University of Windsor

COMP-8547-1-R-2021S

Advanced Computing Concepts
Summer 2021
Assignment 2

Submitted By
Anubha Sharma
Master of Applied Computing
Student Id: 110037181

Submitted To
Dr. Abedalrhman Alkhateeb

University of Windsor

I confirm that I will keep the content of this assignment confidential. I confirm that I have not received any unauthorized assistance in preparing for or writing this assignment. I acknowledge that a mark of 0 may be assigned for copied work.”

Anubha
110037181

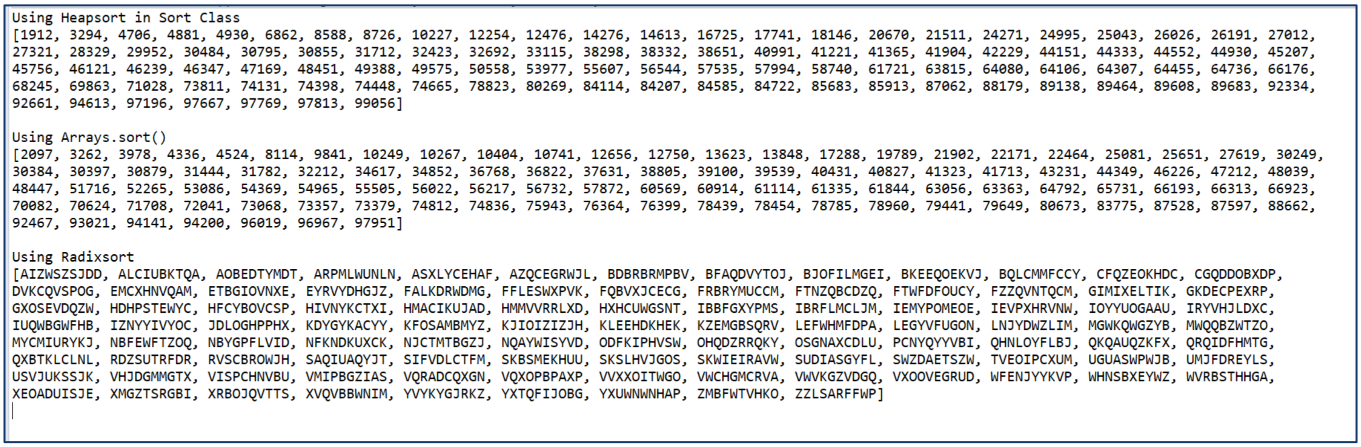
Question 1. Use class Sort.java provided in class, the dual-pivot Quicksort of Java 8 (Arrays.sort), and RadixSort.java provided in class.

Solution:
Please refer the project sorting, package solution, class QuestionOne, and method main for the coded solution.

The steps followed to resolve are as follows:

1. Created a list of 100 random Integers and sorted with HeapSort solution provided in class.
2. Created another list of 100 random Integers and sorted with Arrays.sort.
3. Created a list of 100 random Strings of length 10 and sorted with RadixSort.

