Ben Sottile

4/18/2021

Sorting Assignment- Quick and Merge – Write Up Bellow

Merge and Quick Sort

Implement the Merge and Quick Sort algorithms using the pseudo code on pages 107/109 and 115/116 respectively. Instrument each algorithm, counting comparisons, as follows:

- Merge Sort: one comparison for each time through the while loop within MergeLists
- Quick Sort: one comparison for each time through the for loop within PivotList

Test your code for correctness adding print statements as necessary

Monte Carlo Simulation

Merge Sort

- Run the program on data sets consisting of random integers in the range of 0 to 100000. Run data sizes of 10, 100, 1000, 10000, 100000, 1000000 and plot size vs. count using Microsoft Excel or some other plotting package. Note that you should seed the random number generator so that each test uses the same random sequence.

Quick Sort

- Set the pivot point to be the first location of the list and run the program on data sets consisting of random integers in the range of 0 to 100000. Run data sizes of 10, 100, 1000, 10000, 100000, 1000000 and plot size vs. count using Microsoft Excel or some other plotting package. Note that you should seed the random number generator so that each test uses the same random sequence.
- Set the pivot point to be the last location of the list and run the program on data sets consisting of random integers in the range of 0 to 100000. Run data sizes of 10, 100, 1000, 10000, 100000, 1000000 and plot size vs. count using Microsoft Excel or some other plotting package. Note that you should seed the random number generator with the same value used above so that each test uses the same random sequence.
- Set the pivot point to be a random location in the list and run the program on data sets consisting of random integers in the range of 0 to 100000. Run data sizes of 10, 100, 1000, 10000, 100000, 1000000 and plot size vs. count using Microsoft Excel or some other plotting package. Note that you should seed the random number generator so that each test uses the same random sequence.

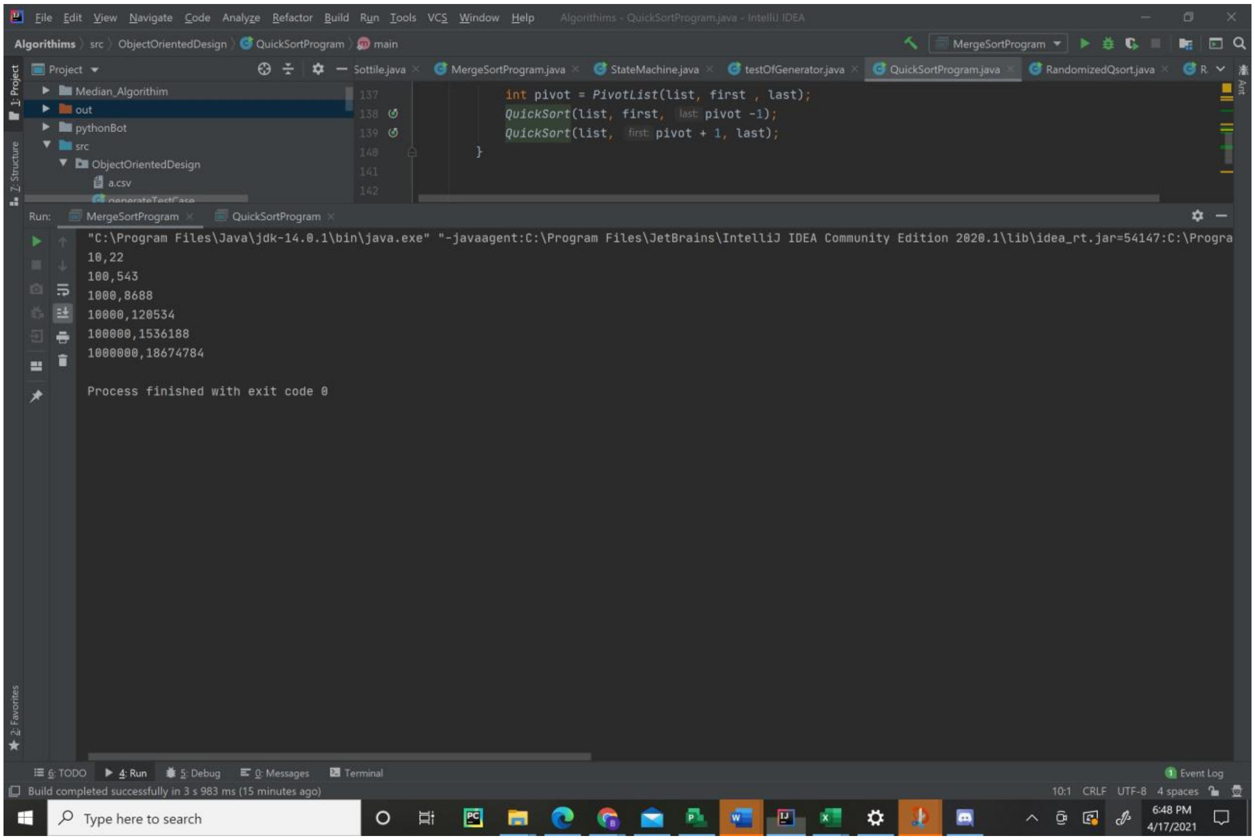
- Set the pivot point to be the first number in the list and run the program on data sets consisting of in sequence (smallest to largest) of 0 to 100000 for a list of 100000. Run the program and report the number of comparisons.
- Set the pivot point to be the first number in the list and run the program on data sets consisting of in sequence (largest to smallest) of 100000 to 0 for a list of 100000. Run the program and report the number of comparisons.

