Sorting (2): Homework Solutions

Emanuele Ballarin

July 14, 2020

Exercise 1

Text:

Generalize the SELECT algorithm to deal also with repeated values and prove that it still belongs to $O(n)$.

Solution:
In order to generalize the `SELECT` algorithm to be able to deal with repeated values, its *vanilla* version has been modified by performing a partition of the input array into three parts: that containing all elements smaller than, equal to and greater than the *pivot*. Such algorithm still belongs to $O(n)$ as it is operatively different from the *standard* `SELECT` just in having to perform (eventually) one comparison more with the *pivot* per element, which is in no way $> O(n)$.

Exercise 2

Text:

- *Implement the `SELECT` algorithm of Ex.1.*
- *Implement a variant of the `QUICK SORT` algorithm using above mentioned `SELECT` to identify the best pivot for partitioning.*
- *Draw a curve to represent the relation between the input size and the execution-time of the two variants of `QUICK SORT` (i.e, those of Ex. 2 and Ex. 1 31/3/2020) and discuss about their complexities.*

Solution:

As it is possible to notice from the graph below, the `QUICK SORT + SELECT` algorithm is noticeably slower than its simpler counterpart for small-sized arrays, and becomes more and more efficient (actually overcoming the simpler `QUICK`

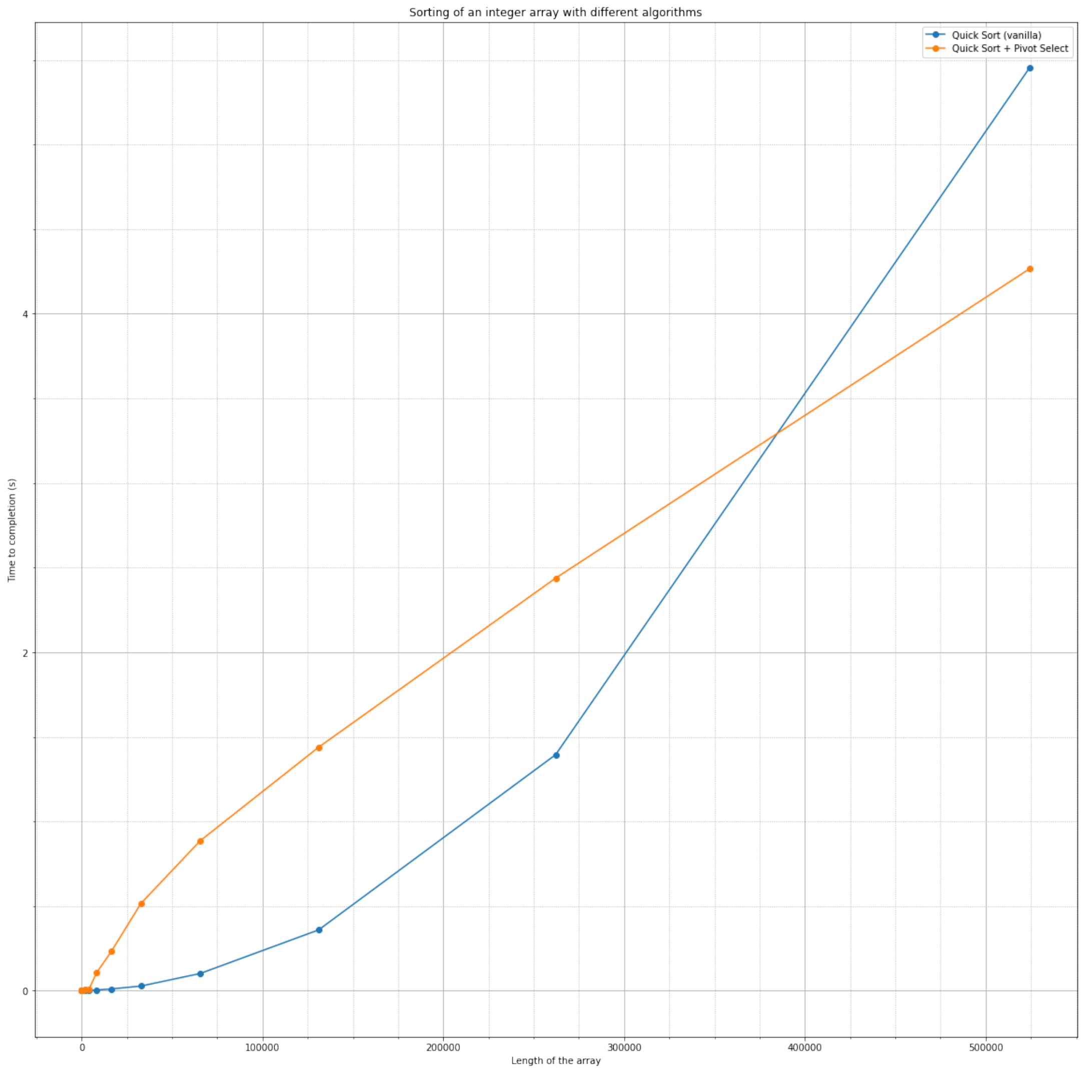
`SORT`) as the array size grows.

