Lab 3.1

Write a C program using Pthreads to sort the content of a **binary** file including a sequence of integer numbers. Implement a threaded quicksort program where the recursive calls to **`quicksort`** are replaced by threads activations, i.e. sorting is done, in parallel, in different regions of the file.

If the difference between the **`right`** and **`left`** indexes is less than a value **`size`**, given as an argument of the command line, sorting is performed by the standard **`quicksort`** algorithm.

This is a sequential recursive implementation of the quicksort algorithm.

```
void quicksort (int v[], int left, int right) {
  int i, j, x, tmp;
  if (left >= right)   return;
  x = v[left];
  i = left - 1;
  j = right + 1;
  while (i < j) {
    while (v[--j] > x);
    while (v[++i] < x);
    if (i < j)
      swap (i,j);
  }
  quicksort (v, left, j);
  quicksort (v, j + 1, right);
}

void swap(int i, int j){
  int tmp;

  tmp = v[i];
  v[i] = v[j];
  v[j] = tmp;
}
```

Lab 3.2

Write a C program using Pthreads that implements the product of two matrices **`m1`** and **`m2`**, of dimension **`nr1 x nc1`** and **`nr2 x nc2,`** respectively, where **`nr1`** and **`nc1`** are the number of rows and columns of matrix **`m1`**, and **`nr2`** and **`nc2`** refer to matrix **`m2.`**
The dimensions of the matrices are given as arguments of the command line, and you must verify that **`nc1`** and **`nr2`** are equal.

The main thread allocate dynamically the two matrices, fills each matrix with increasing integer numbers, starting from 0.
Then, it creates **`nr1*nc2`** threads **`prod_th`**, and an additional thread **`p_th`** that is responsible for printing the product of the two matrices when all threads **`prod_th`** have completed their work.
The main threads does not wait the termination of the **`prod_th`** threads.

Each thread **`prod_th`** computes the product of the **`i`**-th row of matrix **`m1`** and the **`j`**-th column of matrix **`m2`**.