**EXERCISE 1**

**Team Members**: Valentino Di Cianni, Elia Casetta, Ionut Hodor

**Tests made on:**

Model name: iMac

Model identificator: iMac11,3

Operating system: OS X El Capitan, versione 10.11.4

Processor name: Intel Core i7

Processor speed: 2,93 GHz

Processor number: 1

Core number: 4

Cache L2 (per Core): 256 KB

Cache L3: 8 MB

Memory: 12 GB

Processor interconnection speed: 4,8 GT/s

Boot ROM version: IM112.0057.B03

SMC version: 1.59f2

**Expectations before tests:**

Insertion sort:

#Best case: O(n)

#Middle case: O(n^2)

#Worse case: O(n^2)

Quick Sort:

#Best case: O(n log n)

#Middle case: O(n log n)

#Worse case: O(n^2)

Merge Sort:

#Best case: O(n log n)

#Middle case: O(n log n)

#Worse case: O(n log n)

Heap Sort:

#Best case: O(n log n)

#Middle case: O(n log n)

#Worse case: O(n log n)

Due to algorithms complexity, we expect a faster sorting on quick sort, heap sort and merge sort

on normal conditions, while a much more slower sorting on insertion sort.

For avoiding quick sort’s worse case, the pivot is randomly choosen, so this particular scenario cannot

happen.

We also expect similar times for heap, merge, quick sort in any case.

We don’t know which field is faster for sorting, but we suppose they are(from faster to slower):

int > float > string

**Tests:**

Starting tests on:Insertion sort

.

test on null array passed in 0.00001 seconds

.

test on empty array passed in 0.00000 seconds

.

test on all equal element array passed in 0.26549 seconds

.

test on full array\_int 🡪 FAIL (it lasts more than 10 minutes)

.

test on full array\_string 🡪 FAIL (it lasts more than 10 minutes)

.

test on full array\_float 🡪 FAIL (it lasts more than 10 minutes)

.

(all equal elements)

Starting tests on:Quick sort

.

test on null array passed in 0.00000 seconds

.

test on empty array passed in 0.00000 seconds

.

test on int array passed in 9.07783 seconds

.

test on string array passed in 6.75476 seconds

.

test on float array passed in 9.15217 seconds

.

test on all equal element array passed in 5.46141 seconds

6 tests passed in 59.60883 seconds

(this includes the time for free the memory)

Starting tests on:Merge sort

.

test on null array passed in 0.00000 seconds

.

test on empty array passed in 0.00000 seconds

.

test on int array passed in 20.79862 seconds

.

test on string array passed in 18.91609 seconds

.

test on float array passed in 21.11045 seconds

.

test on all equal element array passed in 11.52904 seconds

6 tests passed in 114.66720 seconds

(this includes the time for free the memory)

Starting tests on:Heap sort

.

test on null array passed in 0.00000 seconds

.

test on empty array passed in 0.00000 seconds

.

test on int array passed in 44.45782 seconds

.

test on string array passed in 32.78064 seconds

.

test on float array passed in 44.34304 seconds

.

test on all equal element array passed in 0.76823 seconds

6 tests passed in 165.54058 seconds

**Conclusions:**

As expected, insertion sort does not finish in 10 minutes, but it lasts much more.

The difference of time between quick sort and heap /merge sort It is due to the fact that quick sort orders the array, while merge/heap sort create temporarily structures to store results, and that costs more time. Anyway, they are definitely faster than insertion sort.

The all equal element array is the fastest on all sorting algorithms.

The compare string function seems to be unexpectedly faster than “int” and “float” compare function, but after some tests we found out that that is not faster, but the difference in timing is due to many duplicates in records.csv ‘s string field. In fact, after a quick parsing, results are incredible: the firsts 10 words in string field are repeated around 20K / 60K times. For example the word “Clïò”, which is quite unusual, is repeated around 20K times. All this reps make the sorting in String field the most efficient and fast.