James Lockhart

CMSC 451

Professor Howard

October 10, 2016

**Project 2**

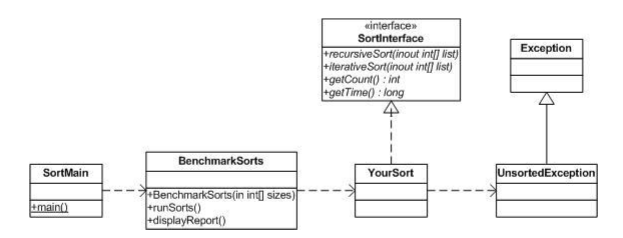
**Objective**

Project 2 involves an analysis of the results that you obtained in first project. You are to submit a paper, written with Microsoft Word, that discusses the results of your analysis.

This project is an extension of the material that I coded in project 1. Project 1 detail

Involve writing the code to perform the benchmarking of the insertion sort algorithm. My program will include both an iterative and recursive version of the algorithm. The program itself will identify some critical operation to count that reflects the overall performance. To have a better gauge of how the insertion sort methods, I will record the time it takes to complete each iteration. I will produce 50 data sets for each value of n, the size of the data set and average the result of those 50 runs. The exact same data will be used for the iterative and the recursive algorithms.

According to the specifications I had to code according to the following UML design:



**Design and Functionality**

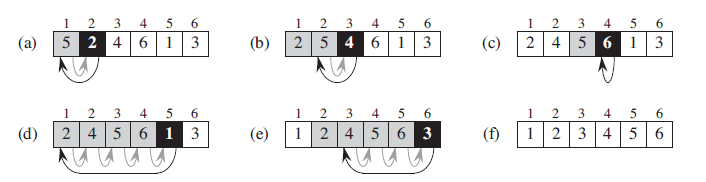
**Understanding Insertion Sort:**

When I first designed the program, I needed to understand how insertions sort methods worked. I went to the following website to grasp a better understanding: [www.java2novice.com](http://www.java2novice.com).

Through the site I learned that there are quite a few advantages to Insertion Sort:

1. Simple
2. Efficient for small data sets
3. Stable
4. Only requires a constant amount of O(1)

The basic premise will split the array into two arrays and iterate through it until only one remains and it’s sorted. The following display will show how it works.



Next some pseudo code will design the proper algorithm.

**for** i ← 1 **to** length(A)-1

j ← i

**while** j > 0 and A[j-1] > A[j]

**swap** A[j] and A[j-1]

j ← j - 1

**end while**

**end for**

**Analysis of algorithm:**

For the analysis we have to judge the running time of an algorithm on a particular input when each step is executed. We say that a statement takes *ci* steps to execute and executed *n* times contributes *cin* to the total running time of the algorithm. To compute the running time, T(*n*), we sum the products of the cost and times column. That is, the running time of the algorithm is the sum of running times for each statement executed.

So, we have:

T(*n*) = *c*1*n* + *c*2 (*n* − 1) + 0 (*n* − 1) + *c*4 (*n* − 1) + *c*5 ∑2 ≤ *j* ≤ *n* ( *tj* )+ *c*6 ∑2 ≤ *j* ≤ *n* (*tj* − 1) + *c*7 ∑2 ≤ *j* ≤ *n* (*tj* − 1) + *c*8 (*n* − 1)

**Best-Case:**

The best case for any scenario is if the array is already sorted. It will still have to use the while loops however because it still has to check to make sure each number is in the correct location. The following equation will sum up our best case scenario.