Additionally, as it is possible to notice, the curve representing the simple `QUICK SORT` seems to be convex, whereas the other seems to be concave right from the beginning.

Indeed, this is explainable by considering that the complexity of `QUICK SORT + SELECT` – i.e. a version of `QUICK SORT` that uses `SELECT` and the *median of medians* algorithm to determine a pivot leading to a balanced partition – is $O(n\ log(n))$ asymptotically, but heavily affected by overhead for smaller arrays.

On the other hand, while still being $O(n\ log(n))$ on average, simpler `QUICK SORT` makes it difficult to choose a pivot leading to even acceptable partition balance as the arrays grows in size, eventually skewing complexity toward the worst case $O(n^2)$.

Benchmarks:


Sorting of an integer array with different algorithms
Quick Sort (vanilla)
Quick Sort + Pivot Select
Time to completion (s)
Length of the array
0
2
4
0
100000
200000
300000
400000
500000

Exercise 3

Text:

In the algorithm **SELECT**, *the input elements are divided into chunks of* 5*. Will the algorithm work in linear time if they are divided into chunks of* 7*? What about chunks of* 3*?*

Solution:
If we choose to partition the n input elements in chunks of 7, we will have $\lceil n/7 \rceil$ chunks in total and thus – by exploiting an already known result – we will have $4 \cdot (\lceil (1/2)\ \lceil n/7 \rceil \rceil - 2)\ \geq\ (2n/7) - 8$ elements larger than the *median of medians.*

The complexity of the algorithm becomes the solution of the following recursive equation:

$$T(n) = T(\lceil n/7 \rceil) + T((5n/7) + 8) + O(n).$$

We can solve such equation by substitution, *guessing* that $T(n) \leq cn;\ c > 0$ and choosing $c'n;\ c' > 0$ as a generic representative for the class $O(n)$. It is also safe to assume that $n > 7$.

We obtain, as a consequence:

$$T(n) = T(\lceil n/7 \rceil) + T((5n/7)+8) + c'n \leq c(n/7+1) + c((5n/7)+8) + c'n = c(6n/7) + 9c + c'n.$$

By regrouping, we can show that $T(n) = cn + \Delta \leq cn \iff \Delta = -(cn/7) + 9c + c'n \leq 0$.

This is equivalent to the condition $c \geq (7nc'/(n-63))$, which can then be satisfied by imposing e.g. $n \geq 126;\ c \geq 14c'$, thus proving that $T(n) \in O(n)$.

In an analogous fashion, if we choose chunks of 3 elements, we obtain that the number of chunks is $\lceil n/3 \rceil$ and that of the elements greater than the *median of medians* is $2 \cdot (\lceil (1/2)\ \lceil n/3 \rceil \rceil - 2)\ \geq\ (2n/6) - 4$.
The complexity of the algorithm becomes the solution of the following recursive equation:

$$T(n) = T(\lceil n/3 \rceil) + T((4n/6) + 4) + O(n).$$

This time – solving again by substitution – we guess that $T(n) > cn;\ c > 0$ and we choose $c'n;\ c' > 0$ as a generic representative for the class $O(n)$.

We obtain

$T(n) = T(\lceil n/3 \rceil) + T((4n/6)+4) + c'n \geq c(n/3) + c((4n/6)+4) + c'n = cn + 4c + c'n > cn.$

that cannot be bounded linearly in any case.

Exercise 4

Text:
Suppose that you have a "black-box" worst-case linear-time subroutine to get the position in A of the value that would be in position n/2 if A was sorted. Give a simple, linear-time algorithm that solves the selection problem for an arbitrary position i.

