Phuc Hong Le

CSS 342

Professor Ahmed Awad

November 25, 2017

**Assignment 4 Documentation**

**Size of the array to be sorted is 30**

**initial vector for Improved mergeSort:**

**items[0] = 7**

**items[1] = 19**

**items[2] = 23**

**items[3] = 8**

**items[4] = 10**

**items[5] = 2**

**items[6] = 24**

**items[7] = 0**

**items[8] = 5**

**items[9] = 12**

**items[10] = 27**

**items[11] = 22**

**items[12] = 3**

**items[13] = 9**

**items[14] = 17**

**items[15] = 18**

**items[16] = 26**

**items[17] = 25**

**items[18] = 20**

**items[19] = 21**

**items[20] = 13**

**items[21] = 6**

**items[22] = 1**

**items[23] = 11**

**items[24] = 28**

**items[25] = 4**

**items[26] = 16**

**items[27] = 14**

**items[28] = 15**

**items[29] = 29**

**elapsed time: 7**

**sorted:**

**items[0] = 0**

**items[1] = 1**

**items[2] = 2**

**items[3] = 3**

**items[4] = 4**

**items[5] = 5**

**items[6] = 6**

**items[7] = 7**

**items[8] = 8**

**items[9] = 9**

**items[10] = 10**

**items[11] = 11**

**items[12] = 12**

**items[13] = 13**

**items[14] = 14**

**items[15] = 15**

**items[16] = 16**

**items[17] = 17**

**items[18] = 18**

**items[19] = 19**

**items[20] = 20**

**items[21] = 21**

**items[22] = 22**

**items[23] = 23**

**items[24] = 24**

**items[25] = 25**

**items[26] = 26**

**items[27] = 27**

**items[28] = 28**

**items[29] = 29**

**initial vector for Original mergeSort:**

**items[0] = 7**

**items[1] = 19**

**items[2] = 23**

**items[3] = 8**

**items[4] = 10**

**items[5] = 2**

**items[6] = 24**

**items[7] = 0**

**items[8] = 5**

**items[9] = 12**

**items[10] = 27**

**items[11] = 22**

**items[12] = 3**

**items[13] = 9**

**items[14] = 17**

**items[15] = 18**

**items[16] = 26**

**items[17] = 25**

**items[18] = 20**

**items[19] = 21**

**items[20] = 13**

**items[21] = 6**

**items[22] = 1**

**items[23] = 11**

**items[24] = 28**

**items[25] = 4**

**items[26] = 16**

**items[27] = 14**

**items[28] = 15**

