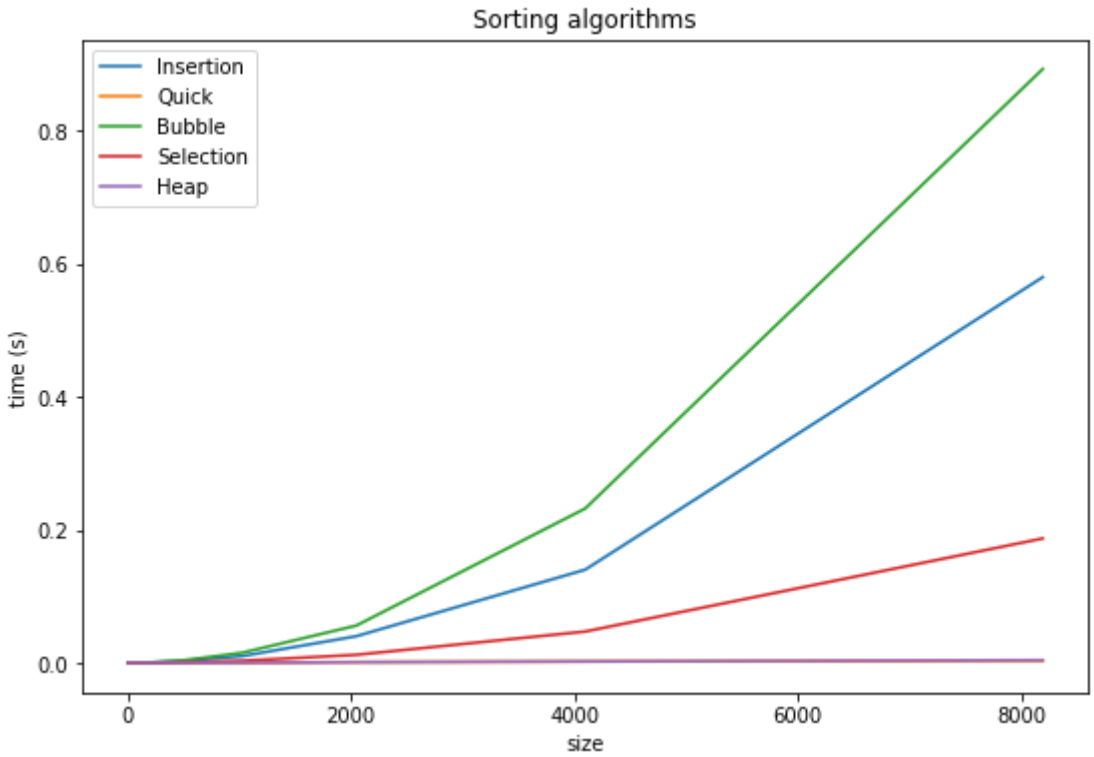
Sorting: Homework 1

Ex. 1

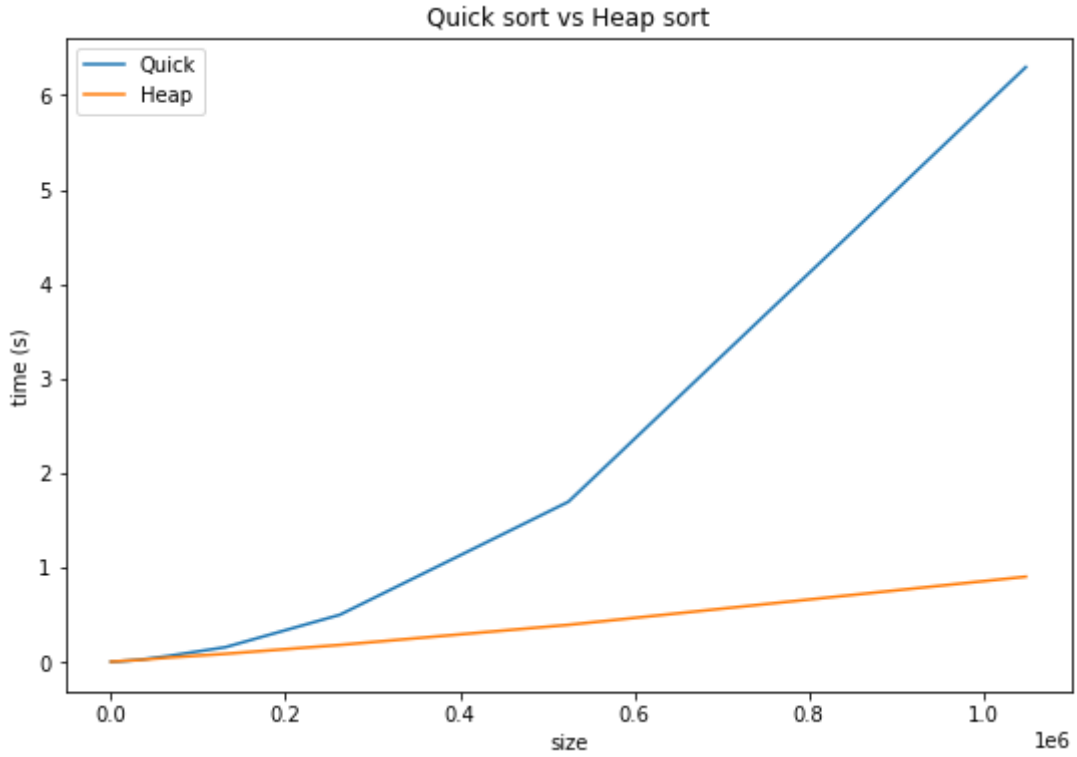
The code for `INSERTION SORT`, `QUICK SORT`, `BUBBLE SORT`, `SELECTION SORT` and `HEAP SORT` can be found, together with the codes required in the second homework about sorting, in the subfolder `src`. Instructions for compilation can be found in the file `README.md`.

Ex. 2

The following plot shows the execution time of 5 different sorting algorithms (insertion sort, quick sort, bubble sort, selection sort and heap sort) on randomly initialized arrays of variable size (varying from 2 to 2^{13}).