Solution:
In the case such arbitrary position i is actually $i = n/2$, just a call to the *black-box routine* solves the problem in linear time. Otherwise, it is possible to call such routine nonetheless on the array A and subsequently partition such array, according to the output of the routine itself, in two sub-arrays of values greater or smaller than such output.
At this point, with a comparison it is possible to determine in which sub-array the element looked-for should be, and to call on such sub-array the *black-box routine*. Recursively applying this procedure leads to termination in asymptotic linear time.

More formally, the algorithm can be summarized as follows:

```
LINEAR_SELECT(A,i):
  m ← BLACK_BOX(A)
  if i=n/2 return m
  (A_smaller, A_larger) ← PARTITION(A, m)
  if i<n/2 return LINEAR_SELECT(A_smaller,i)
  return LINEAR_SELECT(A_larger,i-(n/2))
```

The complexity of such algorithm is the solution to the recursive equation:

$$T(n) = T(n/2) + O(n) \in O(n)$$

since both the *black-box routine* and a partitioning have complexity $O(n)$.

Exercise 5

Text:
Solve the following recursive equations by using both the recursion tree and the substitution method:

1. $T_1(n) = 2 \cdot T_1(n/2) + O(n)$
2. $T_2(n) = 2 \cdot T_2(\lceil n/2 \rceil) + T_2(\lfloor n/2 \rfloor) + \Theta(1)$
3. $T_3(n) = 3 \cdot T_3(n/2) + O(n)$
4. $T_4(n) = 7 \cdot T_4(n/2) + \Theta(n^2)$

Solutions:

1. *Recursion Tree*:

The recursion tree generated by the given equation has – at any level – twice the elements of the previous level. For such reason, starting to index levels from the root ($i = 0$) downwards, generic level i presents 2^i nodes, of cost $O(n/(2^i))$ each.

By choosing cn as a representative for the complexity class $O(n)$, we obtain a total cost per level of cn, which is independent of i.

By summing over all levels (which are $log_2(n)$), we obtain that:

$$T_1(n) = cn \sum_{i=0}^{log_2(n)} 1 < cn\ log_2(n) \in O(n\ log_2(n)).$$

1. *Substitution*:

To start, we can *guess* that the solution belongs to the complexity class $O(n\ log_2(n))$ and we can choose $cn\ log_2(n)$ as a representative for such class.

Then we assume our hypothesis holds $\forall m < n$, meaning formally that $T_1(m) \leq cm\ log_2(m),\ \forall m < n$.

At this point, we can choose $c'n$ as a representative of the class $O(n)$, obtaining that:

$$T_1(n) = 2T_1(n/2) + c'n \leq 2c\ (n/2)\ log_2(n/2) + c'n \leq cn\ log_2(n) - cn + c'n.$$

To conclude, we just notice that $cn\ log_2(n) - cn + c'n \leq cn\ log_2(n) \iff c \geq c'$.

2. *Recursion Tree*:

As a starting point for the resolution, we can notice that – in such case – the branching stemming from each node is asymmetric. In fact, one branch (namely: the left) always depends on just *floor* operations, whereas the other (namely: the right) just on *ceiling* operations.

In general, we also notice that each level has twice the number of nodes in the previous one and that each node has constant complexity.

Recalling that $\forall n \in \mathbb{N}, \exists m \in \mathbb{N}$ *s.t.* $n \leq 2^m \leq 2n$ and by considering extreme branches only, we can show that:

- Leftmost branch has length $\leq log_2(2n)$ and can be used to bound overall complexity from above;
- Rightmost branch has length $\geq log_2(n/2)$ and can be used to bound overall complexity from below.

If we choose cn as a representative for complexity class $\Theta(1)$ and we recall that level i has 2^i nodes (starting from the root $i = 0$), we obtain that:

- $T_2(n) \geq \sum_{i=0}^{log_2(n/2)} (c\ 2^i) \geq c\ 2^{(log_2(n/2))+1} - 1 \geq cn - c\ \rightarrow\ T_2(n) \in \Omega(n)$;
- $T_2(n) \leq \sum_{i=0}^{log_2(2n)} (c\ 2^i) \leq c\ 2^{(log_2(2n))+1} - 1 = 4cn - c\ \rightarrow\ T_2(n) \in O(n)$.

As a general consequence, $T_2(n) \in \Theta(n)$.

