**INFO6205 Summer 2022 Team Project**

**Report for Quick Sort and Shell Sort**

Project Group 6

Parth Kataria – 002964611

**Quick Sort Analysis**

Part 1 – Code

The code has been migrated from [rchillyard](https://github.com/rchillyard)/[**INFO6205**](https://github.com/rchillyard/INFO6205) to [INFO-6205](https://github.com/INFO-6205)/**[HuskySort](https://github.com/INFO-6205/HuskySort)** with modifications explained below. The quick sort basic code with test cases for the same moved to HuskySort repository to take advantage of the better benchmarking tools.

1. QuickSort\_standard.java



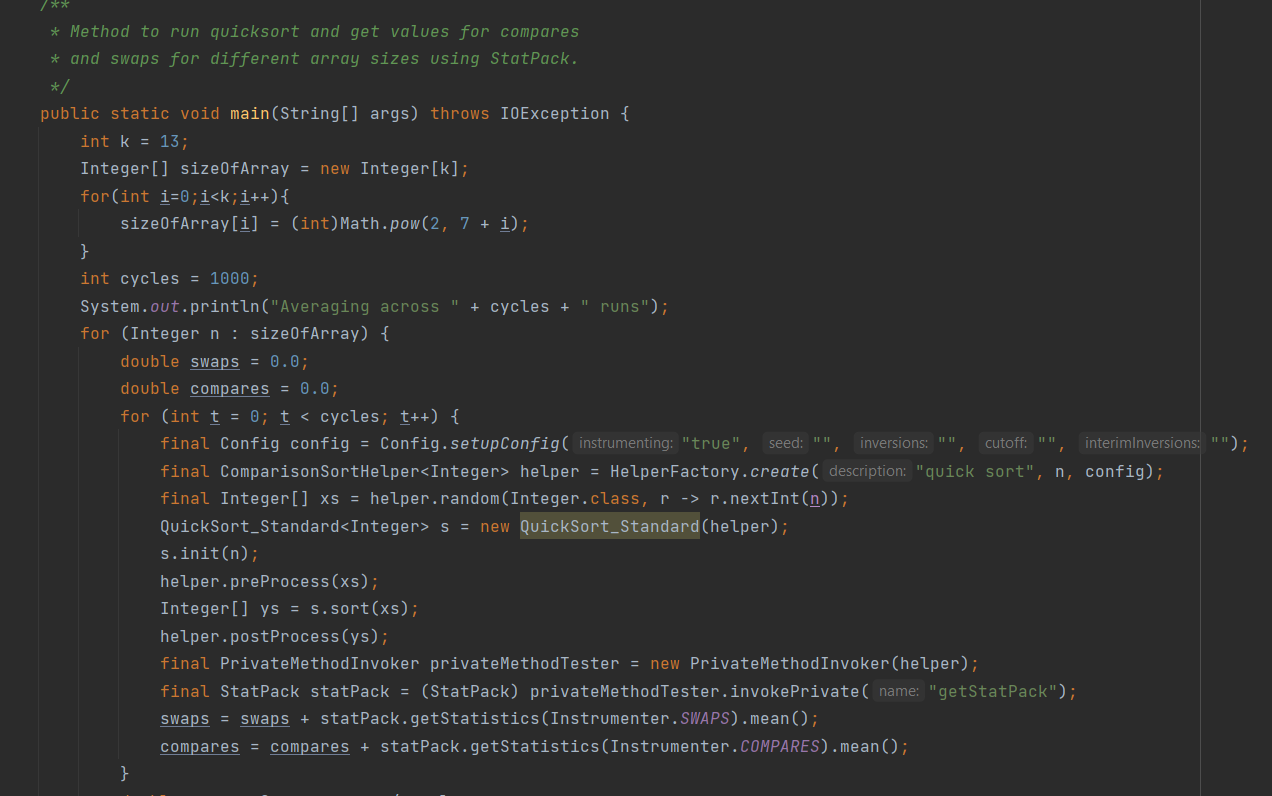
1. QuickSort\_StandardTest.java



1. Config.java



Main function to run the quicksort basic function and get the average number of swaps and compares using StatPack



A screenshot of a computer

Description automatically generated

Changes in config file

Text

Description automatically generated

Text

Description automatically generated

Changes in quicksort file to stop function to switch to insertion sort and use quicksort all the way through

Text

Description automatically generated

Part 2 - Test Cases for quick sort

A screenshot of a computer

Description automatically generated with medium confidence

Results of Main function

A picture containing text

Description automatically generated

Part 3 - Observations

As seen in the values below, the number of compares in not exactly 6 times the number of swaps. However, as the array size is increasing, the ratio of compares by swaps is also heading towards six. It can be considered that when the array size reaches a high value and number of runs increase, the compares by swap ratio would be six. Due to limitation of the current available system and processing power of the device, we are not running the benchmarking for higher array size.

Table

Description automatically generated

Chart, line chart

Description automatically generated

It can also be proved theoretically that the number of compares is 1.39nlg(n) as seen below

Text, letter

Description automatically generated

Table

Description automatically generated with medium confidence

However, to prove that swaps is 0.23nlg(n) which is one sixth of compares, we use the practical data we have got from benchmarking

It can be seen in the observations below that the swaps is 0.23nlg(n) as the ratio of theoretical compares by practical swaps is approximately sixTable

Description automatically generated

Please check the excel file for detailed observations



Conclusion:

As we see in our observations, the theoretical value of number of compares for expected case of quicksort is 1.39nlog(n). In the below graph, we also see that our practical value comes close to it.

