# Homework 04: Sorting

Roberto Corti

May 5, 2020

## Exercise 1

**By using the code at:**

`https://github.com/albertocasagrande/AD_sorting`

**implement INSERTION SORT, QUICK SORT, BUBBLE SORT, SELECTION SORT, and HEAP SORT.**

These functions are implemented inside this folder following their relatives pseudo-codes. They are respectevely written into the files `src/insertion_sort.c`, `src/quick_sort.c`, `src/bubble_sort.c`, `src/selection_sort.c`, `src/heap_sort`. All of these are tested into the `testing_sort` executable.

## Exercise 2

**For each of the implemented algorithm, draw a curve to represent the relation between the input size and the execution-time.**

## Exercise 3

**Argue about the following statement and answer the questions:**

(a) **HEAP SORT on a array $A$ whose length is $n$ takes time $O(n)$.** Since HEAP SORT complexity is given by the complexity of one single call of BUILD HEAP (which is $\Theta(n)$) and $n$ calls of EXTRACT MIN (which is $O(\log i)$, where $i$ is relative numbers of nodes in the heap) the overall cost of HEAP SORT is $O(n \log n)$. This higher bound represent an greater one respect to $O(n)$; thus in general HEAP SORT on a array $A$ of length $n$ doesn't take time $O(n)$.
However, in a specific case in which the heap is built in such a way that EXTRACT MIN costs $\Theta(1)$, then the overall cost is $\Theta(n)$ and the statement holds.

(b) **HEAP SORT on a array $A$ whose length is $n$ takes time** $\Omega(n)$. As we explained in the point (a), we know that HEAP SORT in the best-scenario case has an overall cost that is $\Theta(n)$. Then, since $T_{HS}(n) = \Theta(n)$ this is equivalent to say that $T_{HS}(n) = \Omega(n)$ and $O(n)$. Thus, since this lower bound holds for the best scenario will also hold for every possible case and we conclude that the statement is true.

(c) **What is the worst case complexity for HEAP SORT?** The worst-case scenario in HEAP SORT is that one in which we do $n$ calls of EXTRACT MIN operation calls and for every iteration $i$ the cost of this function is $\Theta(\log i)$, where $i$ is the relatives number of nodes down the heap. Thus,

$$T_{HS}(n) = \Theta(n) + \sum_{i=1}^{n} \Theta(\log i) = \Theta(n) + \Theta(n \log n) = \Theta(n \log n).$$

(d) **QUICK SORT on a array $A$ whose length is $n$ takes time.** $O(n^3)$. On an array $A$ of length $n$ QUICK SORT has on average a time performance $T_{QS} = \Theta(n \log_2 n)$. This will mean that on average $T_{QS}$ is both $O(n \log_2 n)$ and $\Omega(n \log_2 n)$. Since $O(n \log_2 n) \subset O(n^3)$ we can say that on average QUICK SORT takes time $O(n^3)$. However, using such higher bound cannot be useful during performance analysis since we have found, in this case $n \log_2 n$, a cheaper function that can be a bound to our complexity function. The same reasoning can be applied to the worst-case scenario which has complexity equal to $\Theta(n^2)$.

(e) **What is the complexity of QUICK SORT?** As we already stated in the previous point the complexity of QUICK SORT is on average and in the optimal case $\Theta(n \log n)$, while for the worst-case is $\Theta(n^2)$.

(f) **BUBBLE SORT on a array $A$ whose length is $n$ takes time** $\Omega(n)$. Since BUBBLE SORT involves two loops over the length $n$ of the array $A$, for any scenario the complexity of BUBBLE SORT is $\Theta(n^2)$. Automatically, BUBBLE SORT takes time $\Omega(n^2)$ and because $\Omega(n^2) \subset \Omega(n)$ we can conclude that the statement is formally true. However, we know that there exists a lower bound, in this case $n^2$, higher than $n$ and so this sentence is not useful in a performance analysis.

(g) **What is the complexity of BUBBLE SORT?**
As we say before, BUBBLE SORT involves two loops over the length $n$ of the array $A$, for any scenario the complexity of BUBBLE SORT is $\Theta(n^2)$. Automatically, BUBBLE SORT has a complexity $\Omega(n^2)$.

## Exercise 4

**Solve the following recursive equation:**

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

Let's use the above the relation for substituting the recursive term till join the case in which $n = 32$:

$$\begin{aligned}
T(n) &= 3T\left(\frac{n}{4}\right) + cn^{3/2}, \\
&= 3\left[3T\left(\frac{n}{4^2}\right) + c\left(\frac{n}{4}\right)^{3/2}\right] + cn^{3/2}, \\
&= 3^2T\left(\frac{n}{4^2}\right) + 3c\left(\frac{n}{4}\right)^{3/2} + cn^{3/2}, \\
&= \ldots \\
&= 3^kT\left(\frac{n}{4^k}\right) + cn^{3/2}\sum_{i=0}^{k-1}\left(\frac{3}{8}\right)^i, \\
&= 3^kT\left(\frac{n}{4^k}\right) + cn^{3/2}\left[\frac{1-(3/8)^k}{5/8}\right].
\end{aligned}$$

Looking to the base case, we want that $k^*$ that satisfies $\frac{n}{4^{k^*}} = 32 \Leftrightarrow k^* = \log_4 \frac{n}{32}$. Then,

$$\begin{aligned}
T(n) &= 3^{\log_4(n/32)} \cdot c' + cn^{3/2}\left[\frac{1-(3/8)^{\log_4(n/32)}}{5/8}\right], \\
&= 3^{\log_3(n/32)/\log_3 4} + cn^{3/2}\left[\frac{1-(3/8)^{\log_{3/8}(n/32)/\log_{3/8} 4}}{5/8}\right], \\
&= \frac{n}{32}^{1/\log_3 4} + cn^{3/2}\left[\frac{1-(n/32)^{1/\log_{3/8} 4}}{5/8}\right]
\end{aligned}$$

Thus, $T(n) \in O(n^{3/2})$.