OUTPUT:



```
Using Heapsort in Sort Class
[1912, 3294, 4706, 4881, 4930, 6862, 8588, 8726, 10227, 12254, 12476, 14276, 14613, 16725, 17741, 18146, 20670, 21511, 24271, 24995, 25043, 26026, 26191, 27012,
27321, 28329, 29952, 30484, 30795, 30855, 31712, 32423, 32692, 33115, 38298, 38332, 38651, 40991, 41221, 41365, 41904, 42229, 44151, 44333, 44552, 44930, 45207,
45756, 46121, 46239, 46347, 47169, 48451, 49388, 49575, 50558, 53977, 55607, 56544, 57535, 57994, 58740, 61721, 63815, 64080, 64106, 64307, 64455, 64736, 66176,
68245, 69863, 71028, 73811, 74131, 74398, 74448, 74665, 78823, 80269, 84114, 84207, 84585, 84722, 85683, 85913, 87062, 88179, 89138, 89464, 89608, 89683, 92334,
92661, 94613, 97196, 97667, 97769, 97813, 99056]

Using Arrays.sort()
[2097, 3262, 3978, 4336, 4524, 8114, 9841, 10249, 10267, 10404, 10741, 12656, 12750, 13623, 13848, 17288, 19789, 21902, 22171, 22464, 25081, 25651, 27619, 30249,
30384, 30397, 30879, 31444, 31782, 32212, 34617, 34852, 36768, 36822, 37631, 38805, 39100, 39539, 40431, 40827, 41323, 41713, 43231, 44349, 46226, 47212, 48039,
48447, 51716, 52265, 53086, 54369, 54965, 55505, 56022, 56217, 56732, 57872, 60569, 60914, 61114, 61335, 61844, 63056, 63363, 64792, 65731, 66193, 66313, 66923,
70082, 70624, 71708, 72041, 73068, 73357, 73379, 74812, 74836, 75943, 76364, 76399, 78439, 78454, 78785, 78960, 79441, 79649, 80673, 83775, 87528, 87597, 88662,
92467, 93021, 94141, 94200, 96019, 96967, 97951]

Using Radixsort
[AIZWSZSJDD, ALCIUBKTQA, AOBEDTYMDT, ARPMLWUNLN, ASXLYCEHAF, AZQCEGRWJL, BDBRBRMPBV, BFAQDVYTOJ, BJOFILMGEI, BKEEQOEKVJ, BQLCMMFCCY, CFQZEOKHDC, CGQDDOBXDP,
DVKCQVSPOG, EMCXHNVQAM, ETBGIOVNXE, EYRVYDHGJZ, FALKDRWDMG, FFLESWXPVK, FQBVXJCECG, FRBRYMUCCM, FTNZQBCDZQ, FTWFDFOUCY, FZZQVNTQCM, GIMIXELTIK, GKDECPEXRP,
GXOSEVDQZW, HDHPSTEWYC, HFCYBOVCSP, HIVNYKCTXI, HMACIKUJAD, HMMVVRRLXD, HXHCUWGSNT, IBBFGXYPMS, IBRFLMCLJM, IEMYPOMEOE, IEVPXHRVNW, IOYYUOGAAU, IRYVHJLDXC,
IUQWBGWFHB, IZNYYIVYOC, JDLOGHPPHX, KDYGYKACYY, KFOSAMBMYZ, KJIOIZIZJH, KLEEHDKHEK, KZEMGBSQRV, LEFWHMFDPA, LEGYVFUGON, LNJYDWZLIM, MGWKQWGZYB, MWQQBZWTZO,
MYCMIURYKJ, NBFEWFTZOQ, NBYGPFLVID, NFKNDKUXCK, NJCTMTBGZJ, NQAYWISYVD, ODFKIPHVSW, OHQDZRRQKY, OSGNAXCDLU, PCNYQYYVBI, QHNLOYFLBJ, QKQAUQZKFX, QRQIDFHMTG,
QXBTKLCLNL, RDZSUTRFDR, RVSCBROWJH, SAQIUAQYJT, SIFVDLCTFM, SKBSMEKHUU, SKSLHVJGOS, SKWIEIRAVW, SUDIASGYFL, SWZDAETSZW, TVEOIPCXUM, UGUASWPWJB, UMJFDREYLS,
USVJUKSSJK, VHJDGMMGTX, VISPCHNVBU, VMIPBGZIAS, VQRADCQXGN, VQXOPBPAXP, VVXXOITWGO, VWCHGMCRVA, VWVKGZVDGQ, VXOOVEGRUD, WFENJYYKVP, WHNSBXEYWZ, WVRBSTHHGA,
XEOADUISJE, XMGZTSRGBI, XRBOJQVTTS, XVQVBBWNIM, YVYKYGJRKZ, YXTQFIJOBG, YXUWNWNHAP, ZMBFWTVHKO, ZZLSARFFWP]
```

Question 2:
Do the following for Mergesort, Quicksort, Heapsort and dual-pivot Quicksort:

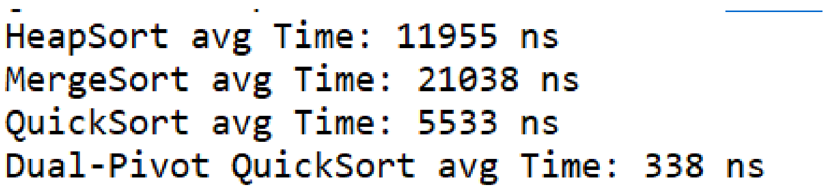
a. Create 100,000 random keys (of type long) and sort them. Repeat this 100 times.
b. Compute the average CPU time taken to sort the keys for the four methods.
c. Comment on the results and compare them to the average-case complexities discussed in class.

Solution 2:
Please refer to project sorting, package solution, class QuestionTwo and method main for the coded solution.

Steps followed for resolving are as follows.

1. Created an array of 100,000 random integers with Math.random
2. Created four references for the same array to pass into each sorting algorithm.
3. Sorted them with mergesort, quicksort, heapsort methods present in Sort.java provided in class.
4. Also, sorted with Arrays.sort.
5. Repeated steps 1-4 for 100 times.
6. Calculated the average time.

