Name: Morad E.

ADS Assessed Exercise 1

Guide to running the code:

Import the zip file as a project into the Eclipse Workspace, open up the package called ex1 and run the contained class files.

==========

Part 1

a) Implemented Quicksort using the pseudocode included in the lecture notes. Comments are provided in the code.

b) Implemented a variant of Quicksort which returns without sorting subarrays with fewer than k elements and then uses Insertion-sort to sort the entire nearly-sorted array, based off the lecture notes.

c) Implemented a variant of Quicksort using the median-of-three partitioning scheme.

d) Implemented a three-way-quicksort. For this implementation, a partition of three is made use of.
The left part is configured to be the part which is less than the pivot, then an assignment is made for scanning the array from left to right; the right part is assigned to be greater than the pivot. The pivot value is given the value of the array's right element. In the case that the while loop evaluates to true, depending on whether the indexed array item is greater than, equal to, or less than the pivot value, swapping occurs with appropriate operations taking place to increment or decrement variables. (Although, in the case of two values being compared equaling each other, there is no need to swap - simply move to the next item.)
After the while loop terminates, an integer array is returned containing the values in variables for the left and right part. In the threewayQuickSort method, there is a conditional check to ensure the left part is less than the right part, and if the check evaluates to true, then the partitionThree method is called, using the array's left and right values as parameters to be stored in an integer array.
Ultimately, since three regions are being considered (i.e, this implementation includes elements equal to the pivot), the method makes three calls to itself (threeWayQuickSort), based on elements less than, equal to and greater than the pivot. Finally, the sorted array is returned.

Part 2

For each implementation, a method named isSorted was used to check that the algorithms sorted properly (which eventually all managed to). Sample times taken to sort for the different algorithms has been collated into a table, shown below. Additionally, sample runs were performed for Insertionsort and Mergesort and included as well.

Time taken to sort (ns)					
Quicksort	QuickInsertionSort	Medianof3Quicksort	3WayQuicksort	Insertionsort	Mergesort
46,700	13,800	47,000	10,400	7,900	23,600
87,900	74,400	251,900	128,400	20,500	73,900
330,800	298,700	222,500	82,000	114,800	154,800
3,899,200	18,073,700	2,155,000	1,159,500	4,342,000	1,176,000
63,197,800	65,704,900	12,385,900	7,153,800	228,494,200	25,825,900
160,391,600	208,633,400	150,033,400	65,670,200	33,231,924,900	130,519,400
346,398,600	439,194,200	311,473,000	158,853,800	159,711,067,200	221,725,600
1,323,765,900	972,226,000	1,150,516,400	33,589,100	31,550,430,500	64,019,100

Insertion sort has a worst case time complexity of n**2 (when the array is reverse sorted) but a best case time complexity of n (when the array is already sorted). Merge sort has a best and worst case time complexity of nlogn; for Quicksort, in its best case (when partioning happens to occur in the middle so that subarrays are equal in size), it's best case is similar to Mergesort's of nlogn from the recurrence equation T(n) = 2T(n/2) + O(n) [which is the cost of partioning]. However, the worst case for Quicksort happens when partitioning is unbalanced in each recursive call (one array has n-1 elements but the other has none), with worst case time complexity of n**2 (when array is already sorted). This is not as good as insertion sort's (big-oh) complexity of n when it handles sorted arrays.
Overall from the data, using insertion sort is rather good for smaller data sets (and it's refined further when QuickInsertionSort makes use of it). For larger data sets, Quicksort performs very well (as long it is not in the worst case where the array is already sorted); the Median of Three Quicksort can also perform very well, and likewise Three Way Quicksort can be really fast. Depending on the pivot partitioning, Quicksort's results can improve a lot. Merge sort also does very well furthermore, and is faster than the normal Quicksort, especially with larger data sets.

Part 3

For the implementation to generate pathological input sequences, use of a method called pathologicalInput is made First, the pathological array is initialised and an index value counter set up. Processing inside a for loop, using conditionals, even numbers are added to the left positions of the pathological array while odd numbers are added to the right positions. Finally an array ascending in even numbers and then descending in odd numbers is returned, which results in more recursive passes being needed to be made when sorting, enhancing sorting complexity.