1. Quicksort works by picking up a pivot.
Push the elements lesser than the pivot to its left and elements greater than to its right.
Recursively do the same operation for the left part and the right part.

In parallel quicksort, we do a similar kind of operations parallely.

```
parallel Qsort
{   if ( element pointed by low < element pointed by high
    {   int pivot = partition ()
        parallel Qsort (array, low, pivot-1)
    }
    parallel Qsort (array, pivot+1, high)
```

use open mp tasks to execute both these recursive calls parallely.
Similarly other nested calls will also be divided into different tasks and will be assigned to processors.

continued ....

```
int partition ()
{
Create 2 arrays of size as input array.
In this case choose pivot as the last element.

for each element in the input array.
    if element > pivot.
        put it in the 2nd array.
    else
        put it in 1st array.

    for i = 0 to no of elements in left array.
        copy all elements into output array.

    for i = 0 to no of elements in right array
        copy all elements into output array.
    return the pivot position.
}
```

2) void histogram (size_t n, const double *x, int num_bins, double **bin_bdry, size_t **bin_count).

Step 1 - To determine the range of each bin. We need to calculate the max and min number in the array.

Use Openmp reduce to calculate max and min number in array.

Then range for each bin is $\frac{max - min}{num\_bins}$.

For each number in the array determine the bin number it has to fall under, & increment the bincount. This can be done using.

$$\left[\frac{number}{no\ of\ bins}\right]$$

gives the bin number in which the number falls.

Execute this using a parallel for loop. Since 1 or more threads might update the same bin_count, simultaneously. Use openmp lock mechanism to prevent the race condition.

Or.. Calculate local bin_counts for each thread And then calculate global_count by merging local counts.