# SDP 2016/17 - Lab 3 - Davide Gallitelli S241521

## E01 - Report

The goal of this exercise is to sort an integer stored in a binary file. Two approaches are possible:

- reading the file and generating a local copy of the array
- mapping the file to memory and operate on it directly

The second approach proves to be more effective because the array might be very large. For this reason, the approach chosen is the second one.

The first step was to open the stream for the binary file, and map it to memory using the *mmap()* function.

```
paddr = mmap ((caddr_t) 0, len, PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);
```

In order to provide the lenght of the file to the *mmap()* function, the *fstat()* function was used, which returns to the *struct stat stat_buf* information about the file, including the file size. The *mmap()* return the address that points to the beginning of the binary file. This address will be casted to *int** so it can be considered as an integer array. This value is passed as an argument to the quicksort function.

The *quicksort()* function starts as provided by the exercise text. However, it behaves differently when the size of the array to be sorted is greater than the user-defined size.

When this condition is met, we call a threaded version of the quicksort algorithm: two threads will be instantiated to handle concurrently the workload, that's why two structures are needed, each defining the access range on the array by means of the left and right parameters.

The quicksort function is called recursively until the array is sorted. This means that each of the two initial thread will keep instantiating more threads until the access range is below the specified threshold. When this happens, the standard quicksort function will handle the sorting.