O. Guzelyte (160421859)
CSC2023: Algorithm Design and Analysis
Assignment 1: Implementing and Comparing Sorting Algorithms

TASK 3. Testing and Evaluation answers

1. Contrary to its purpose of sorting large arrays, Quicksort performs well when sorting the small array of "test1.txt" (83 comparisons). It performs almost as well as Insertion sort (73 comparisons), which is counter-intuitive. This happens because after partitioning the left size of array is randomized and the right one is sorted. Quicksort gets average of approximately $N\log_2 N$ performance. Array after partitioning looks like this: 101 7 73 41 57 18 66 25 12 65 97 51 **103** (pivot) 110 118.
2. Quicksort performs worse while sorting the second array of "test2.txt". The array is the same length as the first one (n = 15), however, it is sorted by default in a somewhat ascending order, therefore the partitions are very uneven. Quicksort gets average of approximately N^2 performance and makes 108 comparisons as opposing to 30 by Insertion sort (since array is in an ascending order). Array after partitioning looks like this: 14 5 18 25 41 110 43 73 66 65 97 103 59 112 **118** (pivot).
3. Quicksort performs well with the large array of "test3.txt", gets a bit bigger than average of $N\log_2 N$ performance and makes 1008 comparisons, however, Insertion sort performs notoriously badly with this type of array. The array is large and seems to have small numbers at the end of it, therefore, Insertion sort is forced to make 2882 comparisons, which is 1874 comparisons more than Quicksort's performance.
4. Quicksort performs badly with the 4th array of "test4.txt" even though it has the same length as array of "test3.txt". Quicksort does 1563 comparisons, which is poorer than the average performance of the "test3.txt" array (1008 comparisons). This happens because the array is in an ascending order and the partitioning is very uneven.
5. The fact that the 4th array of "test4.txt" is in ascending order is what causes Insertion sort to speed up. Even though Quicksort does 1563 comparisons to sort the array, Insertion sort does 447, which is 1116 comparisons less. This result is contradictory to "test3.txt" array results when Insertion sort did 1874 more comparisons than Quicksort sorting an array of the same size as "test4.txt".

TASK 5. Further Testing and Evaluation answers

1. The new sorting algorithm performs better than Insertion sort when testing the 5th array ("test5.txt") because it contains a lot of duplicate values. Insertion sort keeps finding smaller values and adding them into beginning of the sorted array. Newsort finds the smallest value from the current position and puts it on the left side of a completely sorted array. When there are a lot of duplicate values, this process is greatly accelerated, therefore Newsort performs better – 1682 comparisons, while Insertion sort does 3011.
2. Worst case performance occurs when there are no duplicates in the array and the numbers are completely randomised, the performance then would be $O(N^2)$.
3. Best case performance is when array consists of one duplicate value repeated throughout the array, so the algorithm goes through the array once. Big Oh performance is O(N).

Actual Test Results

----- Insertion sort testing -----

Input Array 1, File: test1.txt.
007 012 018 025 041 051 057 065 066 073
097 101 103 110 118
Insertion sort comparison counter: 73, File: test1.txt.

Input Array 2, File: test2.txt.
005 014 018 025 041 043 059 065 066 073
097 103 110 112 118
Insertion sort comparison counter: 30, File: test2.txt.

Input Array 3, File: test3.txt.
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
138 140 143 144 147 148 150 150 153 156
158 169 169 169 170 171 175 178 180 184
184 184 184 189 190 193 198 199 199 199
Insertion sort comparison counter: 2882, File: test3.txt.

Input Array 4, File: test4.txt.
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
152 163 169 169 170 170 170 180 190 190
Insertion sort comparison counter: 447, File: test4.txt.

Input Array 5, File: test5.txt.
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
Insertion sort comparison counter: 3011, File: test5.txt.

----- Quick sort testing -----

Input Array 1, File: test1.txt.
007 012 018 025 041 051 057 065 066 073
097 101 103 110 118

Quicksort comparison counter: 83, File: test1.txt.

Input Array 2, File: test2.txt.
005 014 018 025 041 043 059 065 066 073
097 103 110 112 118
Quicksort comparison counter: 108, File: test2.txt.

Input Array 3, File: test3.txt.
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
138 140 143 144 147 148 150 150 153 156
158 169 169 169 170 171 175 178 180 184
184 184 184 189 190 193 198 199 199 199
Quicksort comparison counter: 1008, File: test3.txt.

Input Array 4, File: test4.txt.
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
152 163 169 169 170 170 170 180 190 190
Quicksort comparison counter: 1563, File: test4.txt.

----- New sort testing -----

Input Array 1, File: test3.txt.
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
138 140 143 144 147 148 150 150 153 156
158 169 169 169 170 171 175 178 180 184
184 184 184 189 190 193 198 199 199 199
Newsort sort comparison counter: 7388, File: test3.txt.

Input Array 2, File: test4.txt.

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
152 163 169 169 170 170 170 180 190 190
Newsort sort comparison counter: 4971, File: test4.txt.

Input Array 3, File: test5.txt.
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
Newsort sort comparison counter: 1682, File: test5.txt.