Homework 04: Sorting

Roberto Corti

May 8, 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 number of nodes in the heap) the overall cost of HEAP SORT is $O(n \log n)$. This higher bound represent a 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-case scenario has an overall cost of $\Theta(n)$. Then, since $T_{HS}(n) = \Theta(n)$ this is equivalent to say that $T_{HS}(n) = \Omega(n)$ and $O(n)$. Thus, if 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 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 it 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 said before, BUBBLE SORT involves two loops over the length n of the array A, then for any scenario the complexity of BUBBLE SORT is $\Theta(n^2)$.

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 in order to build a recursion tree. each node represents the cost of a single sub-problem somewhere in the set of recursive function invocations. We sum the costs within each level of the tree to obtain a set of per-level costs, and then we sum all the per-level costs to determine the total cost of all levels of the recursion. For convenience, we assume that n is an exact power of 4, so that all sub-problem sizes are integers.

DISEGNO

Given the tree at Figure ..., I add up the costs over all levels to determine the cost for the entire tree:

$$\begin{aligned} T(n) &= cn^{3/2} + \frac{3}{8}cn^{3/2} + ... + \left(\frac{3}{8}\right)^{\log_4(n/32)-1} + \Theta(n^{\log_4 3}), \\ &= cn^{3/2} \sum_{i=0}^{\log_4(n/32)-1} \left(\frac{3}{8}\right)^i + \Theta(n^{\log_4 3}), \\ &\leq cn^{3/2} \sum_{i=0}^{+\infty} \left(\frac{3}{8}\right)^i + \Theta(n^{\log_4 3}), \\ &= \frac{8}{5}cn^{3/2} + \Theta(n^{\log_4 3}) \quad \in O(n^{3/2}). \end{aligned}$$

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