T(*n*) = *c*1*n* + *c*2 (*n* − 1) + *c*4 (*n* − 1) + *c*5 ∑2 ≤ *j* ≤ *n* (1) + *c*6 ∑2 ≤ *j* ≤ *n* (1 − 1) + *c*7 ∑2 ≤ *j* ≤ *n* (1 − 1) + *c*8 (*n* − 1)

T(*n*) = *c*1*n* + *c*2 (*n* − 1) + *c*4 (*n* − 1) + *c*5 (*n* − 1) + *c*8 (*n* − 1)

T(*n*) = *(c*1 + *c*2 + *c*4  + *c*5  + *c*8 ) *n* + (*c*2  + *c*4  + *c*5  + *c*8)

**Worst-Case:**

Our worst case will occur when we decide to sort it in reverse order. In the reverse order, we always find that *A*[*i*] is greater than the key in the while-loop test. So, we must compare each element *A*[*j*] with each element in the entire sorted sub array *A*[1 .. *j* − 1] and so *tj* = *j* for *j* = 2, 3, ..., *n*. Equivalently, we can say that since the while-loop exits because *i* reaches to 0, there is one additional test after (*j* − 1) tests.

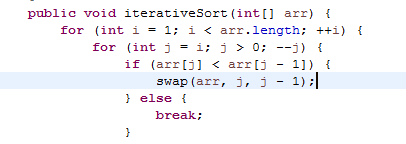
The following equation will have to be used to find the worst-case scenario:

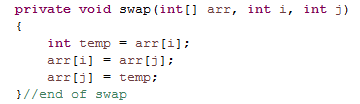
T(*n*) = *c*1*n* + *c*2 (*n* − 1) + *c*4  (*n* − 1) + *c*5 ∑2 ≤ *j* ≤ *n* [*n*(*n* +1)/2 + 1] + *c*6 ∑2 ≤ *j* ≤ *n* [*n*(*n* − 1)/2] + *c*7 ∑2 ≤ *j* ≤ *n* [*n*(*n* − 1)/2] + *c*8 (*n* − 1)

T(*n*) = (*c*5/2 + *c*6/2 + *c*7/2) *n*2 + (*c*1 + *c*2 + *c*4 + *c*5/2 − *c*6/2 − *c*7/2 + *c*8) *n* − (*c*2 + *c*4 + *c*5 + *c*8)

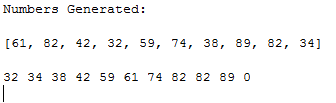
**Designing the algorithm for java:**

Taking the pseudo code, we can start to program the iteritave portion of the sort method.

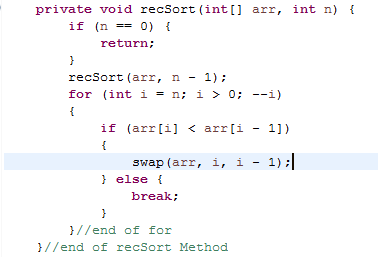




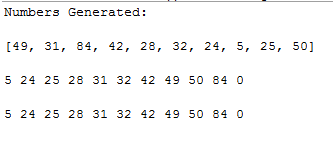
Next I need to test the code. According to the assignment requirements the values store in the array need to be randomly assigned. I used the Math.random function to achieve this. Next I pushed values into the method and the resulting output is shown below.



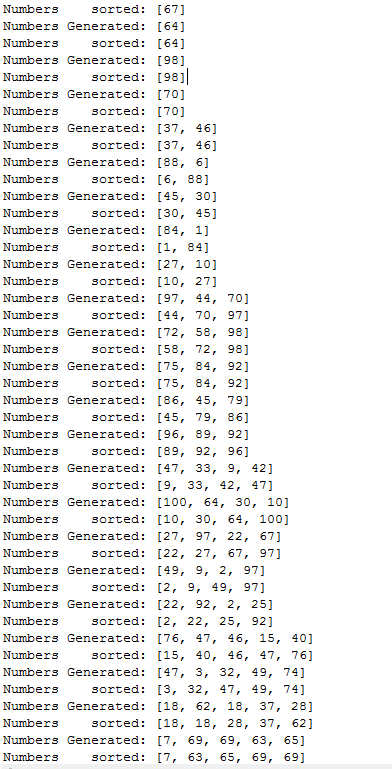
Since Insertion Sort using the iterative method is working, the next test is to use recursion to achieve the same results. I use the following code listed below in my following test.