Deliverables

- Source code
- A reflection document including
    - Plot size vs. count for both sort algorithms for various sized lists of uniformly distributed random numbers (two separate plots)
        - Discuss the effects the different pivot locations have on the results
    - Counts for ordered data lists in Quick Sort algorithm
    - Essay describing successes and difficulties

Reflection document must be a PDF file. Do not submit documents of type MSWord, Pages, OpenOffice, etc.

Merge Sort

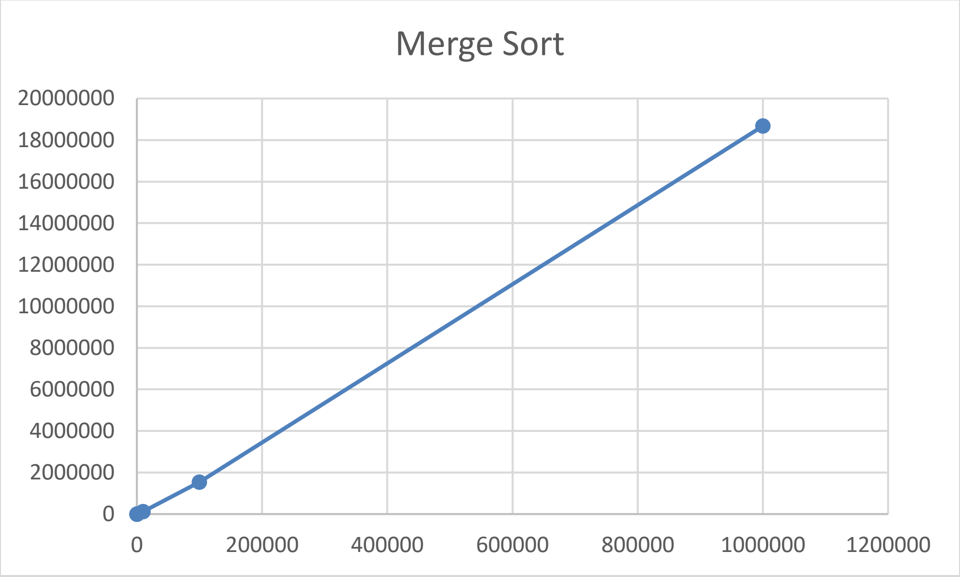


```
            int pivot = PivotList(list, first , last);
            QuickSort(list, first, pivot -1);
            QuickSort(list, pivot + 1, last);
        }
```

