Practical 5 – Efficent Sorting Algorithms

Timing results and excel sheet for performance can be found here in the repo -

C:\Users\Colmr\Desktop\algo\algorithms20290-2021-repository-CFR2000\Practical Resources\Analysis-Results\wk5

```
******************************
Insertion sort testing time
******************************
**********


Test 0 - 10 inputs
 elapsed time = 0.0

************************ Next
Test
******************************
********

Test 1 - 100 inputs
 elapsed time = 0.031

************************ Next
Test
******************************
********

Test 2 - 1000 inputs
 elapsed time = 0.047

************************ Next
Test
******************************
********

Test 3 - 10000 inputs
 elapsed time = 0.063

************************ Next
Test
******************************
********

Test 4 - 100000 inputs
 elapsed time = 1.515
```

```
******************************
Merge Sort testing Time
******************************
**********

Test 0 - 10 inputs
 elapsed time = 0.0

************************ Next
Test
******************************
**********

Test 1 - 100 inputs
 elapsed time = 0.078

************************ Next
Test
******************************
**********

Test 2 - 1000 inputs
 elapsed time = 0.094

************************ Next
Test
******************************
**********

Test 3 - 10000 inputs
 elapsed time = 0.11

************************ Next
Test
******************************
**********

Test 4 - 100000 inputs
 elapsed time = 0.188
```

1. Q1. Implementing Mergesort from pseudo-code and comparing Mergesort to Insertion Sort for increasing input sizes

I can conclude from comparing insertion sort to merge sort that

- Insertion sort preforms better (quicker time) when dealing with smaler input sizes to sort

Insertion sort was faster than merge sort in 4/5 tests but was slower when input size was 100000.

Merge Sort is efficent for sorting larger input sizes, Slower comparative to the other sort algorithms for smaller tasks.

uses more memory space to store the sub elements of the initial split list.

Interestingly a super sort method could be created by implementing these two sorts together in the same algorithm.

Big O-Notation

Algorithm	***Best***	***Average***	***Worst***
Merge Sort	O(n*log(n))	O(n*log(n))	O(n*log(n))
Merge Sort	O(n)	O(n^2)	O(n^2)

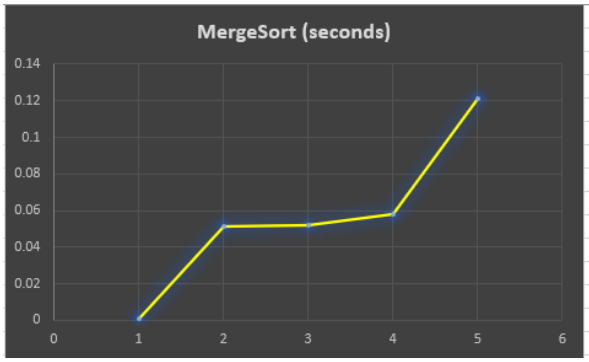
Q2. Merge Sort Enhanced

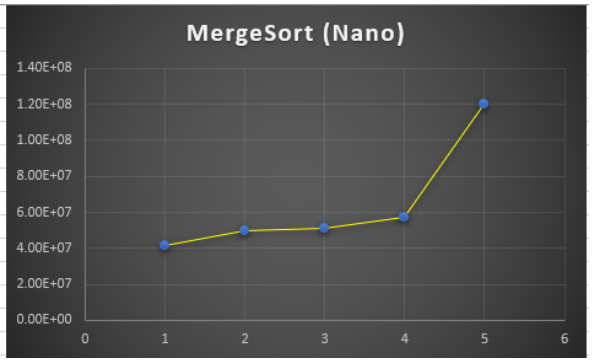
```
public class MergeSortEnhanced {


    public static class mergeSortEnhanced {

        public static boolean isSorted(int[] arr){
            boolean isSorted = true;
            for (int i = 0; i<arr.length-1; i++){
                if (arr[i] > arr[1+1]){
                    isSorted = false;
                    return isSorted;
                }
            }
            return isSorted;
        }

        public static void merge(int[] a, int low, int mid, int hi) {
            //copy the array a to an aux array


            int l = mid - low + 1;
            int r = hi - mid;

            int[] leftArray = new int[l];
            int[] rightArray = new int[r];

            for (int i = 0; i < l; ++i) {
                leftArray[i] = a[low + i];
            }

            for (int j = 0; j < r; ++j) {
                rightArray[j] = a[mid + 1 + j];
            }
```

```
            int i = 0, j = 0;
            int k = low;
            while (i < l && j < r) {
                if (leftArray[i] <= rightArray[j]) {
                    a[k] = leftArray[i];
                    i++;
                } else {
                    a[k] = rightArray[j];
                    j++;
                }

                k++;

            }

            while (i < l) {
                a[k] = leftArray[i];
                i++;
                k++;
            }

            while (j < r) {
                a[k] = rightArray[j];
                j++;
                k++;
            }
        }


        // recursive
        public static int[] sortEnhanced(int[] array, int left, int right)
{

            boolean isSorted = isSorted(array);
            if (isSorted){
                return array;
            } with smaller inputs than merge sort
            if (array.length <= 100){
                Sorts_starter_code smallCase = new Sorts_starter_code();
                smallCase.insertionSort(array);
                StdOut.println("Input size is < 100, so calling insertion
sort on array!");
                return array;
            }

            if (left < right) {
                int mid = (left+right)/2;
                // sorts first and second halves
                sortEnhanced(array, left, mid); //sorting left
                sortEnhanced(array, mid + 1, right); // sorting right
                // merge the sorted halves
                if (array[mid] > array[mid+1]) {
//                      StdOut.println("mid is greater than mid+1 s0 merge()
is needed.");
                    merge(array, left, mid, right);
                }
            }
            return array;
        }
}
```

Q3. Comparing 3 algorithms.