**items[29] = 29**

**elapsed time: 8**

**sorted:**

**items[0] = 0**

**items[1] = 1**

**items[2] = 2**

**items[3] = 3**

**items[4] = 4**

**items[5] = 5**

**items[6] = 6**

**items[7] = 7**

**items[8] = 8**

**items[9] = 9**

**items[10] = 10**

**items[11] = 11**

**items[12] = 12**

**items[13] = 13**

**items[14] = 14**

**items[15] = 15**

**items[16] = 16**

**items[17] = 17**

**items[18] = 18**

**items[19] = 19**

**items[20] = 20**

**items[21] = 21**

**items[22] = 22**

**items[23] = 23**

**items[24] = 24**

**items[25] = 25**

**items[26] = 26**

**items[27] = 27**

**items[28] = 28**

**items[29] = 29**

**initial vector for QuickSort:**

**items[0] = 7**

**items[1] = 19**

**items[2] = 23**

**items[3] = 8**

**items[4] = 10**

**items[5] = 2**

**items[6] = 24**

**items[7] = 0**

**items[8] = 5**

**items[9] = 12**

**items[10] = 27**

**items[11] = 22**

**items[12] = 3**

**items[13] = 9**

**items[14] = 17**

**items[15] = 18**

**items[16] = 26**

**items[17] = 25**

**items[18] = 20**

**items[19] = 21**

**items[20] = 13**

**items[21] = 6**

**items[22] = 1**

**items[23] = 11**

**items[24] = 28**

**items[25] = 4**

**items[26] = 16**

**items[27] = 14**

**items[28] = 15**

**items[29] = 29**

**elapsed time: 3**

**sorted:**

**items[0] = 0**

**items[1] = 1**

**items[2] = 2**

**items[3] = 3**

**items[4] = 4**

**items[5] = 5**

**items[6] = 6**

**items[7] = 7**

**items[8] = 8**

**items[9] = 9**

**items[10] = 10**

**items[11] = 11**

**items[12] = 12**

**items[13] = 13**

**items[14] = 14**

**items[15] = 15**

**items[16] = 16**

**items[17] = 17**

**items[18] = 18**

**items[19] = 19**

**items[20] = 20**

**items[21] = 21**

**items[22] = 22**

**items[23] = 23**

**items[24] = 24**

**items[25] = 25**

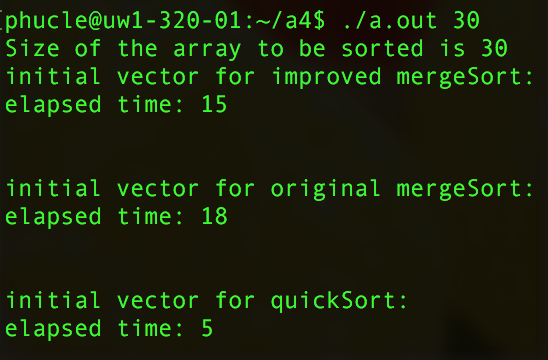
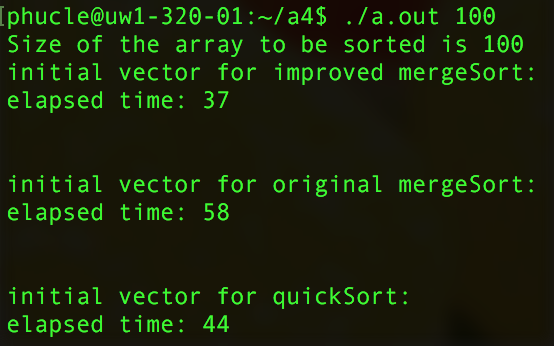
**items[26] = 26**

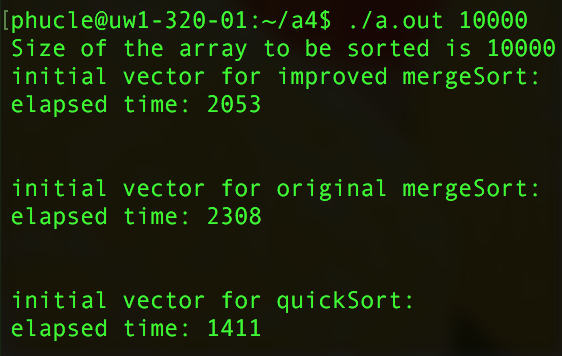
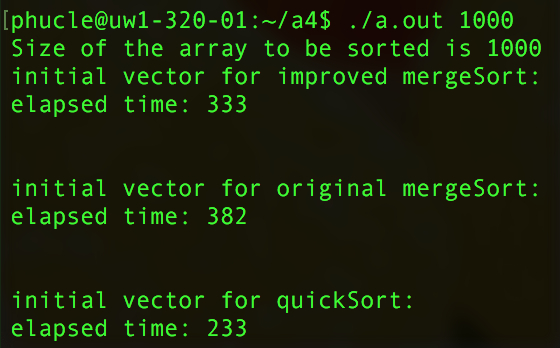
**items[27] = 27**

