

Sorting (1): Homework Solutions

Emanuele Ballarin

July 14, 2020

Exercises 1, 2

Text:

By using the code at:

https://github.com/albertocasagrande/AD_sorting

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

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

Solution:

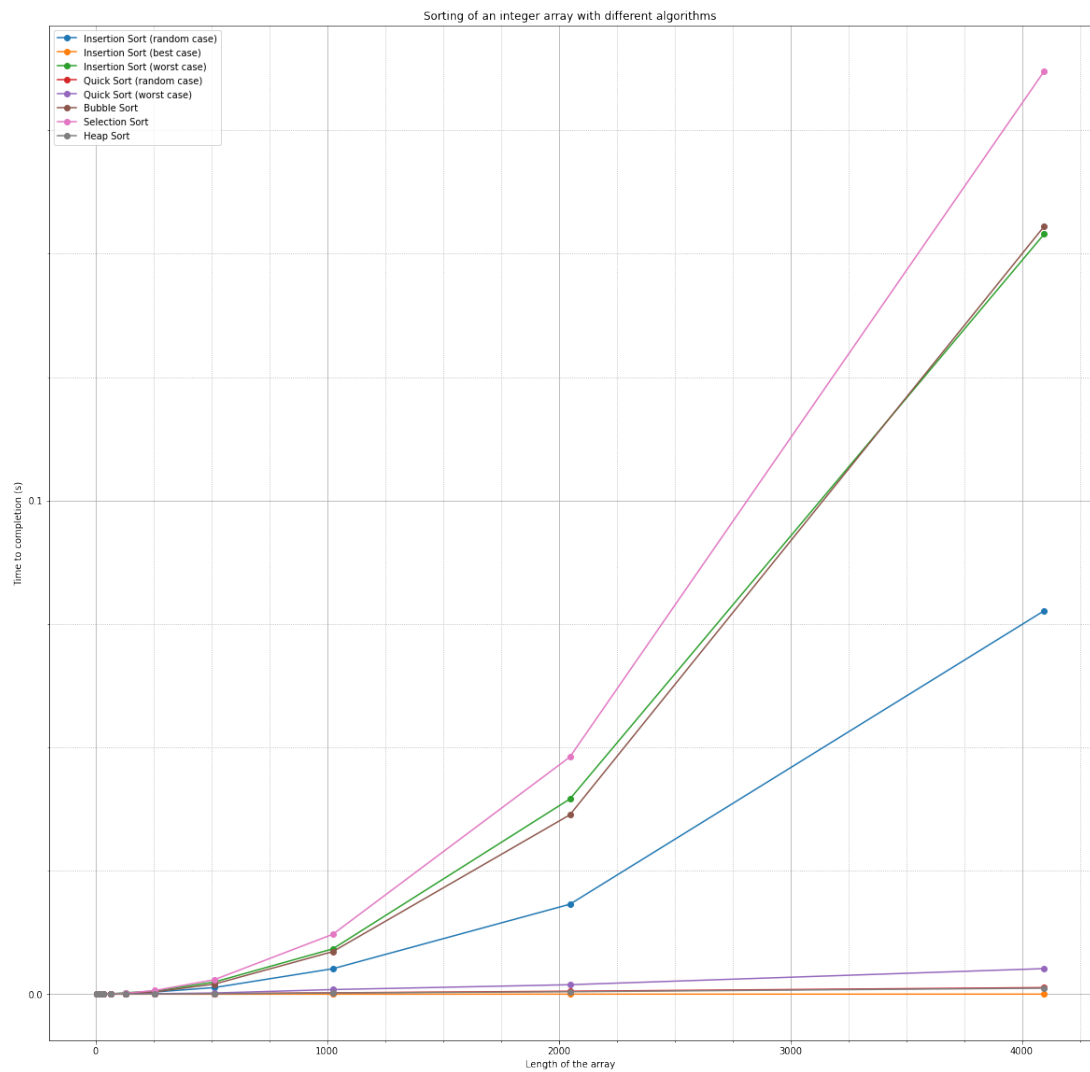
All the requested sorting algorithms have been implemented by faithfully following their description contained in the lecture materials. Most notably, original *Hoare's* partitioning has been used in QUICK SORT instead of other popular variants (namely Lomuto's and Sedgewick's).

Additionally, the *random pivoting* heuristic has been also tested for pivot selection in QUICK SORT, as a way to improve execution times in the non-asymptotic regime.

Benchmarks:

As it is possible to see from the graph, the behavior of the different sorting algorithms is in line with the theoretical predictions obtained from asymptotic complexity analysis, with the only notable exception of SELECTION SORT being the slowest. Though formally $O(n^2)$ in its best, average and worst cases, in fact SELECTION SORT involves less swaps w.r.t. other typical $O(n^2)$ approaches like BUBBLE SORT. This effect may be due to a still non-asymptotic regime for BUBBLE SORT and SELECTION SORT, or due to some uneven application of (automatic) compiler optimizations between the two.

Another interesting point is noticing the almost-equivalent behavior of **HEAP SORT** and **QUICK SORT** (*random case*). The former, however, offers tighter worst-case bounds.



