)
size    <- c(1024, 2048, 4096, 8192, 16384, 32768, 65536, 131072, 262144,
             524288, 1048576)
avgtime_merge <- c(285331, 220681, 415529, 968964, 2107698, 4706981,
             10090095, 22031027, 54286768, 118486867, 258073055)
avgtime_quick <- c(122650, 143538, 344854, 795043, 1720882, 3822487,
             8013602, 17734078, 39346052, 85464453, 197979874)
timeingdata <- data.frame(size, avgtime_merge, avgtime_quick)
ggplot() + geom_line(data = timeingdata, aes(x = size, y = avgtime_merge,
    colour = "merge sort")) + geom_point() +
    geom_line(data = timeingdata, aes(x = size, y = avgtime_quick,
    colour = "quick sort")) + geom_point() + labs(
    x = "size of set",
    y = "average time / nanosecond",
    title = "Average Run Time of Different Algorithms On Random Lists")
```



- Triple kill! Quick sort performs better in all the three cases.

5. Do the actual running times of your sorting methods exhibit the growth rates you expected to see? Why or why not? Please be thorough in this explanation.

- Yes. For ascending/descending sorted lists, the growth rates of both mergesort and quicksort are fitted as linear. But for random lists, the growth rates of both mergesort and quicksort approximately follow O(N logN).
- Besides, the sorting speed of mergesort is slower but more stable compared to quicksort.