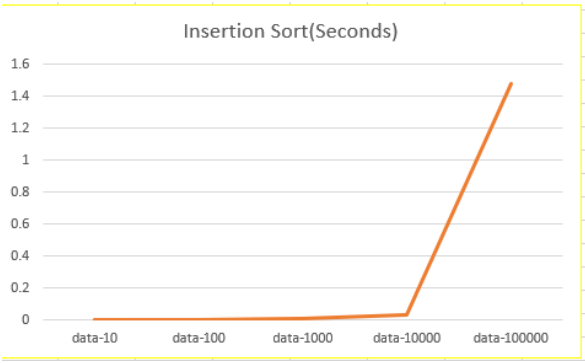
Algorithm	Inputs	Time(Seconds)	Time(nano)
Merge Sort	data-10	0.001	4.16E+07
	data-100	0.051	4.95E+07
	data-1000	0.052	5.14E+07
	data-10000	0.058	5.74E+07
	data-100000	0.121	1.20E+08

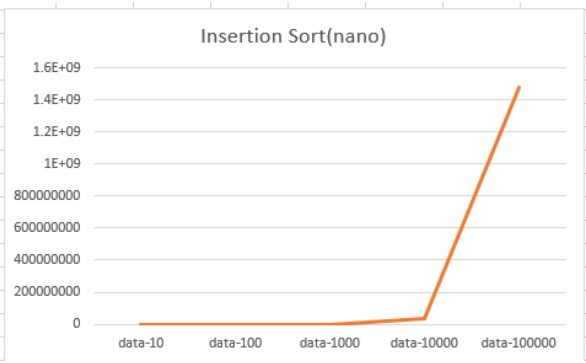




Conclusion: Runs in logarithmic time, More efficent on larger inputs for sorting.

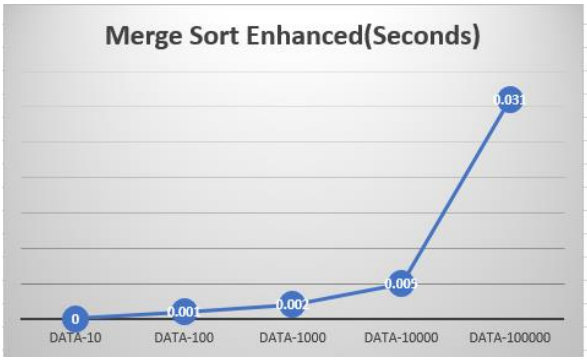
Algorithm	Inputs	Time(Seconds)	Time(nano)
Insertion Sort	data-10	0	0.12537
	data-100	0.001	0.19577
	data-1000	0.009	0.97051
	data-10000	0.036	3.64E+07
	data-100000	1.475	1.48E+09

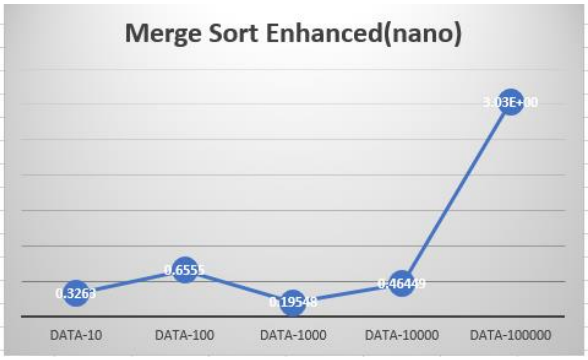




Conclusion: Runs in n-squared time, works very well on smaller input values to sort for ex. Inputs below 1000.

Algorithm	Inputs	Time(Seconds)	Time(nano)
Merge Sort Enhanced	data-10	0	0.3263
	data-100	0.001	0.6555
	data-1000	0.002	0.19548
	data-10000	0.005	0.46449
	data-100000	0.031	3.03E+00





Conclusion: A super sort, combining the best components of insertion and merge sort together. Time is cut by 10-15% by swithcing to insertion sort for smal subarrays.

Quick Questions

1. Mergesort guarantees to sort an array in _____ time, regardless of the input:

   A. Linear time
   B. Quadratic time
   C. Linearithmic time
   D. Logarithmic time

2. The main disadvantage of MergeSort is:

   A. It is difficult to implement
   B. It uses extra space in proportion to the size of the input
   C. It is an unstable sort
   D. None of the above

3. Merge sort makes use of which common algorithm strategy?

   A. Dynamic Programming
   B. Branch-and-bound
   C. Greedy approach
   D. Divide and conquer

4. Which sorting algorithm will take the least time when all elements of the input array are identical?

   A. Insertion Sort
   B. MergeSort
   C. Selection Sort
   D. Bogo Sort

5. Which sorting algorithm should you use when the order of input is not known?

A. Mergesort
B. Insertion sort
C. Selection sort
D. Shell sort