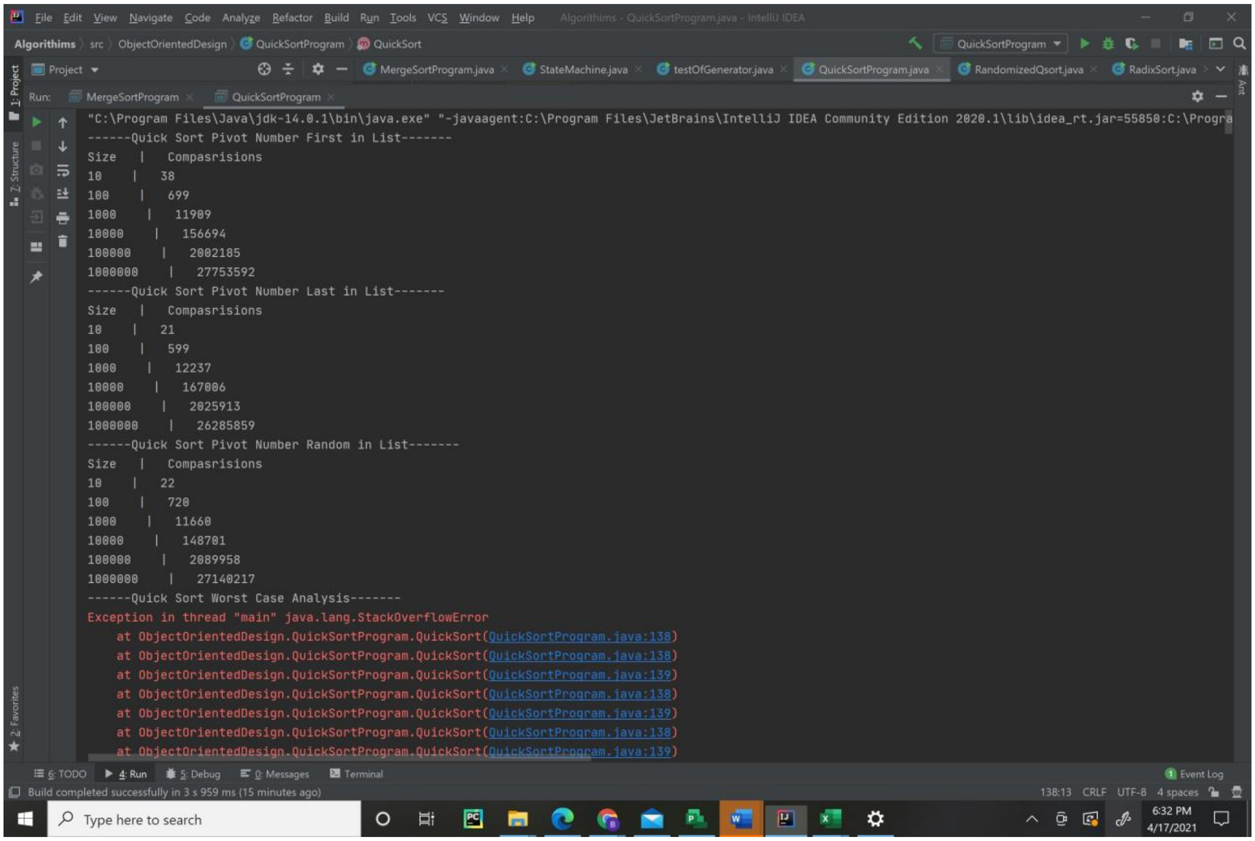
```
"C:\Program Files\Java\jdk-14.0.1\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2020.1\lib\idea_rt.jar=54147:C:\Progra
10,22
100,543
1000,8688
10000,120534
100000,1536188
1000000,18674784

Process finished with exit code 0
```

Build completed successfully in 3 s 983 ms (15 minutes ago)

