**Assignment 1 Test Report**

*Test1.txt*

Input Array:

101 007 073 110 057 118 066 025 012 065

097 051 018 041 103

Insertion Sort Result:

007 012 018 025 041 051 057 065 066 073

097 101 103 110 118

**Number of comparisons: 73**

Quicksort Result:

007 012 018 025 041 051 057 065 066 073

097 101 103 110 118

**Number of comparisons: 83**

In test 1, it can be seen that insertion sort had the best performance, carrying out 10 less comparisons than quicksort. It can be concluded that this is because quicksort is not optimal for small arrays, and furthermore the pivot point in the array is the second largest element, resulting in very uneven partitions which hinders the performance of the algorithm.

*Test2.txt*

Input Array:

014 005 018 025 041 110 043 073 066 065

097 103 059 112 118

Insertion Sort Result:

005 014 018 025 041 043 059 065 066 073

097 103 110 112 118

**Number of comparisons: 30**

Quicksort Result:

005 014 018 025 041 043 059 065 066 073

097 103 110 112 118

**Number of comparisons: 108**

From the comparison results from test 2 it can be seen that insertion sort has the best performance. This can, again, be attributed to the small size of the input array, but also to the fact that the input array is already almost sorted. This leans towards the best case scenario with insertion sort, whereas in quicksort this creates the worst case scenario, especially since the pivot point selected is the largest value in the array. This creates extremely uneven partitions which hinders the performance of the algorithm.

*Test3.txt*

Input Array:

140 190 094 099 169 046 073 171 199 193

148 184 113 184 046 069 009 158 063 118

001 169 175 020 189 002 125 045 033 023

169 147 178 043 003 033 150 076 006 138

107 020 110 088 136 144 106 055 128 001

129 101 029 020 198 011 061 184 091 056

080 184 001 094 028 005 092 095 077 127

103 081 069 001 057 070 150 153 066 072

046 143 083 046 079 170 199 156 199 101

055 180 105 030 020 071 085 046 021 055

Insertion Sort Result:

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

**Number of comparisons: 2840**

Quicksort Result:

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

**Number of comparisons: 1008**

Newsort Result:

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

**Number of comparisons: 7388**

It can be seen from the number of comparisons performed by the insertion sort and quicksort algorithms on the input array that quicksort is by far the better performing of the two, carrying out well under half of the comparisons done by insertion sort. This is because the data in the array is randomly placed, hindering the performance of insertion sort as its best case occurs when the data is already sorted. Furthermore, the pivot point in the input array is a median element within the given values, creating the conditions for quicksort’s best case, allowing the algorithm to create fairly even partitions.

*Test4.txt*

Input Array:

001 003 041 003 001 003 005 006 009 001

011 021 021 003 021 005 009 009 028 029

030 033 041 028 041 033 043 041 043 033

054 055 055 056 060 060 041 056 063 056

057 066 067 069 069 070 071 190 073 074

079 138 080 085 080 091 080 094 091 094

094 095 099 094 101 101 094 101 103 105

101 107 101 115 118 115 127 115 127 136

080 115 136 147 148 148 150 152 148 152

170 152 163 169 170 170 169 190 074 180

Insertion Sort Results:

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

**Number of comparisons: 363**

Quicksort Results:

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

**Number of comparisons: 1563**

Newsort Results:

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

**Number of comparisons: 4971**

*Test5.txt*

Insertion Sort Results:

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

**Number of comparisons: 2728**

Newsort Results:

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

**Number of comparisons: 1682**

It can be seen from the results produced by applying the insertion sort and quicksort algorithms to input 4 that yet again, insertion sort has the best performance with fewer comparisons. This is because the input array does not to be changed too much to be sorted. With the larger elements found in the left of the array, the smaller ones being found in the right, and the pivot element being the second largest value, the quicksort algorithm produces very uneven partitions, thus needing to perform more comparisons to sort the array.

Overall, the most important things to be concluded from these test results are:

* Quicksort does not perform well on small arrays, whereas insertion sort does.
* Quicksort does not perform well on ‘nearly sorted’ arrays, as it results in uneven partitions.
* Quicksort performs best when the pivot point is a median value in the array, as this allows it to create even partitions.
* Although insertion sort produced the best performance when sorting most of the inputs, it still took a large amount of comparisons to sort larger arrays. Thus, it would be recommended to use a different sorting algorithm on larger inputs.

Question 5)

1. The newsort algorithm only performs better than insertion sort in test 5. This is because the newsort algorithm performs best on arrays that have many duplicates and the test5.txt input contains duplicates of every value in the array.
2. The worst case performance for the newsort algorithm is O(N^2). This occurs when there are no duplicate values in the input array.
3. The best case performance for the newsort algorithm is O(N). This occurs when the input array is made up entirely of duplicate values.