<u>Sorting Algorithms Evaluation</u>: Mergesort, Quicksort, Heapsort and Dual-pivot Quick-sort.

❖ *For Integers*

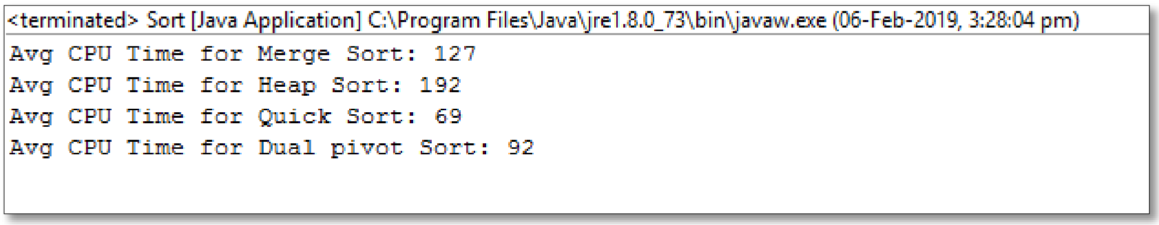
- The main() function of class Sort.java(in Sorting) generates 100,000 random keys of type long by using nextLong() method and sort them by methods**: mergeSort()**, **heapsort()**, **quicksort()** and **sort()** 100 times. The random keys generated are different every time.
- For average CPU time, I have calculated the total time taken to sort and divided the total by 100. Table 1 shows the results:

Sorting Algorithm	Mergesort	Quicksort	Heapsort	Dual-pivot Quicksort
Time taken (milliseconds)	127	69	192	92

Table 1: Average CPU time taken to sort

- The average-case time complexities for Mergesort, Quicksort, Heapsort and dual-pivot Quicksort is **O(n log n**). Among them, Quicksort is fast and Heapsort is slow according to results in Table 1.
- Mergesort recursively solves two subproblems and uses extra array, thus takes additional linear time in copying throughout the algorithm. Like Mergesort, Quicksort also follows the same approach. Though Quicksort is fast, because the partitioning into two subproblems here can be done **in place** and very efficiently.
- Quicksort and dual-pivot Quicksort both have somewhat related average times. However, dual-pivot Quicksort is little faster than Quicksort when data is huge.

<u>Output:</u>



```
<terminated> Sort [Java Application] C:\Program Files\Java\jre1.8.0_73\bin\javaw.exe (06-Feb-2019, 3:28:04 pm)
Avg CPU Time for Merge Sort: 127
Avg CPU Time for Heap Sort: 192
Avg CPU Time for Quick Sort: 69
Avg CPU Time for Dual pivot Sort: 92
```

❖ *For String:*

- The main() function of class RadixSort.java generates the random string of different length and sort them by the five methods mentioned.
- The average CPU time is computed by total time taken to sort divided by 10. The average times to sort different length of strings are shown in Table 2.

	Average time taken to sort in milliseconds				
String Length	MergeSort	QuickSort	HeapSort	Dual-pivot Quicksort	Radix Sort
4	269	157	250	279	133
6	197	143	251	133	155
8	209	154	266	141	213
10	208	152	266	149	272