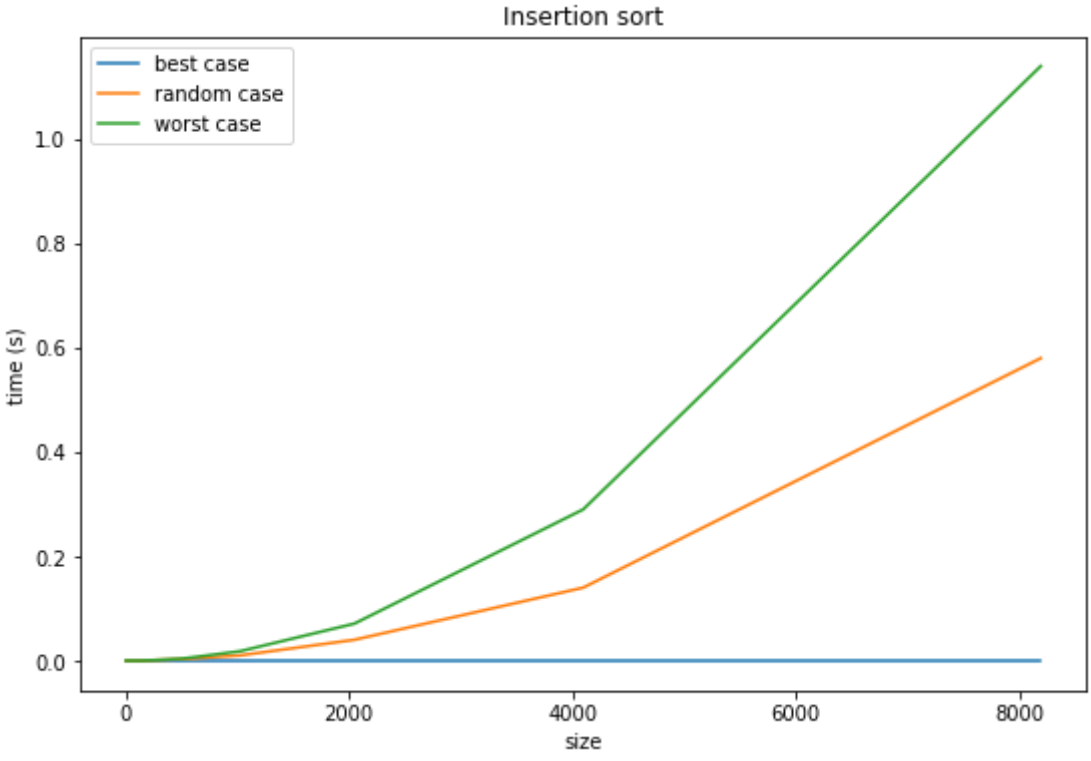


The worst performing algorithms are bubble sort, insertion sort and selection sort, which all have complexity in $O(n^2)$ and actually show some quadratic behaviour in times. On this scale, the curves for heap sort and quick sort are almost indistinguishable from zero and between themselves, so a specific plot may be useful:



From this plot, once again obtained running on random arrays, we can see that for higher sizes of the input heap sort outperforms quick sort.

