Luvisutto Alberto, Algorithmic design exam

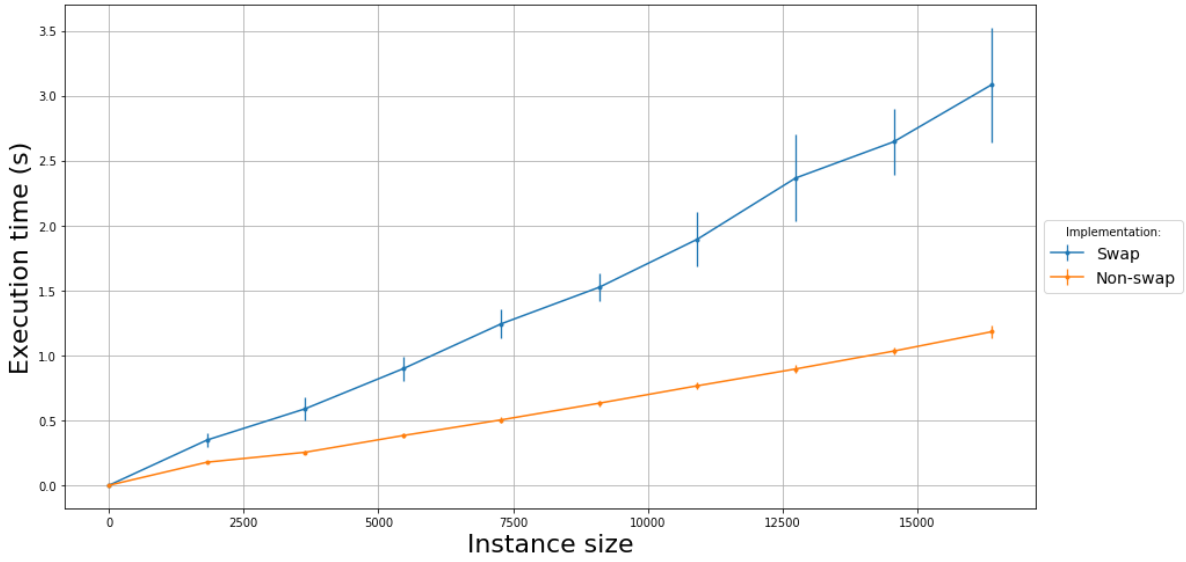
Homework: Binary Heaps_02

Ex.1

By modifying the code written during the last lessons, provide an array-based implementation of binary heaps which avoids to swap the elements in the array A

I implemented the "non-swap" version of binary heaps as suggested in the hint. The most important modifications performed are in the functions `decrease_key()`, `extract_min()`, `insert_value()` and, of course, `swap_keys()`. Some macros have been changed too, together with the addition of one more element to the `bin_heaps` struct, in order to keep the count of the removed elements from the heap, to avoid indexing problems.

I report here a graph containing the comparison in terms of execution time for `test_delete_min` between the swap and the "non-swap" version.



It is clearly visible that in the non-swap implementation we have an improvement in execution time. Another observation that can be done is that there is much less variability between different runs, as visible from the error bars plotted in the graph.

Ex.2

Consider the algorithm [see 04_Homework2 pdf file] where A is an array. Compute the time-complexity of the algorithm when:

- `build`, `is_empty` $\in \Theta(1)$, `extract_min` $\in \Theta(|D|)$;

In this case the function `extract_min`, whose complexity is $\Theta(|D|)$, is executed $|D|$ times. Since the complexity of `is_empty` is $\Theta(1)$, it means that the `while` block complexity is $|D| * \Theta(|D|) = \Theta(|D|^2)$. For this reason, the `build` complexity is not significant and the total complexity of the algorithm is $\Theta(|D|^2)$.

- `build` $\in \Theta(|A|)$, `is_empty` $\in \Theta(1)$, `extract_min` $\in O(\log |D|)$;

In this case, using a similar reasoning, the `while` block complexity is $|D| * O(\log |D|) = O(|D| \log |D|)$.

Since, in general, we don't know how big $|A|$ is compared with $|D|$, we could write the final complexity as $O(|A| + |D| \log |D|)$.

Instead, if we assume that $|A| = |D|$ (as it probably is from the way the algorithm is written) we can write the final complexity as $O(|D| \log |D|)$.