```
Question #3 : BINARY SEARCH
---------------------------


OMP algo
---------------
ompBSearch(....) {
//data size: n
//# threads: t
//key, arr[], id (0 to t-1)
size=ceil(n/t)
seg_start=id*size
if (seg_start>=n) return // data size is less than number of threads available
seg_end=(id+1)*size-1
if (seg_end>=n || (id==t-1 && seg_end<n-1)) { // resolve boundary conditions
   seg_end=n-1
   size=seg_end-seg_start+1
}
if (key<seg_start || key>seg_end) return NULL // key is not in this block
//search now
seg_center=seg_start+size/2
if (seg_start==seg_end && seg_center != key) return NULL // key not found
if (seg_start <= key < seg_center)
  ompBsearch(seg_start, size/2, key)
else if (seg_center < key <= seg_finish)
  ompBsearch(seg_center+1, size-size/2-1, key)
else
  return seg_center // Key found
}

MPI Algorithm
-----------------
mpiBsearch(....) {
// num procs = p
split array of size n in p chunks of n/p
MPI_Scatter each chunk to the procs in the group
Each processor runs ompBSearch(....) (shown above) and reports
     - either NULL for key not found
     - or the element that matched (only one because the array has no duplicates)
Report result
}

The implementation is not complete and no performance numbers are available to report. The working code so far
developed is included. It is the OMP part. The MPI code is not included because it is work in progress.
```

```
The command accepts:
-n for array size
-k for key value (optional if not mentioned then a random key is used)

The program generates a sorted array of random unsigned integers. The search is applied on this array.
```