**items[28] = 28**

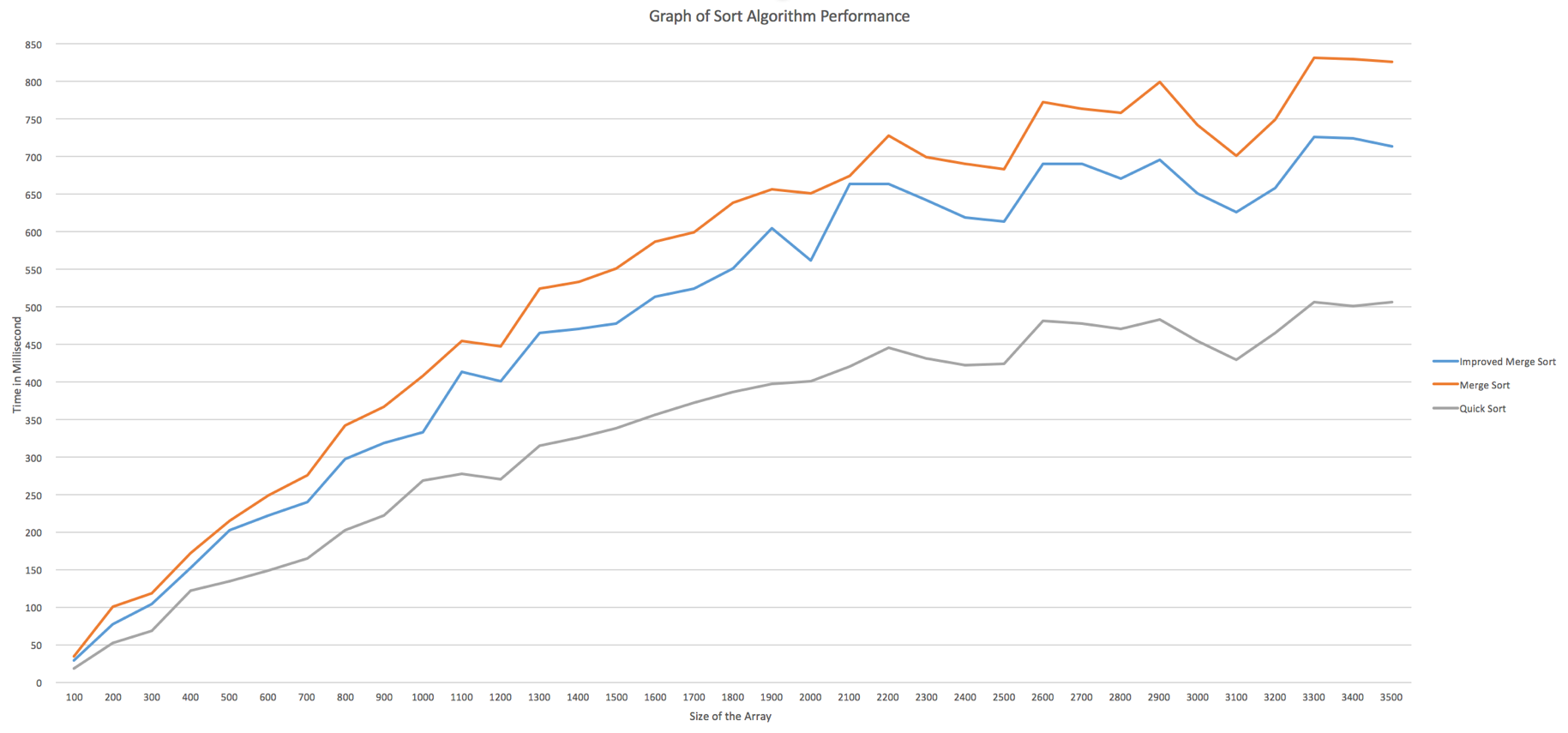
**items[29] = 29**

**Program ended with exit code: 0**





Graph of each sort algorithm performance:



According to the data from the graph and the output, we can see that my improved merge sort is very efficient for array with big size rather than small size. The reason for this is because the data need to be transfer between two different arrays (original array and the temporary array). As the array size grows, the number of sorted elements need to pass back will grow accordingly; therefore, it is efficient with arrays with big size. The copy between the original array and the temporary array is made every time the temporary array is filled with sorted elements from the original array. Only then, the original array will make a copy of the sorted temp-array and use it for the next iteration.

I used the same data set in order to get all three sorts’ run-time more precisely as the sort might need to work differently with different data set. I modified the driver.cpp, mergeSort.cpp, and quicksort.cpp functions to fit the driver functions needs. By doing this, all three sorts can work on the same testing system which will be easier and more efficient to get the correct run-time of all sorts.