Chart, line chart

Description automatically generated

If we see the observations for Expected Compares/Practical swaps below, the values are almost six.

Hence, I agree that the number of swaps in quicksort is 1/6 times the number of comparisons.

References

1. [rchillyard](https://github.com/rchillyard)/[**INFO6205**](https://github.com/rchillyard/INFO6205) - <https://github.com/rchillyard/INFO6205>
2. [INFO-6205](https://github.com/INFO-6205)/[**HuskySort**](https://github.com/INFO-6205/HuskySort) - <https://github.com/INFO-6205/HuskySort>
3. Performance of Quicksort - http://homepages.math.uic.edu/~leon/cs-mcs401-r07/handouts/quicksort-continued.pdf

**Shell Sort Analysis**

Part 1 – Code

The code has been migrated from [rchillyard](https://github.com/rchillyard)/[**INFO6205**](https://github.com/rchillyard/INFO6205) to [INFO-6205](https://github.com/INFO-6205)/**[HuskySort](https://github.com/INFO-6205/HuskySort)** with modifications explained below. The shell sort code with test cases for the same moved to HuskySort repository to take advantage of the better benchmarking tools.

1. ShellSort.java



1. QuickSort\_StandardTest.java



Main function to run the ShellSort function and get the average number compares using StatPack

Using Knuth’s sequence in shellsort

The main function has three cases for best case scenario, average case scenario and worst case scenario

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated with medium confidence

Text

Description automatically generated

A picture containing graphical user interface

Description automatically generated

Part 2 – Test Case

A screenshot of a computer

Description automatically generated with medium confidence

Output for Main function

Text

Description automatically generated

Part 3 – Observations

1. When shell sorting a sorted array, we can see the number of comparisons in the below readings

Table

Description automatically generated

The number of comparisons is logarithmic as seen in the observations with the base of log as 3

Please note that these readings are for Knuth’s gap sequence.

The base of the logs can also be derived theoretically

The best case, like insertion sort, is when the array is already sorted. The number of comparisons for each of the increment-based insertion sorts is the length of the array. Therefore we can determine:

comparisons = n, for 1 sort with elements 1-apart (last step)

+ 4 \* n/4, for 4 sorts with elements 4-apart (next-to-last step)

+ 13 \* n/13, for 13 sorts with elements 13-apart

+ ...

Each term is n. The question is how many terms are there?

h < n

h = h\*3 + 1

h = h/3

As h is divided by 3 every time to get a new value, the total terms is the value of k such that

3^k < n

Graph to compare practical and expected values

Chart, line chart

Description automatically generated

Hence, we can derive that the base of the log will be 3.

1. Expression for average number of comparisons for Knuth’s sequence

Table

Description automatically generated

If we observe the practical values from benchmarking, the increase in the value is proportional to N^1.25. To be specific it comes to around 2.4\*(N)^1.25. Please see the graph below for comparison of expected and practical values

Chart, line chart

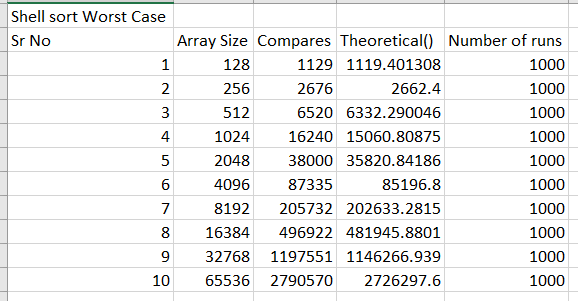
Description automatically generated

Hence an approximate expression for the average number of compares for shell-sort for Knuth’s sequence would be 2.4\*(N)^1.25. Due to limitation of the current available system and processing power of the device, we are not running the benchmarking for higher array size.

1. Expression for the worst – case number of comparisons for the Knuth’s gap sequence

We could not determine any input that would be worst case for shell sort as the purpose of using shell sort is that there won’t be any worst case. Shell sort even works better for an inverted array. The worst case would be using a bad gap sequence. For getting worst case for Knuth’s gap sequence, we pass input as random array and out of 1000 runs, we use the result of the array which has maximum comparisons.

Please see the observations below



After observing the practical values, we can see that the compares are directly proportional to (N)^1.25. Specifically, it comes close to 2.6\*(N)^1.25. Due to limitation of the current available system and processing power of the device, we are not running the benchmarking for higher array size. Please see the graph below for comparison of expected and practical values

Chart, line chart

Description automatically generated

Hence we can derive that comparisons for worst case would be 2.6\*(N)^1.25

Please check the excel for details of the observations



Part 4 – Conclusion

As we saw in our observations, we can conclude that

1. When shell-sorting a sorted array, the number of comparisons is logarithmic. The base of the logs for Knuth’s gap sequence is 3.
2. The expression for the average number of comparisons for shell – sort using Knuth’s gap sequence is approximately 2.4\*(N)^1.25
3. The expression for the worst-case number of comparisons for Knuth’s gap sequence is approximately 2.6\*(N)^1.25

References

1. [rchillyard](https://github.com/rchillyard)/[**INFO6205**](https://github.com/rchillyard/INFO6205) - <https://github.com/rchillyard/INFO6205>
2. [INFO-6205](https://github.com/INFO-6205)/[**HuskySort**](https://github.com/INFO-6205/HuskySort) - <https://github.com/INFO-6205/HuskySort>
3. Shellsort & Algorithmic Comparisons - https://www.cs.wcupa.edu/rkline/ds/shell-comparison.html