2. *Substitution*:

We approach the task to prove that our *guess* – namely $T_2(n) \in \Theta(n)$ – is true by both showing that $T_2(n) \in \Omega(n)$ and $T_2(n) \in O(n)$ by substitution.

First, we *guess* that $T_2(n) \in O(n)$, we choose 1 as a representative of $\Theta(1)$ and $cn - d$ as a representative of $O(n)$. We are required to make such – seemingly unusual – latter choice in order to avoid getting stuck in the calculations with lower-order terms.

By assuming that $T_2(m) \leq cm - d,\ \forall m < n$, we have that:

$$T_2(n) \leq c(\lceil n/2 \rceil) + c(\lfloor n/2 \rfloor) - 2d + 1\ \leq\ cn - 2d + 1.$$

Lastly, we have that $cn - 2d + 1 < cn - d \iff d \geq 1$, concluding the first part of the proof.

Now, we *guess* that $T_2(n) \in \Omega(n)$, we choose cn as a representative of $\Omega(n)$ and 1 as a representative of $\Theta(1)$.

This way, we obtain that:

$$T_2(n) \geq c(\lceil n/2 \rceil) + c(\lfloor n/2 \rfloor) + 1\ \geq cn + 1 \geq cn,\ \forall c > 0.$$

completing the proof.

3. *Recursion Tree*:

In this case, each level has thrice the number of nodes as the previous one, making the number of nodes at the i-th level 3^i. The cost per node at level i is $c\ (n/2^i)$ (choosing c as a representative of $O(n)$) and thus making the overall cost of level i equal to $cn\ (3/2)^i$.
Since tree depth is $log_2(n)$, we obtain that:

$$T_3(n) \leq \sum_{i=0}^{log_2(n)} cn\ (3/2)^i = (1/2)\ cn\ ((3/2)^{log_2(n)} - 1) = cn\ (3n^{(log_2(3))-1} - 1) \in O(n^{(log_2(3))})$$

3. *Substitution*:

We start by *guessing* that $T_3(n) \in O(n^{(log_2(3))})$, by choosing $c'n$ as a representative of $O(n)$ and $cn^{log_2(3)} - dn$ as a representative of $O(n^{(log_2(3))})$ (for the same reasons as in the previous equation).

This allows us to show that:

$$T_3(n) \leq 3\ \left((cn^{log_2(3)})/3 - dn/2\right) + c'n = cn^{log_2(3)} - n((3d/2) - c').$$

To conclude, we notice that $cn^{log_2(3)} - n((3d/2) - c') \leq cn^{log_2(3)} - dn \iff d \leq 2c'$.

4. *Recursion Tree*:

In this last case, we have at level i of the tree 7 times the nodes of level $i-1$, thus giving at level i 7^i nodes of cost – having chosen cn^2 as a representative of $\Theta(n^2)$ – $c\ (n/(2^i))^2$ each. The total cost at level i is $7^i\ c\ (n/(2^i))^2$.

Consequently, being $log_2(n)$ the depth of the tree, we obtain:

$$T_4(n) \leq \sum_{i=0}^{log_2(n)} (7/4)^i\ cn^2 = (cn^2)\ (4/3)\ (7/4)^{log_2(n)+1} - 1 = (cn^2)\ (4/3)\ \left((7/4)\ n^{log_2(7) - log_2(4)} - 1\right)$$

$$= (cn^2)\ (4/3)\ \left((7/4)\ n^{log_2(7)-2} - 1\right) \sim O(n^{log_2(7)}).$$

4. *Substitution*:

We start by *guessing* that $T_4(n) \in O(n^{log_2(7)})$, and by choosing $c'n^2$ as a representative of $\Theta(n^2)$ and $cn^{log_2(7)} - dn^2$ as a representative of $O(n^{log_2(7)})$.

As the inductive hypothesis, we assume $T(m) \leq cm^{log_2(7)} - dm^2, \ \forall m < n$, obtaining that:

$$T_4(n) \leq c'n^2 + 7c \ ((n/2)^{log_2(7)} - (dn^2)/2) = cn^{log_2(7)} - n^2 \ ((7d)/2 - c').$$

And finally, we notice that $cn^{log_2(7)} - n^2 \ ((7d)/2 - c') \ \leq cn^{log_2(7)} - dn^2 \iff d \leq (2/5)c'$, concluding the proof.