After passing the values of the array to the method, the results from both methods are shown below



Next I will start to design according to the UML provided by the professor. Due to the methods that have to be implemented, I had to make small adjustments to my program. I first needed to run a small test that will increment the array size after running the specified number of tests. The test size will be small and only increment by 1 each time. Each test will be run five times before incrementing. The results are below.

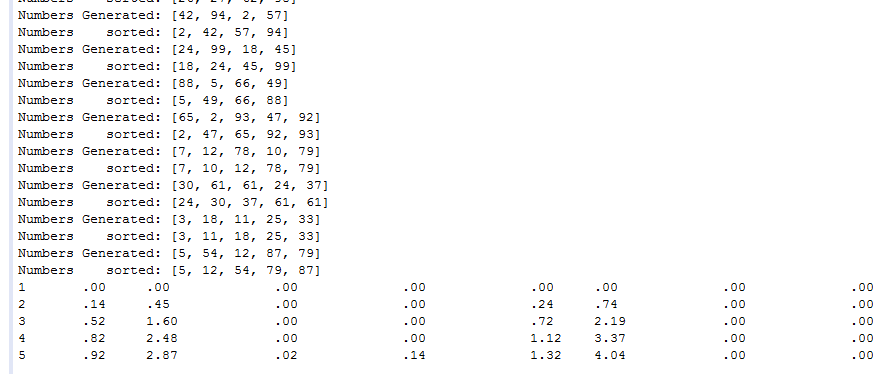


After the successful test run, I also had to test the recursive method implementation. The test provided the same results as listed above. Finally, I need to test the efficiency according the requirements of the project.

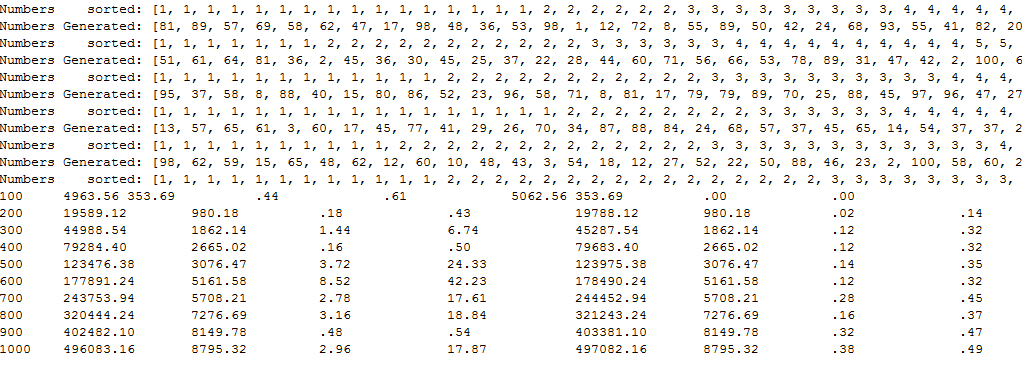
|  |  |  |
| --- | --- | --- |
| Data Set Size N | Iterative | Recursive |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Average Critical Operation n Count | Standard Deviation n of Count | Average Execution Time | Standard Deviation of Time | Average Critical Operation n Count | Standard Deviation n of Count | Average Execution Time | Standard Deviation of Time |

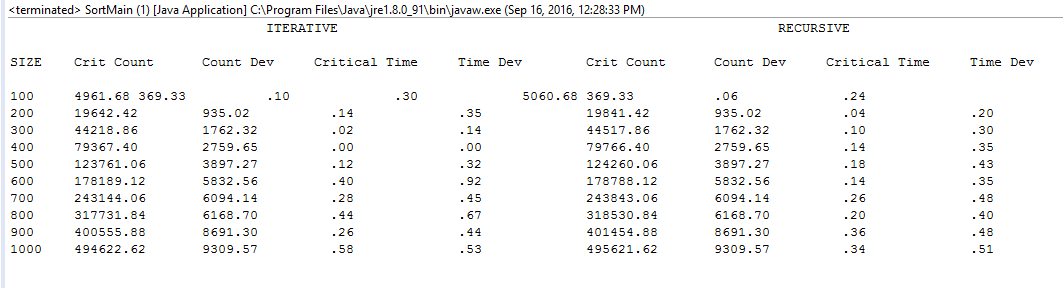
Using the same small data listed, as before, I will test with the above results.



