CSC2023 - Algorithms Assignment 1

For this assignment I was asked to implement three sorting algorithms in java based on some given pseudo code, I was then asked to analyse the performance of the sorting algorithms when sorting arrays of various sizes by counting the comparisons involving the array made by each algorithm.

# Insertion Sort and Quicksort

Insertion Sort and Quicksort were the first two algorithms I implemented. For these algorithms I was almost able to follow the pseudo exactly. However I encountered issues when trying to accurately implement a comparison count for each algorithm. In order to accurately count the comparisons made to the array for these two algorithms I have Implemented a comparison method, this method simply returns the Boolean result of the expression passed in as its parameter, which while useless by itself has helped as I can increment the appropriate comparison counter in the method body. This allows me to accurately count the number of comparisons without compromising the structure of the code as I can simply call these methods using the expressions as parameters in place of just the expressions. As instructed I ran four tests on each algorithm using the provided text files.

The following shows the output of the display method after the array has been processed by each sorting algorithm.

## Test 1 (Test1.txt):

For test one I was given the following 15 unformatted integers:

101, 7, 73, 110, 57, 120, 66, 25, 12, 65,

97, 51, 18, 41, 103

Which upon running through my sorting algorithms produced these results:

|  |  |
| --- | --- |
| Insertion Sort | Quicksort |
| 007 012 018 025 041 051 057 065 066 073  097 101 103 110 120 | 007 012 018 025 041 051 057 065 066 073  097 101 103 110 120 |

## Test 2 (Test2.txt):

For test two I was given the following 15 unformatted integers:

14, 7, 18, 25, 41, 110, 43, 73, 66, 65,

97, 103, 59, 112, 120

Which upon running through my sorting algorithms produced these results:

|  |  |
| --- | --- |
| Insertion Sort | Quicksort |
| 007 014 018 025 041 043 059 065 066 073  097 103 110 112 120 | 007 014 018 025 041 043 059 065 066 073  097 103 110 112 120 |

## Test 3 (Test3.txt):

For test three I was given the following 100 unformatted integers:

150, 190, 94, 99, 169, 46, 73, 171, 199, 193, 148, 184, 113, 184, 46, 69, 9, 158, 63, 118, 1, 169, 175, 20, 189, 2, 125, 45, 33, 23, 169, 147, 178, 43, 3, 33, 150, 76, 6, 138, 107, 20, 110, 88, 136, 144, 106, 55, 128, 1, 129, 101, 29, 20, 196, 11, 61, 184, 91, 56, 80, 184, 1, 94, 28, 5, 92, 95, 77, 127, 103, 81, 69, 1, 57, 70, 150, 153, 66, 72, 46, 143, 83, 46, 79, 170, 199, 156, 199, 101, 55, 180, 105, 30, 20, 71, 85, 46, 21, 55

Which upon running through my sorting algorithms produced these results:

|  |  |
| --- | --- |
| Insertion Sort | Quicksort |
| 001 001 001 001 002 003 005 006 009 011  020 020 020 020 021 023 028 029 030 033  033 043 045 046 046 046 046 046 055 055  055 056 057 061 063 066 069 069 070 071  072 073 076 077 079 080 081 083 085 088  091 092 094 094 095 099 101 101 103 105  106 107 110 113 118 125 127 128 129 136  138 143 144 147 148 150 150 150 153 156  158 169 169 169 170 171 175 178 180 184  184 184 184 189 190 193 196 199 199 199 | 001 001 001 001 002 003 005 006 009 011  020 020 020 020 021 023 028 029 030 033  033 043 045 046 046 046 046 046 055 055  055 056 057 061 063 066 069 069 070 071  072 073 076 077 079 080 081 083 085 088  091 092 094 094 095 099 101 101 103 105  106 107 110 113 118 125 127 128 129 136  138 143 144 147 148 150 150 150 153 156  158 169 169 169 170 171 175 178 180 184  184 184 184 189 190 193 196 199 199 199 |

## Test 4 (Test4.txt):

For test four I was given the following 100 unformatted integers:

1, 3, 41, 3, 1, 3, 5, 6, 9, 1, 11, 21, 21, 3, 21, 5, 9, 9, 28, 29, 30, 33, 41, 28, 41, 33, 43, 41, 43, 33, 54, 55, 55, 56, 60, 60, 41, 56, 63, 56, 57, 66, 67, 69, 69, 70, 71, 180, 73, 74, 79, 138, 80, 85, 80, 91, 80, 94, 91, 94, 94, 95, 99, 94, 101, 101, 94, 101, 103, 105, 101, 107, 101, 115, 118, 115, 127, 115, 127, 136, 80, 115, 136, 147, 148, 148, 150, 152, 148, 152, 170, 152, 163, 169, 170, 170, 169, 180, 74, 180