OUTPUT:
Dual -pivot quicksort i.e. Arrays.sort has taken the least amount of time , 338 ns while merge sort has taken the most amount of time, 21038 ns.



```
HeapSort avg Time: 11955 ns
MergeSort avg Time: 21038 ns
QuickSort avg Time: 5533 ns
Dual-Pivot QuickSort avg Time: 338 ns
```

Question 3: Do the following for the four sorting methods of #2, and for Radix sort:

a. Create 100,000 random strings of length 4 and sort them using the five sorting methods.

b. Repeat (a) 10 times and compute the average CPU time that takes to sort the keys for the five methods.

c. Repeat (a) and (b) with strings of length 6, 8, 10.

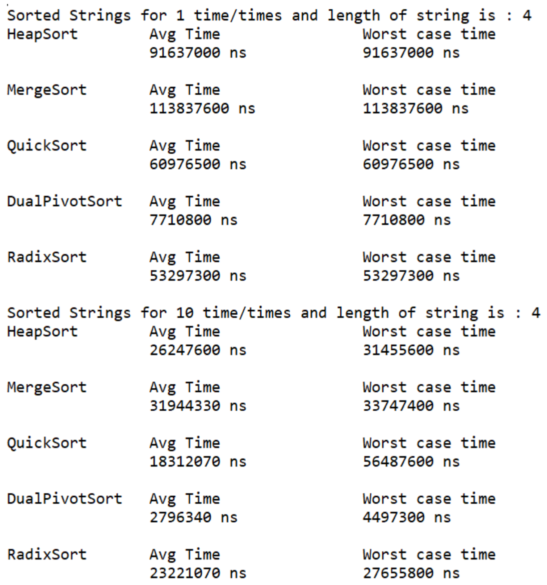
d. Create a table with the results and compare the times with the average-case and worstcase complexities as studied in class.

Solution 3:
Please refer to project sorting, package solution, class QuestionThree and method sortStrings for the coded solution.

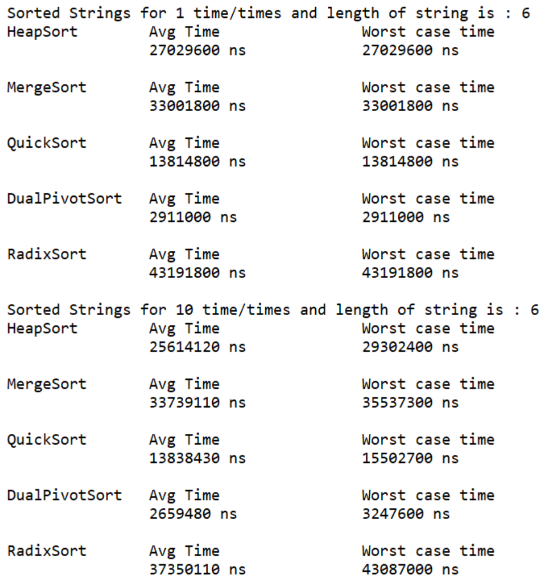
Steps followed to resolve are as follows:

1. Created an array of 100,000 random Strings of length 4 using method generateRandomStringArray in class Helper.java
2. Created five references for the same array to pass into each sorting algorithm.
3. Sorted them with mergesort, quicksort, heapsort methods present in Sort.java provided in class.
4. Also, sorted with Arrays.sort.
5. Also, sorted with RadixSort.java
6. For part b, repeated 1-5 steps 10 times
7. For part c, use step 1 to create string of length 6,8, 10
8. For part d, repeat 1-5 steps 10 times.