10	22
100	543
1000	8688
10000	120534
100000	1536188
1000000	18674784



Quicksort



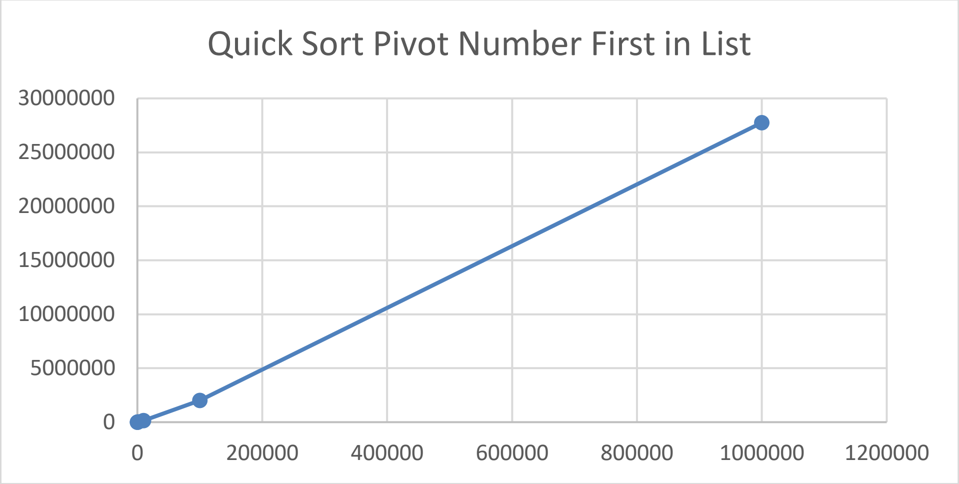
```
"C:\Program Files\Java\jdk-14.0.1\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2020.1\lib\idea_rt.jar=55850:C:\Progra
------Quick Sort Pivot Number First in List-------
Size   |   Compasrisions
10     |   38
100    |   699
1000    |   11909
10000    |   156694
100000    |   2002185
1000000    |   27753592
------Quick Sort Pivot Number Last in List-------
Size   |   Compasrisions
10     |   21
100    |   599
1000    |   12237
10000    |   167006
100000    |   2025913
1000000    |   26285859
------Quick Sort Pivot Number Random in List-------
Size   |   Compasrisions
10     |   22
100    |   720
1000    |   11660
10000    |   148701
100000    |   2089958
1000000    |   27140217
------Quick Sort Worst Case Analysis-------
Exception in thread "main" java.lang.StackOverflowError
    at ObjectOrientedDesign.QuickSortProgram.QuickSort(QuickSortProgram.java:138)
    at ObjectOrientedDesign.QuickSortProgram.QuickSort(QuickSortProgram.java:138)
    at ObjectOrientedDesign.QuickSortProgram.QuickSort(QuickSortProgram.java:139)
    at ObjectOrientedDesign.QuickSortProgram.QuickSort(QuickSortProgram.java:138)
    at ObjectOrientedDesign.QuickSortProgram.QuickSort(QuickSortProgram.java:139)
    at ObjectOrientedDesign.QuickSortProgram.QuickSort(QuickSortProgram.java:138)
    at ObjectOrientedDesign.QuickSortProgram.QuickSort(QuickSortProgram.java:139)
```

Build completed successfully in 3 s 959 ms (15 minutes ago)