At first glance it may appear that some of the results are not working but next I need to increase the testing size of not only the array but also the amount of test. After increasing to the specification of the project requirements, the program’s output is shown below.



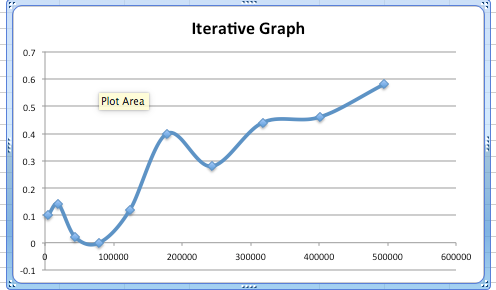
Since my results are achieving the intended results (although the first result never seems to align properly), I next decided to make the output more readable.

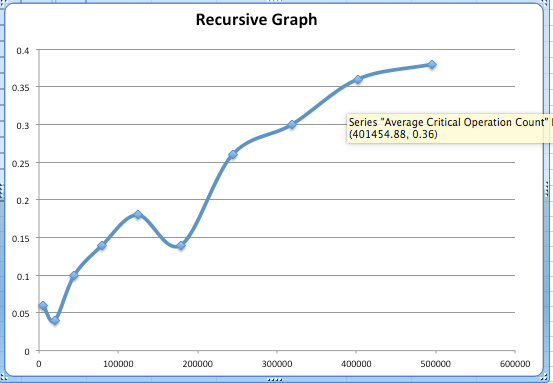


Below are the graphed results from the output.

**Size Recursive Iterative**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Average Critical Operation Count | Standard Deviation of Count | Average Execution Time | Standard Deviation Time | Average Critical Operation Count | Standard Deviation of Count | Average Execution Time | Standard Deviation Time |
| 100 | 4961.68 | 369.33 | .10 | .30 | 5060.68 | 369.33 | .06 | .24 |
| 200 | 19642.42 | 935.02 | .14 | .35 | 19841.42 | 935 | .04 | .20 |
| 300 | 44218.86 | 1762.32 | .02 | .14 | 44517.86 | 1762.32 | .10 | .30 |
| 400 | 79367.40 | 2759.65 | .00 | .00 | 79766.40 | 2759.65 | .14 | .35 |
| 500 | 123761.06 | 3897.27 | .12 | .32 | 12  4260.06 | 3897.27 | .18 | .43 |
| 600 | 178189.12 | 5832.56 | .40 | .92 | 178788.12 | 5832.56 | .14 | .35 |
| 700 | 243144.06 | 6094.14 | .28 | .45 | 243843.06 | 6094.14 | .26 | .48 |
| 800 | 317731.84 | 6168.70 | .44 | .67 | 318530.84 | 6168.70 | .20 | .40 |
| 900 | 400555.88 | 8691.30 | .26 | .44 | 401454.88 | 8691.30 | .36 | .48 |
| 1000 | 494622.62 | 9309.57 | .58 | .53 | 495621.62 | 9309.57 | .34 | .51 |





Interestingly it would seem like the tests would take small dips in the speed around the same intervals. Recursive method wins by a landslide though. As the critical operations increase, recursive seems to handle the operations at a better and faster pace. Since the big O follows O(1), it seemed to run on pace.

**Lessons Learned:** I found both of these projects to be very time consuming and filled with a lot of trial and error. I learned a lot about sorting methods and the different ways to implement them. I also notice that most sorting algorithms are quite similar and can often be confused with each other. I enjoyed the project and will continue to look for more efficient ways to solve the sorting problems.

**Possible Improvements:** I feel that my critical operation count is high when sorting through the arrays. I tried to find new ways to decrease he average number but it always changed the code into a different sorting algorithm than the one assigned to me. I will continue to look into this and post new findings.