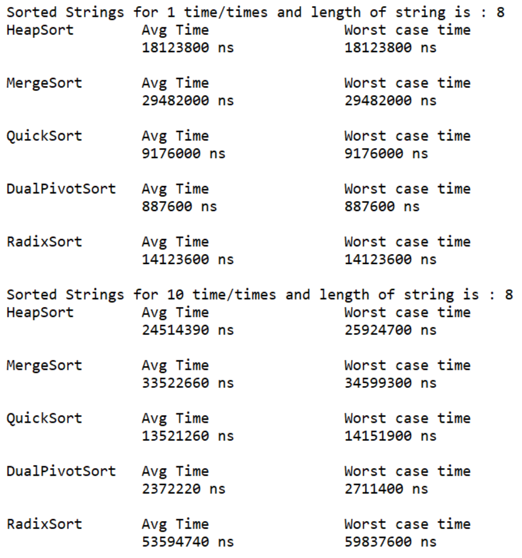
OUTPUT



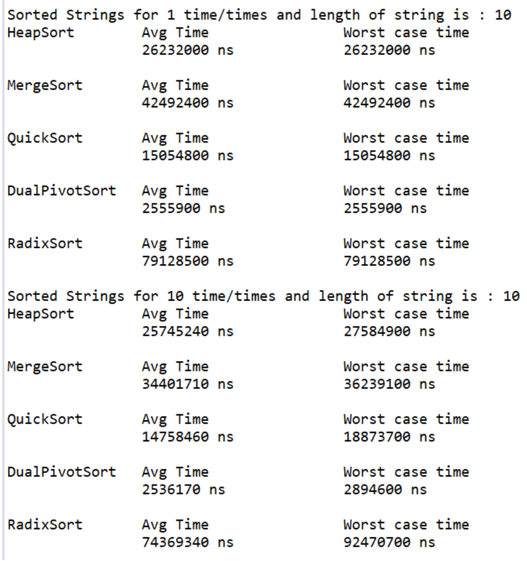
```
Sorted Strings for 1 time/times and length of string is : 4
HeapSort        Avg Time                Worst case time
                91637000 ns             91637000 ns

MergeSort       Avg Time                Worst case time
                113837600 ns            113837600 ns

QuickSort       Avg Time                Worst case time
                60976500 ns             60976500 ns

DualPivotSort   Avg Time                Worst case time
                7710800 ns              7710800 ns

RadixSort       Avg Time                Worst case time
                53297300 ns             53297300 ns

Sorted Strings for 10 time/times and length of string is : 4
HeapSort        Avg Time                Worst case time
                26247600 ns             31455600 ns

MergeSort       Avg Time                Worst case time
                31944330 ns             33747400 ns

QuickSort       Avg Time                Worst case time
                18312070 ns             56487600 ns

DualPivotSort   Avg Time                Worst case time
                2796340 ns              4497300 ns

RadixSort       Avg Time                Worst case time
                23221070 ns             27655800 ns
```



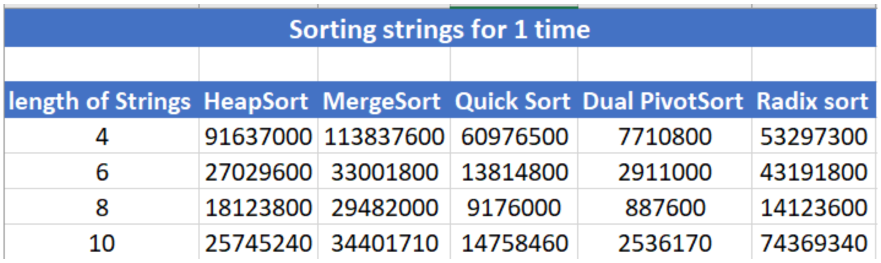
```
Sorted Strings for 1 time/times and length of string is : 6
HeapSort        Avg Time                Worst case time
                27029600 ns             27029600 ns

MergeSort       Avg Time                Worst case time
                33001800 ns             33001800 ns

QuickSort       Avg Time                Worst case time
                13814800 ns             13814800 ns

DualPivotSort   Avg Time                Worst case time
                2911000 ns              2911000 ns

RadixSort       Avg Time                Worst case time
                43191800 ns             43191800 ns

Sorted Strings for 10 time/times and length of string is : 6
HeapSort        Avg Time                Worst case time
                25614120 ns             29302400 ns

MergeSort       Avg Time                Worst case time
                33739110 ns             35537300 ns

QuickSort       Avg Time                Worst case time
                13838430 ns             15502700 ns

DualPivotSort   Avg Time                Worst case time
                2659480 ns              3247600 ns

RadixSort       Avg Time                Worst case time
                37350110 ns             43087000 ns
```



```
Sorted Strings  for 1 time/times and length of string is : 8
HeapSort        Avg Time                Worst case time
                18123800 ns             18123800 ns

MergeSort       Avg Time                Worst case time
                29482000 ns             29482000 ns

QuickSort       Avg Time                Worst case time
                9176000 ns              9176000 ns

DualPivotSort   Avg Time                Worst case time
                887600 ns               887600 ns

RadixSort       Avg Time                Worst case time
                14123600 ns             14123600 ns

Sorted Strings  for 10 time/times and length of string is  : 8
HeapSort        Avg Time                Worst case time
                24514390 ns             25924700 ns

MergeSort       Avg Time                Worst case time
                33522660 ns             34599300 ns

QuickSort       Avg Time                Worst case time
                13521260 ns             14151900 ns

DualPivotSort   Avg Time                Worst case time
                2372220 ns              2711400 ns

RadixSort       Avg Time                Worst case time
                53594740 ns             59837600 ns
```

