Assignment 2 - Group 5

The algorithms were implemented in three different python files: inssort.py, quicksort.py and heapsort.py. These are called in the sort.py file where rangen is also called (line 111), which is why the sort.py file is included in our solution. To test the program the file test.sh is needed. After running test.sh, type in 'python sort.py' on the command line and the different algorithms will run and their time will be printed.
The three different sorting algorithms were analyzed with four different sizes of the array A, A is a random and unsorted array. The results are presented in the table below.

	rangen(1000)	**rangen(10 000)**	**rangen(20 000)**	**rangen(30 000)**
Insertion Sort (s)	0.168652534485	14.2838072777	54.947784423828	131.571787596
Quicksort (s)	0.016951799393	0.10571622849	0.2275168895721	0.562013149261
Heapsort (s)	0.023307085037	0.25786089897	0.9251725673676	1.781873941422

Table 1: Time consumption of the three different sorting algorithms

It is clear from the table that the sorting time for the same unsorted array A depends largely on the method.

Insertionsort: Easy to implement, quick and gives good performance when using small arrays. As we can see in *Table 1* this method is really slow in all of the chosen ranges compared to the other two. This is due to its non-recursive nature.

Quicksort: Quick for small and somewhat big lists. As we can see the running time is the fastest one for all different ranges we tried. Also this method runs faster as the range gets bigger in comparison with the other two. Accordingly this method is the quickest for both big and small lists.

Heapsort: Effective and quick when sorting really big lists. Although the algorithm is slower than quicksort on the ranges we tried, it has a better worst-case performance than both insertion and quick sort (see Table 2). This, and the fact that the best and worst case have the same time-complexity makes the algorithm more reliable for huge arrays than the other methods.

Algorithm	Best-case	Average-case	Worst-case

Insertion sort	$\theta(n)$	$\theta(n^2)$	$\theta(n^2)$
Quick sort	$\theta(nlog(n))$	$\theta(nlog(n))$	$\theta(n^2)$
Heap sort	$\theta(nlog(n))$	$\theta(nlog(n))$	$\theta(nlog(n))$

Table 2: Time complexity

From this table we can further discuss and compare the methods.

Insertion sort: As said earlier, insertion sort is useful when dealing with small arrays. This finding is proved once again when looking at the time complexity for average-case that insertion-sort has $\theta(n^2)$, and therefore it's generally only good for smaller ranges. This is due to its algorithm that scans the whole list after each time the list gets one sorted item. The method is an in-place method, as the other two.

Quick sort: This method is an in-place algorithm, and also uses a divide- and-conquer method which differs från Heapsort. The way the time complexity varies compared to Heap sort is in its worst case, where the worst case in Quicksort requires the *partition* algorithm to run in each step and together with *QuickSort* runs with time complexity $\theta(n^2)$. When discussing comparison, the average case for time complexity is the same, however as earlier concluded a faster algorithm when sorting lists.

Heap sort: Due to the algorithm for Heap sort and the fact that this method does care for the initial look it has the same time complexity in all of the cases. Like Quick sort, Heap sort is an in-place algorithm but without divide and conquer algorithm, also without the high time complexity for the worst case.