Exercise 3

Text:

Argue about the following statements and answer the questions:

- (a) **HEAP SORT on a array A whose length is n takes time $O(n)$.**

In the general case, the complexity of **HEAP SORT** for an n -element array is determined by the contribution of the **build_heap** function operating once on the unsorted array ($O(n)$) and the repeated calls (n) to **extract_min** ($O(\log(n))$ each), amounting to a total of $O(n) + O(n \log(n)) \sim O(n \log(n))$. This holds true for both the (general) *best* and *worst* case scenarios. Thus, the statement is not true in general.

If, however, the statement must be true, it can be the consequence of the (specific) case in which the array is composed of all equal elements, thus reducing the complexity of **extract_min** to $\Theta(1)$ per call. The overall resulting time-complexity in that case amounts to $\Theta(n)$.

- (b) **HEAP SORT on an array A whose length is n takes time $\Omega(n)$.**

As better explained in previous answer, $\Theta(n)$ represents the complexity in the specific best case for the **HEAP SORT** algorithm, making the statement true.

- (c) **What is the worst case complexity for **HEAP SORT**?**

The worst-case time-complexity for **HEAP SORT** is $O(n \log(n))$. The bound is saturated in the case the maximum number of swaps is performed at each call to the **extract_min** function. This generally tight upper bound (compared to the more common $O(n^2)$) makes **HEAP SORT** a worst-case-robust sorting algorithm for many time-bound scenarios.

- (d) **QUICK SORT on a array A whose length is n takes time $O(n^3)$.**

The worst-case time-complexity of **QUICK SORT** is $O(n^2)$. The statement is true, though it provides a very large upper bound. Practical usefulness of this bound is debatable.

- (e) **What is the complexity of **QUICK SORT**?**

The best-case time-complexity of **QUICK SORT** (balanced partitioning at every step) is $O(n \log(n))$. This is also the average-case time-complexity, assuming that the distribution of the elements in the array accepts with equal probability each permutation. The worst-case time-complexity (maximally-imbalanced partitioning: e.g. already-sorted array and leftmost element as pivot) is instead $O(n^2)$.

(f) **BUBBLE SORT on a array A whose length is n takes time $\Omega(n)$.**

The statement is true. Indeed, such complexity is reached in the case of a pre-sorted array. The algorithm just needs to linearly-scan the array, verifying that it is sorted.

(g) **What is the complexity of BUBBLE SORT?**

As explained in previous answer, the best-case complexity of **BUBBLE SORT** is $\Theta(n)$. Average and worst case complexities equal to $O(n^2)$ due to the presence of nested looping over the entire array.

Exercise 4

Solve the following recursive equation:

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

Starting from the recursive equation proposed, we can build a recursion tree to ease its resolution. In this case, $cn^{3/2}$ for a given $c > 0$ fixed, has been chosen as a representative of the complexity class $\Theta(n^{3/2})$.

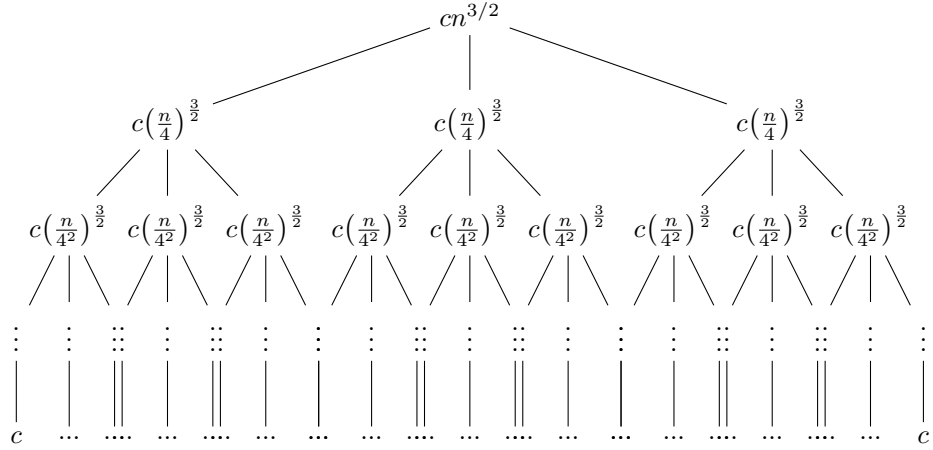


Figure 1: Recursion tree for the recursive equation above

As we can see (or read from the equation), by calling i the level of a node (starting from the root at $i = 0$), the complexity of each node in the tree is determined by $c(\frac{n}{4^i})^{3/2}$, whereas the number of nodes per level is 3^i .

The two expressions multiplied give the total cost per level, which is $c(\frac{3}{8})^i n^{3/2}$.

The last level of recursion is indexed by \hat{i} , such that $(\frac{n}{4^{\hat{i}}}) = 32 \rightarrow \hat{i} = \frac{\log_2(n)-5}{2}$.

The total cost for such last level of the tree is obtained by substituting the \hat{i} just found in the equations shown above, giving $\Theta(3^{-\frac{5}{2}} n^{\frac{\log_2(3)}{2}}) \sim \Theta(n^{\frac{\log_2(3)}{2}})$.

To conclude, we compute:

$$\sum_{i=0}^{\frac{\log_2(3)-5}{2}} (3/8)^i c n^{3/2} + \Theta(n^{\frac{\log_2(3)}{2}}) < \sum_{i=0}^{+\infty} (3/8)^i c n^{3/2} + \Theta(n^{\frac{\log_2(3)}{2}}) = (8/5) c n^{3/2} + \Theta(n^{\frac{\log_2(3)}{2}}) \sim O(n^{3/2}).$$