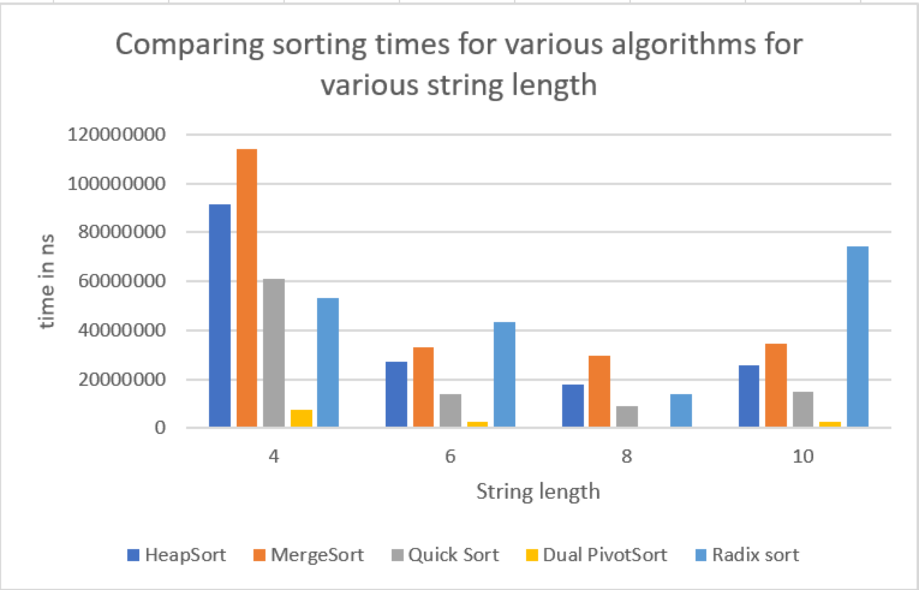


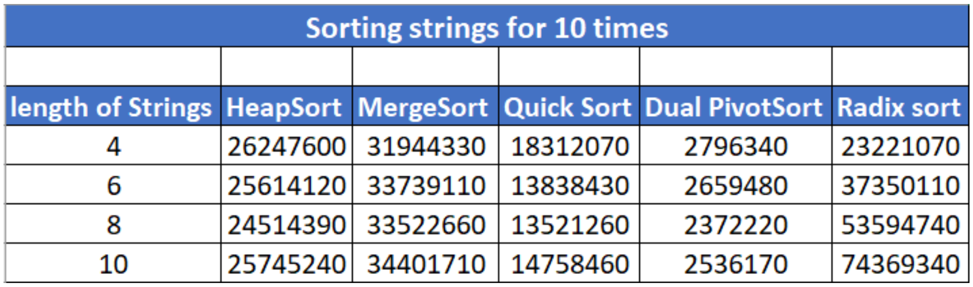
```
Sorted Strings  for 1 time/times and length of string is  : 10
HeapSort        Avg Time                Worst case time
                26232000 ns             26232000 ns

MergeSort       Avg Time                Worst case time
                42492400 ns             42492400 ns

QuickSort       Avg Time                Worst case time
                15054800 ns             15054800 ns

DualPivotSort   Avg Time                Worst case time
                2555900 ns              2555900 ns

RadixSort       Avg Time                Worst case time
                79128500 ns             79128500 ns

Sorted Strings  for 10 time/times and length of string is  : 10
HeapSort        Avg Time                Worst case time
                25745240 ns             27584900 ns

MergeSort       Avg Time                Worst case time
                34401710 ns             36239100 ns

QuickSort       Avg Time                Worst case time
                14758460 ns             18873700 ns

DualPivotSort   Avg Time                Worst case time
                2536170 ns              2894600 ns

RadixSort       Avg Time                Worst case time
                74369340 ns             92470700 ns
```