Table 2: Average time of sorting

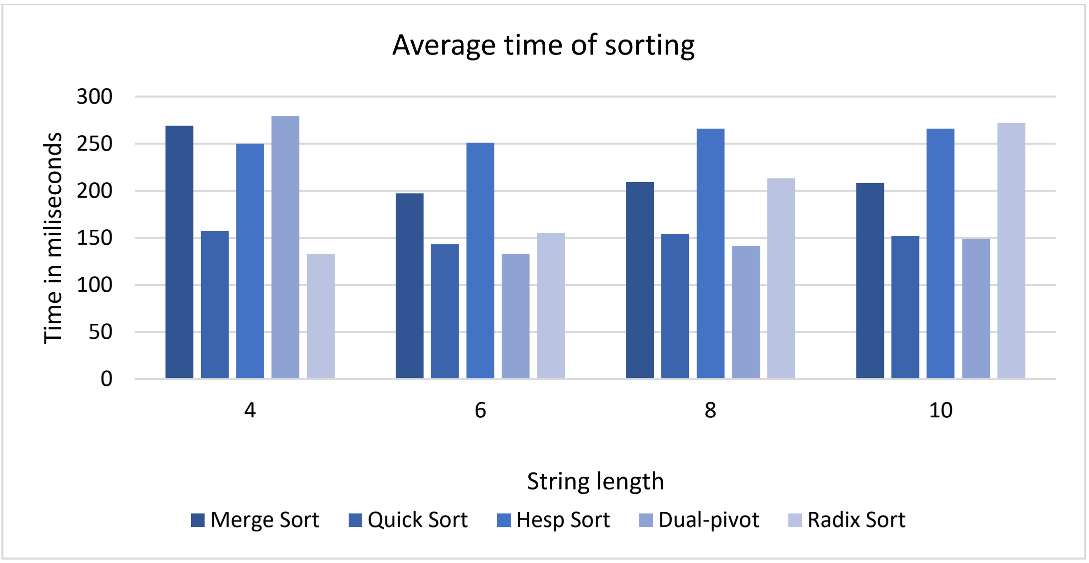
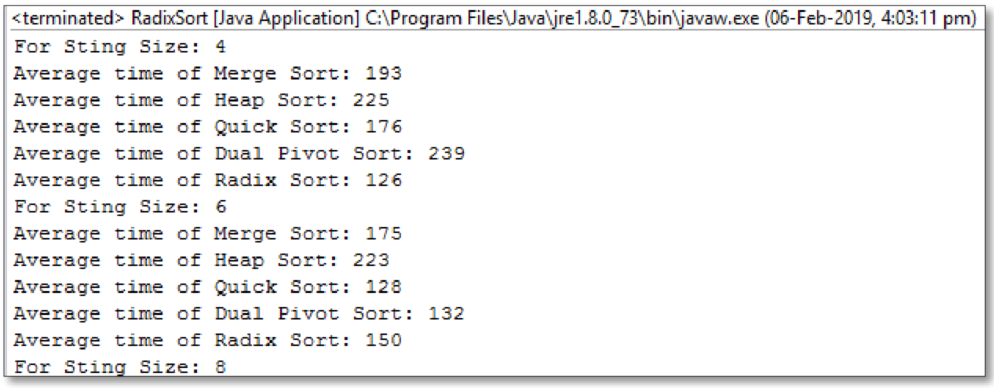


Figure 1: Sorting average time versus string length

- When the string length is small, radix sort runs fast and dual-pivot Quicksort runs slow but when string is large in length, dual pivot Quicksort sorts fast and Radix sort runs slow (Figure 1). Mergesort, Quicksort, HeapSort and dual-pivot QuickSort are comparison based sorting algorithms, whereas Radix sort is a stable sorting algorithm.
- Comparison based algorithm's average-case and worst-case time complexity is O(n log n), except for quick sort and dual pivot quick sort which has worst case running time: O(n^2). For quick sorts, the worst-case occurs when pivot selected is either the minimum or maximum number, but this can be made exponentially unlikely with a little effort.

- The worst-case running time of radix sort is O(d(n+N)), where d is number of passes, n is number of keys and N is number of buckets. Now depending on the situation when d is log n and N is O(n), radix sort runs in O(n log n) and when d is constant and N is O(n), it runs in O(n).

Output:



```
<terminated> RadixSort [Java Application] C:\Program Files\Java\jre1.8.0_73\bin\javaw.exe (06-Feb-2019, 4:03:11 pm)
For Sting Size: 4
Average time of Merge Sort: 193
Average time of Heap Sort: 225
Average time of Quick Sort: 176
Average time of Dual Pivot Sort: 239
Average time of Radix Sort: 126
For Sting Size: 6
Average time of Merge Sort: 175
Average time of Heap Sort: 223
Average time of Quick Sort: 128
Average time of Dual Pivot Sort: 132
Average time of Radix Sort: 150
For Sting Size: 8
```

