# Homework 05: Sorting (Part 2)

Roberto Corti

June 3, 2020

## Exercise 1

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

## Exercise 2

**Download the latest version of the code from**

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

**and**

- **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.**

The new version of SELECT that deals with repeated values described in the previous section is implemented inside the file `select.c` and `quick_sort.c` for the tri-partition implementation.
Once applied this algorithm, I implemented a variant of QUICK SORT using the SELECT method in order to choose the best pivot:

```
Quick-Sort Select(A, left, right):
  while left ≤ right do
     pivot ← select_pivot(A, left, right)
     Quick-Sort Select(A, left, k.first)
     left ← k.second+1
  end while
```

The results, compared with the classic version of QUICK SORT, are represented in the following plot.

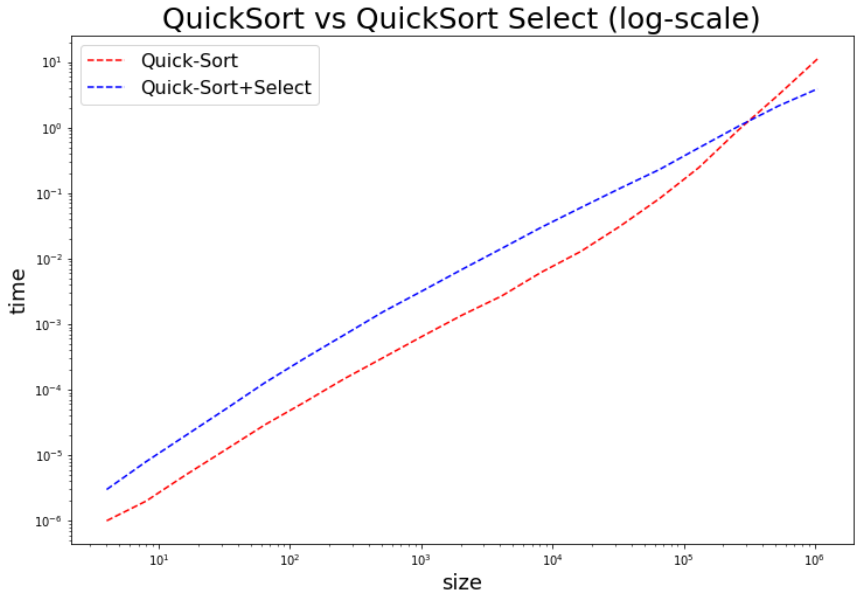


Figure 1: Benchmark of the `quick_sort` method compared to its version when the `select` algorithm is involved in the selection of the pivot.

As it can be observed, the classic QUICK SORT has better performances on small list sizes. However, for big arrays the cost of partitioning becomes relevant and in those cases QUICK-SORT SELECT results to be more efficient.