Sorting strings for 1 time

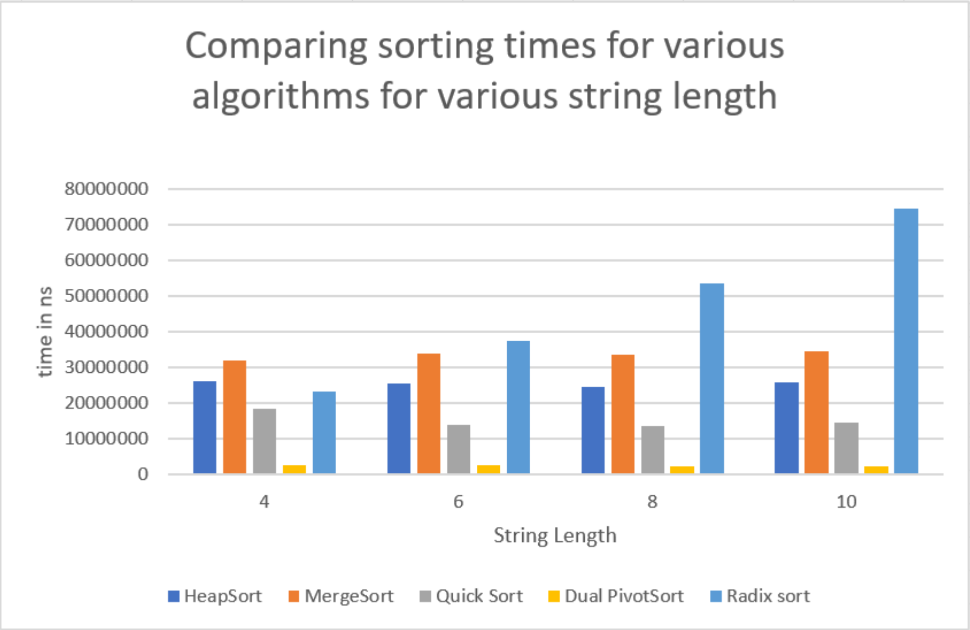
length of Strings	HeapSort	MergeSort	Quick Sort	Dual PivotSort	Radix sort
4	91637000	113837600	60976500	7710800	53297300
6	27029600	33001800	13814800	2911000	43191800
8	18123800	29482000	9176000	887600	14123600
10	25745240	34401710	14758460	2536170	74369340

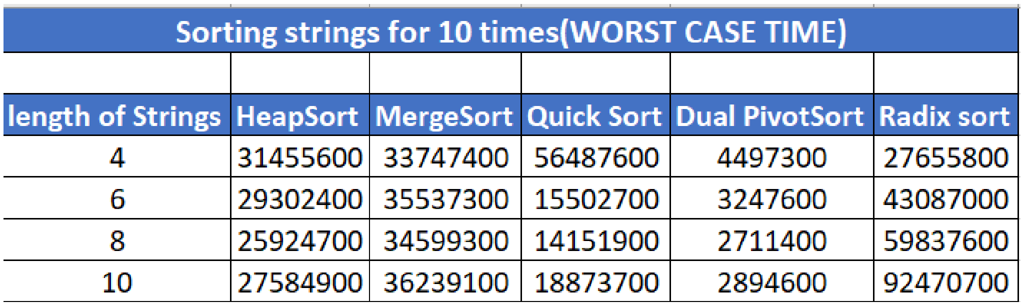




Sorting strings for 10 times

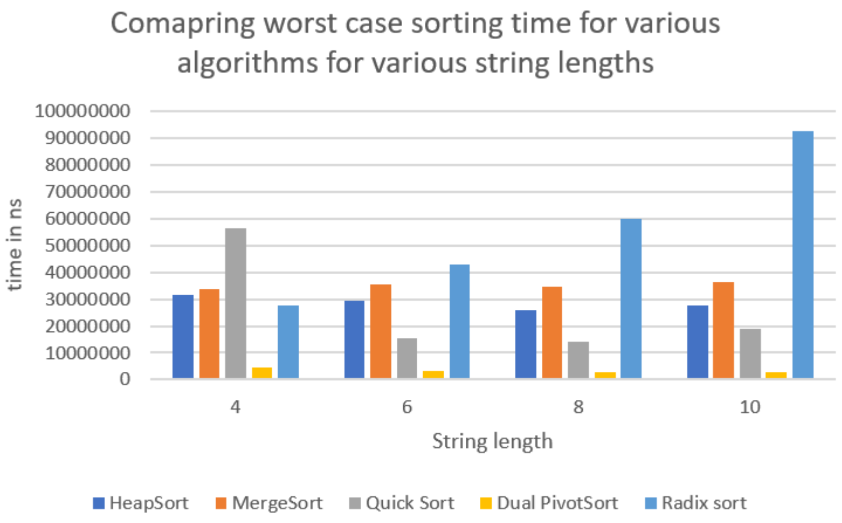
length of Strings	HeapSort	MergeSort	Quick Sort	Dual PivotSort	Radix sort
4	26247600	31944330	18312070	2796340	23221070
6	25614120	33739110	13838430	2659480	37350110
8	24514390	33522660	13521260	2372220	53594740
10	25745240	34401710	14758460	2536170	74369340





Sorting strings for 10 times(WORST CASE TIME)

length of Strings	HeapSort	MergeSort	Quick Sort	Dual PivotSort	Radix sort
4	31455600	33747400	56487600	4497300	27655800
6	29302400	35537300	15502700	3247600	43087000
8	25924700	34599300	14151900	2711400	59837600
10	27584900	36239100	18873700	2894600	92470700



Question 4: Comment on: which sorting method will you use in your applications? in which case? Why?