Quick Sort Pivot Number First in List

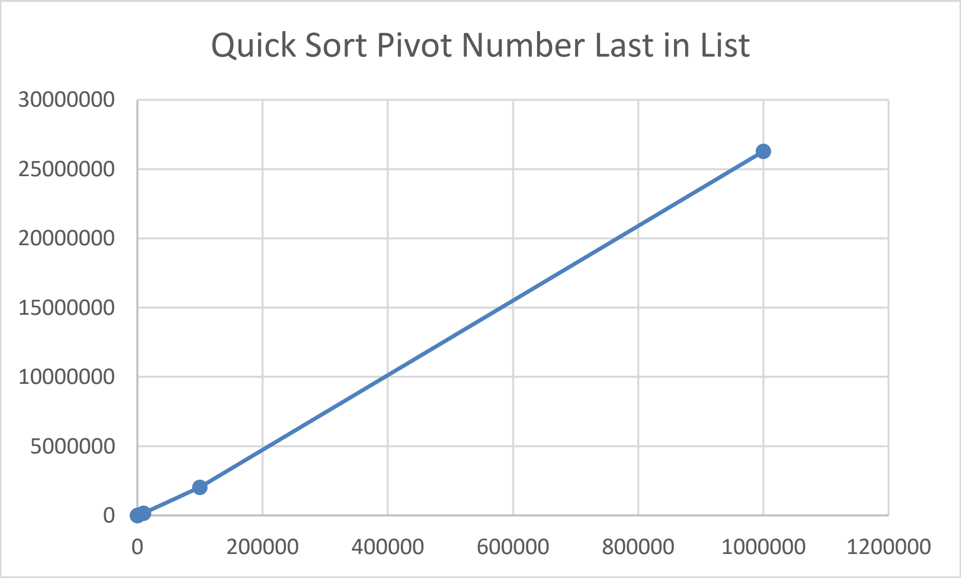
10	38
100	699
1,000	11909

10,000	156694
100,000	2002185
1,000,000	27753592



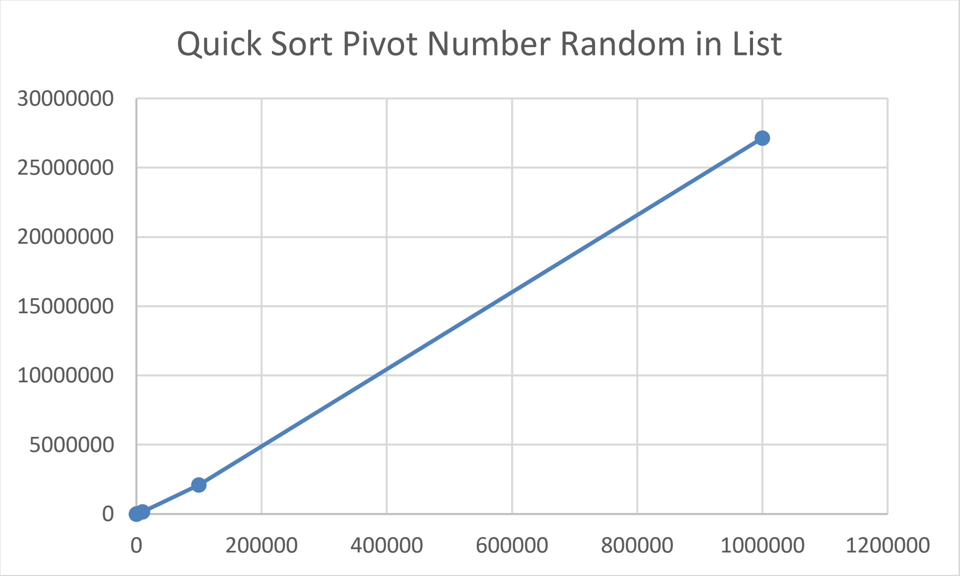
Quick Sort Pivot Number Last in List

10	21
100	599
1,000	12237
10,000	167006
100,000	2025913
1,000,000	26285859

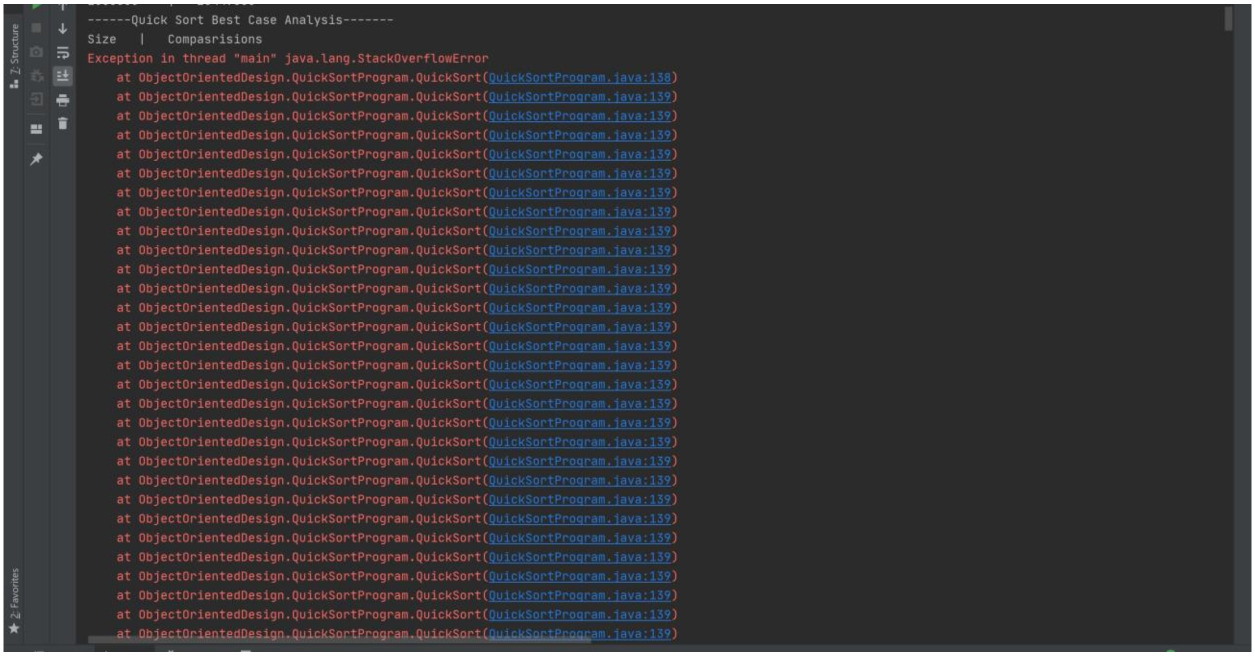


Quick Sort Pivot Number Random in List

10	22
100	720
1000	11660
10000	148701
100000	2089958
1000000	27140217

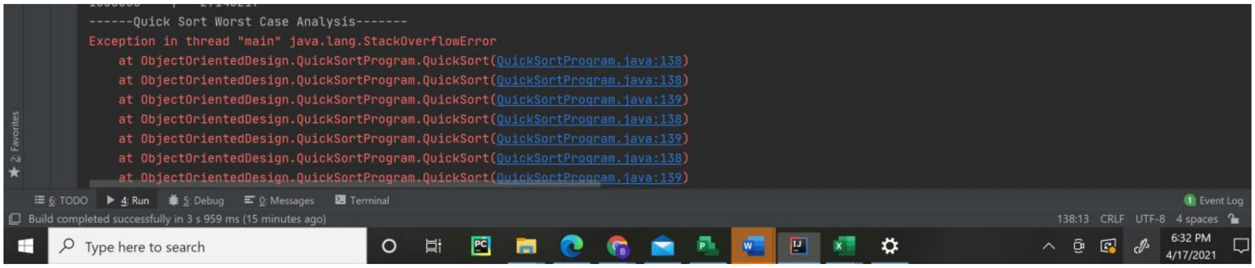


Best Case Analysis with 100,000 Numbers



```
------Quick Sort Best Case Analysis-------
Size    |   Compasrisions
Exception in thread "main" java.lang.StackOverflowError
    at ObjectOrientedDesign.QuickSortProgram.QuickSort(QuickSortProgram.java:138)
    at ObjectOrientedDesign.QuickSortProgram.QuickSort(QuickSortProgram.java:139)
    at ObjectOrientedDesign.QuickSortProgram.QuickSort(QuickSortProgram.java:139)
    at ObjectOrientedDesign.QuickSortProgram.QuickSort(QuickSortProgram.java:139)
    at ObjectOrientedDesign.QuickSortProgram.QuickSort(QuickSortProgram.java:139)
    at ObjectOrientedDesign.QuickSortProgram.QuickSort(QuickSortProgram.java:139)
    at ObjectOrientedDesign.QuickSortProgram.QuickSort(QuickSortProgram.java:139)
    at ObjectOrientedDesign.QuickSortProgram.QuickSort(QuickSortProgram.java:139)
    at ObjectOrientedDesign.QuickSortProgram.QuickSort(QuickSortProgram.java:139)
    at ObjectOrientedDesign.QuickSortProgram.QuickSort(QuickSortProgram.java:139)
    at ObjectOrientedDesign.QuickSortProgram.QuickSort(QuickSortProgram.java:139)
