Problem 5: [15 marks]

Design an iterative version of the quick-sort algorithm using queues. (a) Give a high-level description of your algorithm and explain why it works in a few sentences. (b) Write pseudo code for your iterative algorithm (assume that a queue implementation is already available). (c) In your pseudo-code, write down the invariants corresponding to the pivot element.

The algorithm consists of two ports : Partitioning and qsort.

(a) ⟹ Partitioning : We randomly select a pivot, ~~take~~ swap it with last element of array. Make two pointers i=0 j=n-2
while i<j, we check if arr[i] < pivot and arr[j] ≥ pivot.

- If not we swap arr[i] & arr[j]. At the end, we swap arr[i] and pivot and return(i) [pivot index].
- This ensures all elements before pivot are smaller and after pivot are bigger.

⟹ qSort :

- we have two pointers l and u.
  Initially l=0, u=n-1
- we also have pi (int variable).
- pi = partition(l, u, arr)
- Now we enqueue in the order : l, pi-1, pi+1, u.
- while queue is not empty we dequeue first two elements and assign l = dequeue(), u = dequeue() and run the loop again

Thus we keep on doing the same thing for smaller and smaller sub arrays.

(b)
```
Partition[l,u,arr]
   p = random(l,u)
   swap[arr[p], arr[u]]
do(while (i<j))
      if arr[i] < arr[u]) i++
      if arr(j) ≥ arr(u) j--
      swap(arr[i], arr[j])
swap( arr[i], arr[u])
return i
```

(b) qsort
```
queue(int) Q.
l=0
u=n-1
pi=partition(l,u,arr]
Q.enq(l), Q.enq(pi-1), Q.enq(pi+1), Q.enq(u)
while (Q != empty)
   l = Q.deq()                if l==u {pass}
   u = Q.deq()
else: pi = partition[l,u,arr]
      Q.enq(l), Q.enq(pi-1), Q.enq(pi+1), Q.enq(u)
End while.
```

(c) The pivot element, after partitioning is always at the final and correct place required after sorting.

After partitioning, elements left to pivot are smaller than it, and those to right are greater than it.