Solution4:
On comparing the avg time taken by various algorithms, in my opinion we should dual-pivot quick sort for single and repeated sorting as it has taken the least amount of time irrespective of the length of the String. We can even confirm this by comparing the average time with worst case timings across various cases. The graphs shown above for the various cases also depicts the same result.

Question 5: Use the edit distance (class Sequences.java) implementation provided in the source code.

a. Generate 1,000 pairs of random words of lengths 10, 20, 50 and 100.

b. Compute the edit distance for all words and find the average CPU time for each pair.

c. Compare the CPU times obtained for each word length with the running times of the edit distance algorithm.

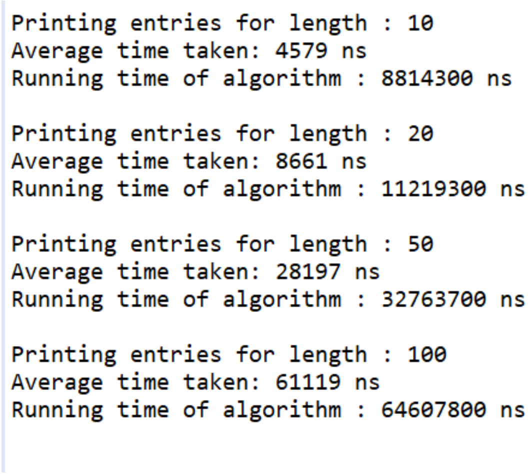
Solution5:
Please refer to project sorting, package solution, class QuestionFive and method main for coded solution

Steps to resolve are as follows:

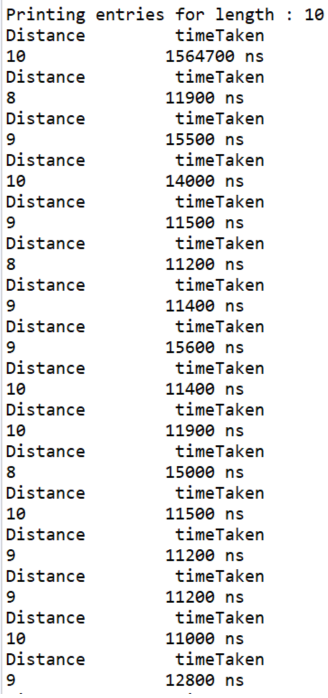
1. Created an array of 1000 random Strings of length 10 using method generateRandomStringArray in class Helper.java
2. For each pair in list, calculate the distance. And calculate the average time (CPU Time).
3. Running time is calculated as the time take to execute whole program.
4. Repeat 1 and 2 for Strings of length 20,50 and 100.

Note: since the output of the program is too long, I have just attached some of the entries for each word length.

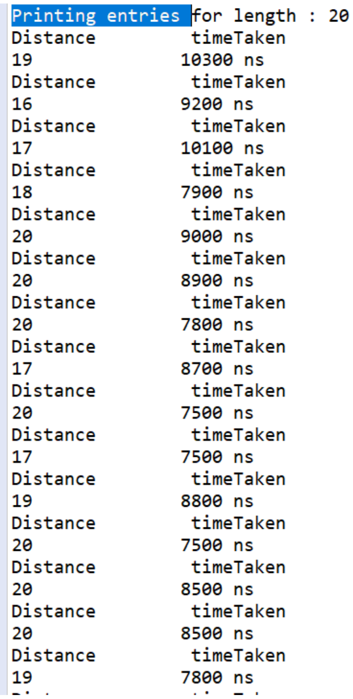
Below are the screenshots of the output:



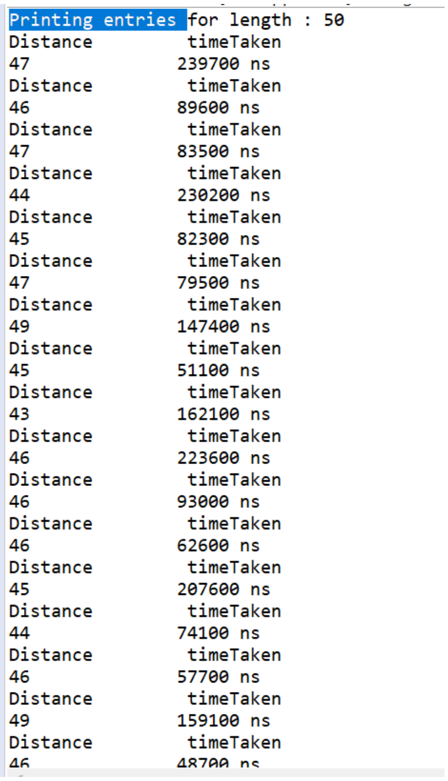
```
Printing entries for length : 10
Average time taken: 4579 ns
Running time of algorithm : 8814300 ns

Printing entries for length : 20
Average time taken: 8661 ns
Running time of algorithm : 11219300 ns

Printing entries for length : 50
Average time taken: 28197 ns
Running time of algorithm : 32763700 ns

Printing entries for length : 100
Average time taken: 61119 ns
Running time of algorithm : 64607800 ns
```