❖ *Which sorting method to use in your applications? in which case? Why?*

- When one has to sort **integers**, one should use quick sort as its performance is fastest as per the experimental results in Table 1. But when data is very large, dual-pivot QuickSort is better choice.
- When one has to sort **strings,** which sorting algorithm to use depends on the length of string. Thus for strings: Radix sort works best when the string length is **smaller**, and for strings with larger length, comparison based dual-pivot Quicksort would be my choice. (**Table 2**).
- As the length of string increases, radix sort takes more time for sorting. The reason here is, suppose for example, when string size increases, there is two possible ways by which it can be sorted:

1. Consider one character at a time, but then the number of passes increases.
2. Consider let's say two characters at a time, so now bucket size becomes N^2.

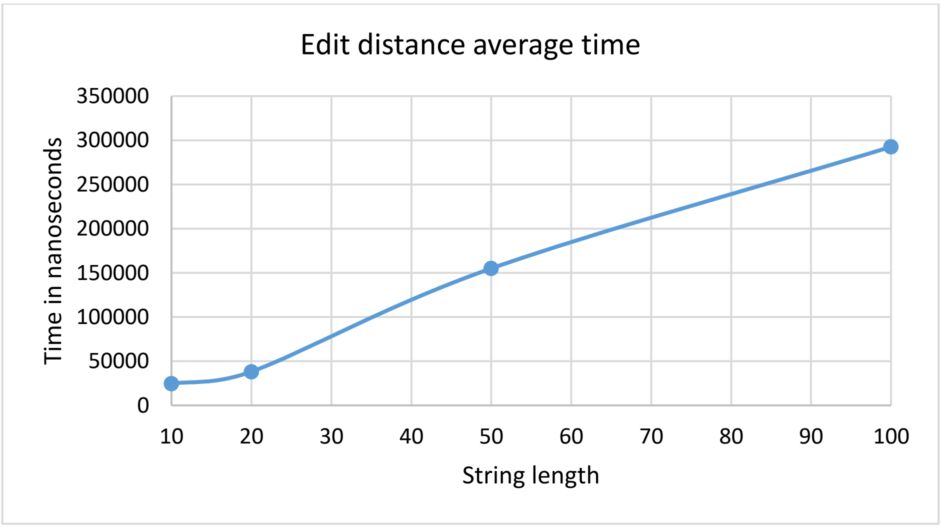
Thus, it is better to use comparison-based sorting algorithm when the number of strings are **small and size is large**.

***String Comparison*: Edit distance**

- The main() function of class Sequences.java (in algorithmDesign) generates 1,000 pair of random words of lengths 10, 20, 50 and 100 and computes the edit distance for all words by function editDistance().
- The total time taken to calculate the edit distance is divided by 1000 for the average CPU time of each pair. CPU times obtained in nanoseconds is as shown in Table 3.

String length	Time taken in nanoseconds
10	24638
20	38210
50	154987
100	292586

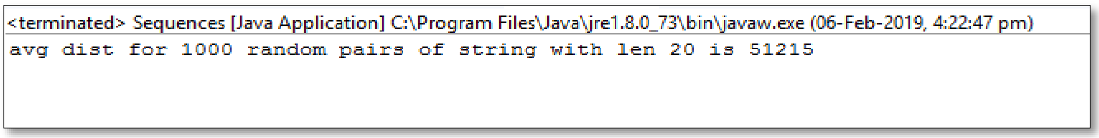
Table 3: Average time of computing edit distance



. **Figure 2: Average time of computing edit distance versus string length**

- From the graph, it is clear that running time of edit distance algorithm is **$O(n^2)$**. For two strings of length n and m, complexity is O(nm) for edit distance. Here the length of both strings is same thus, complexity is $O(n^2)$. As the string length increases, the average time taken grows in a quadratic manner.

Output:



```
<terminated> Sequences [Java Application] C:\Program Files\Java\jre1.8.0_73\bin\javaw.exe (06-Feb-2019, 4:22:47 pm)
avg dist for 1000 random pairs of string with len 20 is 51215
```