Which upon running through my sorting algorithms produced these results:

|  |  |
| --- | --- |
| Insertion Sort | Quicksort |
| 001 001 001 003 003 003 003 005 005 006  009 009 009 011 021 021 021 028 028 029  030 033 033 033 041 041 041 041 041 043  043 054 055 055 056 056 056 057 060 060  063 066 067 069 069 070 071 073 074 074  079 080 080 080 080 085 091 091 094 094  094 094 094 095 099 101 101 101 101 101  103 105 107 115 115 115 115 118 127 127  136 136 138 147 148 148 148 150 152 152  152 163 169 169 170 170 170 180 180 180 | 001 001 001 003 003 003 003 005 005 006  009 009 009 011 021 021 021 028 028 029  030 033 033 033 041 041 041 041 041 043  043 054 055 055 056 056 056 057 060 060  063 066 067 069 069 070 071 073 074 074  079 080 080 080 080 085 091 091 094 094  094 094 094 095 099 101 101 101 101 101  103 105 107 115 115 115 115 118 127 127  136 136 138 147 148 148 148 150 152 152  152 163 169 169 170 170 170 180 180 180 |

As shown in the above tests both my Insertion sort and Quicksort implementation are accurately sorting all the arrays into ascending order with no errors in any test.

I shall now use the number of comparisons involving the array in each algorithm as a basis for analysing performance.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Array Size | Insertion Sort Comparisons | Quicksort Count Comparisons |
| Test 1 | 15 | 72 | 77 |
| Test 2 | 15 | 29 | 105 |
| Test 3 | 100 | 2840 | 970 |
| Test 4 | 100 | 361 | 1518 |

As shown in the table above Insertion sort performs better then quick sort in three out of four tests.

In test one Insertion Sort performs the best beating QuickSort by five comparisons, this is to be expected as the size of the array is only 15 and Quicksort’s divide and conquer strategy works more effectively on arrays of larger sizes, whereas insertion sort works better on smaller arrays on average due it falling under O(N^2) in big Oh notation.

Insertion Sort also performed the best in the second test beating Quicksort by 76 comparisons despite the array size being the same as test one, I think this is because the input plays towards Insertion Sorts best cases as it is closer to being sorted and is a smaller sized array. Since the input is closer to being sorted it has effected Quicksort’s performance as the closer the input is to being sorted the worse quick sort will perform. This is due to the fact that my algorithm uses the rightmost element of the array as the pivot for the partition, and in this case the largest element in the array is the rightmost element.

Test three is where Quicksort performed the best, this test dealt with sorting an array of size 100 whose elements were very unsorted. This played better to Quicksort’s best case, as on average it copes with larger inputs better than insertion sort.

Test four showed Insertion Sort outperforming Quicksort on an array of size 100. I think the reason this has happened is because as shown in the input the array was fairly sorted to begin with; and the largest element (180) was also the rightmost elements. These two factors are closer to Quicksort’s worst case, but the array nearly being sorted is closer to Insertions Sorts’ best case as it requires less comparisons to complete sorting.

To summarise:

Insertion sort is more efficient for sorting smaller arrays and for sorting arrays that are nearly sorted.

Quicksort is more efficient for sorting large randomly sorted arrays.

# New Sort

I was given a new algorithm by the name of new sort that I was told to implement and analyse in the same way as Insertion Sort and Quicksort; by counting the comparisons using the array in the algorithm.

## Test 1 (Test3.txt):

For test one I was given the following 100 unformatted integers:

150, 190, 94, 99, 169, 46, 73, 171, 199, 193, 148, 184, 113, 184, 46, 69, 9, 158, 63, 118, 1, 169, 175, 20, 189, 2, 125, 45, 33, 23, 169, 147, 178, 43, 3, 33, 150, 76, 6, 138, 107, 20, 110, 88, 136, 144, 106, 55, 128, 1, 129, 101, 29, 20, 196, 11, 61, 184, 91, 56, 80, 184, 1, 94, 28, 5, 92, 95, 77, 127, 103, 81, 69, 1, 57, 70, 150, 153, 66, 72, 46, 143, 83, 46, 79, 170, 199, 156, 199, 101, 55, 180, 105, 30, 20, 71, 85, 46, 21, 55

Which upon running through my sorting algorithms produced these results:

|  |
| --- |
| New Sort |
| 001 001 001 001 002 003 005 006 009 011  020 020 020 020 021 023 028 029 030 033  033 043 045 046 046 046 046 046 055 055  055 056 057 061 063 066 069 069 070 071  072 073 076 077 079 080 081 083 085 088  091 092 094 094 095 099 101 101 103 105  106 107 110 113 118 125 127 128 129 136  138 143 144 147 148 150 150 150 153 156  158 169 169 169 170 171 175 178 180 184  184 184 184 189 190 193 196 199 199 199 |

## Test 2 (Test4.txt):

For test one I was given the following 100 unformatted integers:

1, 3, 41, 3, 1, 3, 5, 6, 9, 1, 11, 21, 21, 3, 21, 5, 9, 9, 28, 29, 30, 33, 41, 28, 41, 33, 43, 41, 43, 33, 54, 55, 55, 56, 60, 60, 41, 56, 63, 56, 57, 66, 67, 69, 69, 70, 71, 180, 73, 74, 79, 138, 80, 85, 80, 91, 80, 94, 91, 94, 94, 95, 99, 94, 101, 101, 94, 101, 103, 105, 101, 107, 101, 115, 118, 115, 127, 115, 127, 136, 80, 115, 136, 147, 148, 148, 150, 152, 148, 152, 170, 152, 163, 169, 170, 170, 169, 180, 74, 180

Which upon running through my sorting algorithms produced these results:

|  |
| --- |
| New Sort |
| 001 001 001 003 003 003 003 005 005 006  009 009 009 011 021 021 021 028 028 029  030 033 033 033 041 041 041 041 041 043  043 054 055 055 056 056 056 057 060 060  063 066 067 069 069 070 071 073 074 074  079 080 080 080 080 085 091 091 094 094  094 094 094 095 099 101 101 101 101 101  103 105 107 115 115 115 115 118 127 127  136 136 138 147 148 148 148 150 152 152  152 163 169 169 170 170 170 180 180 180 |

## Test 3 (Test5.txt):

For test one I was given the following 100 unformatted integers:

150, 199, 99, 99, 169, 46, 72, 169, 199, 199, 153, 184, 127, 184, 46, 69, 1, 153, 69, 127, 1, 169, 184, 20, 184, 1, 127, 46, 28, 28, 169, 150, 184, 46, 28, 46, 150, 79, 69, 150, 107, 20, 107, 99, 150, 150, 107, 55, 127, 1, 127, 107, 28, 20, 199, 1, 69, 184, 99, 55, 79, 184, 1, 99, 28, 1, 99, 99, 79, 127, 107, 79, 69, 1, 55, 72, 150, 153, 69, 72, 46, 150, 99, 46, 79, 169, 199, 153, 199, 107, 55, 184, 107, 28, 20, 72, 99, 46, 28, 55

Which upon running through my sorting algorithms produced these results:

|  |
| --- |
| New Sort |
| 001 001 001 001 001 001 001 001 020 020  020 020 028 028 028 028 028 028 028 046  046 046 046 046 046 046 046 055 055 055  055 055 069 069 069 069 069 069 072 072  072 072 079 079 079 079 079 099 099 099  099 099 099 099 099 099 107 107 107 107  107 107 107 127 127 127 127 127 127 150  150 150 150 150 150 150 150 153 153 153  153 169 169 169 169 169 184 184 184 184  184 184 184 184 199 199 199 199 199 199 |

As shown in the above tests my implementation of this new algorithm correctly sorts arrays into ascending order. I was then asked to retest the Insertion Sort algorithm using test5.txt as an input.

# Insertion Sort Test 5 (Test5.txt):

For test one I was given the following 100 unformatted integers:

150, 199, 99, 99, 169, 46, 72, 169, 199, 199, 153, 184, 127, 184, 46, 69, 1, 153, 69, 127, 1, 169, 184, 20, 184, 1, 127, 46, 28, 28, 169, 150, 184, 46, 28, 46, 150, 79, 69, 150, 107, 20, 107, 99, 150, 150, 107, 55, 127, 1, 127, 107, 28, 20, 199, 1, 69, 184, 99, 55, 79, 184, 1, 99, 28, 1, 99, 99, 79, 127, 107, 79, 69, 1, 55, 72, 150, 153, 69, 72, 46, 150, 99, 46, 79, 169, 199, 153, 199, 107, 55, 184, 107, 28, 20, 72, 99, 46, 28, 55

Which upon running through my sorting algorithms produced these results:

|  |
| --- |
| New Sort |
| 001 001 001 001 001 001 001 001 020 020  020 020 028 028 028 028 028 028 028 046  046 046 046 046 046 046 046 055 055 055  055 055 069 069 069 069 069 069 072 072  072 072 079 079 079 079 079 099 099 099  099 099 099 099 099 099 107 107 107 107  107 107 107 127 127 127 127 127 127 150  150 150 150 150 150 150 150 153 153 153  153 169 169 169 169 169 184 184 184 184  184 184 184 184 199 199 199 199 199 199 |

The algorithm completed successfully with no errors. I will now compare the efficiency of New Sort against Insertion Sort.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Array Size | Insertion Sort | New Sort |
| Test 1 (Test3.txt) | 100 | 2840 | 7341 |
| Test 2 (Test4.txt) | 100 | 361 | 4968 |
| Test 3 (Test5.txt) | 100 | 2725 | 1682 |

Insertion sort performed better in tests 2 and 3, with new sort performing better in test 3. From this I can draw a few conclusions.

1. New Sort performs better than insertion sort when the array they are working on contains a lot of duplicate elements, because new sort finds the minimum value in the array and then takes all the values equal to the minimum and swaps them into their correct position, having more duplicates means New Sort has to find the minimum less times and therefore loop less times.
2. The worst case for New Sort occurs when the array only contains unique elements so no duplicates
3. The best case is when the array only contains the same value element, so is entirely made of a single element.