    at ObjectOrientedDesign.QuickSortProgram.QuickSort(QuickSortProgram.java:139)
    at ObjectOrientedDesign.QuickSortProgram.QuickSort(QuickSortProgram.java:139)
    at ObjectOrientedDesign.QuickSortProgram.QuickSort(QuickSortProgram.java:139)
    at ObjectOrientedDesign.QuickSortProgram.QuickSort(QuickSortProgram.java:139)
    at ObjectOrientedDesign.QuickSortProgram.QuickSort(QuickSortProgram.java:139)
    at ObjectOrientedDesign.QuickSortProgram.QuickSort(QuickSortProgram.java:139)
    at ObjectOrientedDesign.QuickSortProgram.QuickSort(QuickSortProgram.java:139)
    at ObjectOrientedDesign.QuickSortProgram.QuickSort(QuickSortProgram.java:139)
    at ObjectOrientedDesign.QuickSortProgram.QuickSort(QuickSortProgram.java:139)
    at ObjectOrientedDesign.QuickSortProgram.QuickSort(QuickSortProgram.java:139)
    at ObjectOrientedDesign.QuickSortProgram.QuickSort(QuickSortProgram.java:139)
    at ObjectOrientedDesign.QuickSortProgram.QuickSort(QuickSortProgram.java:139)
    at ObjectOrientedDesign.QuickSortProgram.QuickSort(QuickSortProgram.java:139)
    at ObjectOrientedDesign.QuickSortProgram.QuickSort(QuickSortProgram.java:139)
    at ObjectOrientedDesign.QuickSortProgram.QuickSort(QuickSortProgram.java:139)
    at ObjectOrientedDesign.QuickSortProgram.QuickSort(QuickSortProgram.java:139)
    at ObjectOrientedDesign.QuickSortProgram.QuickSort(QuickSortProgram.java:139)
    at ObjectOrientedDesign.QuickSortProgram.QuickSort(QuickSortProgram.java:139)
    at ObjectOrientedDesign.QuickSortProgram.QuickSort(QuickSortProgram.java:139)
```