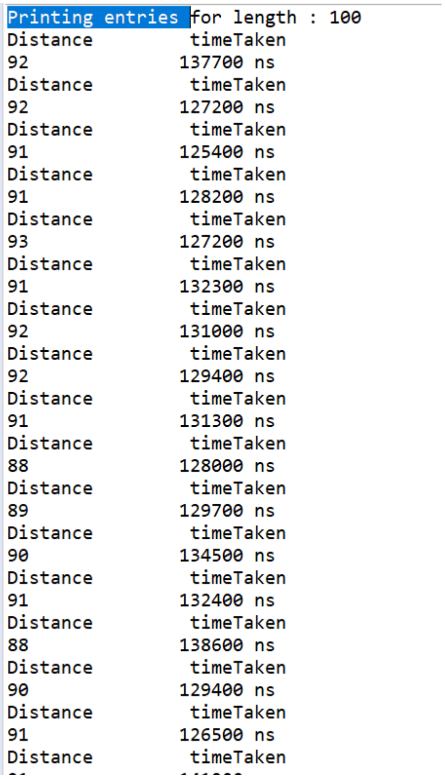
```
Printing entries for length : 10
Distance         timeTaken
10              1564700 ns
Distance         timeTaken
8               11900 ns
Distance         timeTaken
9               15500 ns
Distance         timeTaken
10              14000 ns
Distance         timeTaken
9               11500 ns
Distance         timeTaken
8               11200 ns
Distance         timeTaken
9               11400 ns
Distance         timeTaken
9               15600 ns
Distance         timeTaken
10              11400 ns
Distance         timeTaken
10              11900 ns
Distance         timeTaken
8               15000 ns
Distance         timeTaken
10              11500 ns
Distance         timeTaken
9               11200 ns
Distance         timeTaken
9               11200 ns
Distance         timeTaken
10              11000 ns
Distance         timeTaken
9               12800 ns
```



```
Printing entries for length : 20
Distance         timeTaken
19              10300 ns
Distance         timeTaken
16              9200 ns
Distance         timeTaken
17              10100 ns
Distance         timeTaken
18              7900 ns
Distance         timeTaken
20              9000 ns
Distance         timeTaken
20              8900 ns
Distance         timeTaken
20              7800 ns
Distance         timeTaken
17              8700 ns
Distance         timeTaken
20              7500 ns
Distance         timeTaken
17              7500 ns
Distance         timeTaken
19              8800 ns
Distance         timeTaken
20              7500 ns
Distance         timeTaken
20              8500 ns
Distance         timeTaken
20              8500 ns
Distance         timeTaken
19              7800 ns
```



```
Printing entries for length : 50
Distance         timeTaken
47              239700 ns
Distance         timeTaken
46              89600 ns
Distance         timeTaken
47              83500 ns
Distance         timeTaken
44              230200 ns
Distance         timeTaken
45              82300 ns
Distance         timeTaken
47              79500 ns
Distance         timeTaken
49              147400 ns
Distance         timeTaken
45              51100 ns
Distance         timeTaken
43              162100 ns
Distance         timeTaken
46              223600 ns
Distance         timeTaken
46              93000 ns
Distance         timeTaken
46              62600 ns
Distance         timeTaken
45              207600 ns
Distance         timeTaken
44              74100 ns
Distance         timeTaken
46              57700 ns
Distance         timeTaken
49              159100 ns
Distance         timeTaken
46              48700 ns
```



```
Printing entries for length : 100
Distance         timeTaken
92              137700 ns
Distance         timeTaken
92              127200 ns
Distance         timeTaken
91              125400 ns
Distance         timeTaken
91              128200 ns
Distance         timeTaken
93              127200 ns
Distance         timeTaken
91              132300 ns
Distance         timeTaken
92              131000 ns
Distance         timeTaken
92              129400 ns
Distance         timeTaken
91              131300 ns
Distance         timeTaken
88              128000 ns
Distance         timeTaken
89              129700 ns
Distance         timeTaken
90              134500 ns
Distance         timeTaken
91              132400 ns
Distance         timeTaken
88              138600 ns
Distance         timeTaken
90              129400 ns
Distance         timeTaken
91              126500 ns
Distance         timeTaken
```