Insertion sort shows a great variability in running time with respect to the ordering of the input array. In fact, in the worst case (reverse sorted array), it yields a complexity of $\Theta(n^2)$, while in the best case (already sorted array) the complexity is $\Theta(n)$. These theoretical results are confirmed by the execution times reported in the following plot.



We can see that the worst case curve has a sharp quadratic increase, while the best case one is almost flat on 0 and the random case one lies between the two.

Ex. 3

A.

In general terms, heap sort takes time $O(nlog(n))$ on an array of length n, since it requires n calls to `EXTRACT_MIN`, which is $O(log(i))$ where i is the variable number of nodes in the heap.

So, it is false that heap sort takes time $O(n)$. However, there is a "best case scenario" in which we are trying to sort an array containing all equal elements. In this case, since heap property is automatically verified in any configuration, `EXTRACT_MIN` takes time $\Theta(1)$ (because `HEAPIFY` performs just one swap) and then heap sort takes time $\Theta(n) \subset O(n)$.

B.

Considering the scenario of the previous point, heap sort takes time $\Theta(n) \subset \Omega(n)$, so the statement is true.

C.

The worst case complexity for heap sort is $O(nlog(n))$. In fact, as briefly discussed in point A, in the worst case there are n calls to `EXTRACT_MIN`, each of which takes time $O(log(i))$.

D.

The worst case scenario for quick sort happens when the chosen pivot is always the smallest or largest element of the array. In this case, quick sort takes time $O(n^2)$. Since $O(n^2) \subset O(n^3)$, the statement is formally true.

E.

In the best and average cases, quick sort takes time $\Theta(nlog(n))$, while in the worst case (see point D) the complexity is $\Theta(n^2)$.

F.

Bubble sort takes time $\Omega(n)$. In fact, its complexity is $\Theta(n^2) \subset \Omega(n)$.

G.

As stated in the previous point, the complexity of bubble sort is $\Theta(n^2)$. This is because it requires two nested loops over the array to be sorted.

Ex. 4

The equation

$$T(n) = \begin{cases} \Theta(1) & \text{if n=32} \\ 3 \cdot T(\frac{n}{4}) + \Theta(n^{\frac{3}{2}}) & \text{otherwise} \end{cases}$$

can be solved using a recursion tree.

Choosing $cn^{\frac{3}{2}}$ as a representative for $\Theta(n^{\frac{3}{2}})$, the root costs $cn^{\frac{3}{2}}$ and a generic node at level i costs $c\frac{n^{\frac{3}{2}}}{8^i}$, with the exception of the leaves, which cost $\Theta(1)$.

The height h of the tree is such that $\frac{n}{4^h} = 32 \implies h = log_4(\frac{n}{32})$. Level i contains 3^i nodes.

It follows that:

$$T(n) = \sum_{i=0}^{log_4(\frac{n}{32})-1} c(\frac{3}{8})^i n^{\frac{3}{2}} + 3^{log_4(\frac{n}{32})}\Theta(1) = cn^{\frac{3}{2}} \sum_{i=0}^{log_4(\frac{n}{32})-1} (\frac{3}{8})^i + \Theta(n^{log_4(3)}) \leq$$

$$\leq \frac{8}{5}cn^{\frac{3}{2}} + \Theta(n^{log_4(3)}) \in O(n^{\frac{3}{2}})$$

To obtain the last inequality the sum of the geometric series was taken as an upper bound for the partial sum of the series.

Moreover, since the first recursive call contributes a cost of $\Theta(n^{\frac{3}{2}})$, $T(n)$ must be lower bounded by $\Omega(n^{\frac{3}{2}})$, meaning that $T(n) \in \Theta(n^{\frac{3}{2}})$.