Worst Case Analysis with 100,000 Numbers



```
------Quick Sort Worst Case Analysis-------
Exception in thread "main" java.lang.StackOverflowError
    at ObjectOrientedDesign.QuickSortProgram.QuickSort(QuickSortProgram.java:138)
    at ObjectOrientedDesign.QuickSortProgram.QuickSort(QuickSortProgram.java:138)
    at ObjectOrientedDesign.QuickSortProgram.QuickSort(QuickSortProgram.java:139)
    at ObjectOrientedDesign.QuickSortProgram.QuickSort(QuickSortProgram.java:138)
    at ObjectOrientedDesign.QuickSortProgram.QuickSort(QuickSortProgram.java:139)
    at ObjectOrientedDesign.QuickSortProgram.QuickSort(QuickSortProgram.java:138)
    at ObjectOrientedDesign.QuickSortProgram.QuickSort(QuickSortProgram.java:139)
```

Build completed successfully in 3 s 959 ms (15 minutes ago)

- Plot size vs. count for both sort algorithms for various sized lists of uniformly distributed random numbers (two separate plots)

  - Discuss the effects the different pivot locations have on the results

Different pivot locations have a small effect on the amount of comparisons all sort(done on the same seed) produced comparable results for each sort. I think the only way you really get a more pronounced effect is if the pivots you pick happens to be more effectively and split the data.

- Counts for ordered data lists in Quick Sort algorithm

I was unable to get information as I kept getting a stack overflow errors, which is likely attributed to the recursive nature of this algorithm. For I don’t understand why we are getting an error here though when it was able to sort 1,000,000 numbers. I guess it is the nature of the ordered list.

- Essay describing successes and difficulties

This assignment took quite a bit of time, which was surprising though I had sudo code. I started by re-reading the book, and just coding with the sudo-code and double checking. Both ironically took a bit to debug. I ended up doing more work the I needed by coding unneeded algorithms. I also didn't realize until later that the pivot point didn't technically need to be at its location during the sort. The value at the location not it being at the location maters. Not realizing this caused me to code the Quicksort with the pivot being the last number in the list quick Quick sort and the Quicksort with the pivot being a random number in the list quick Quicksort completely randomly. Not realizing I could pick a random or last number in the list, move it to the front and run it like a first number being the pivot quick sort. Also I screwed up my random number generator by not adding the first value to the generated number as the starting number, which caused a lot of pain. Eventually I coded the sorts. I cannot seem to solve the stackoverflow issue.