The Quicksort algorithm we have seen in class is the following.

```plaintext
QUICKSORT(A, p, r)

1  if p < r:
2      q ← PARTITION(A, p, r)
3      QUICKSORT(A, p, q - 1)
4      QUICKSORT(A, q + 1, r)

PARTITION(A, p, r)

1  i ← p - 1
2  for j ← p to r - 1
4        i ← i + 1
7  return i + 1
```

(1) Make the smallest possible change to the algorithm such that it would stop with a nearly-sorted array. This is accomplished by not sorting subarrays with $k$ elements or less. Mark your change on the copy of the algorithm given above.

(2) With the change you made in part 1, and $k = 3$, how many recursive calls to Quicksort are required to nearly-sort the following array?

```
3  6  1  4  7  2  8  5
```

(3) With the change you made in part 1, $k = 3$, and the array given in part 2, compute the final array obtained by Quicksort.
Answer:

Change line 1 to: if $r - p \geq k$

In the call with indices (1,8), PARTITION results in $q = 5$ and the array:

```
3 1 4 2 5 6 8 7
```

This creates two recursive calls, with indices (1,4) and (6,8).
The call with indices (6,8) returns immediately.
In the call with indices (1,4), PARTITION results in $q = 2$ and the array:

```
1 2 4 3 5 6 8 7
```

This creates two recursive calls, with indices (1,1) and (3,4).
Both calls return immediately.
A total of five calls and the final array is the last one shown above.
The Quicksort algorithm we have seen in class is the following.

```plaintext
QUICKSORT(A, p, r)
1  if p < r:
2      q ← PARTITION(A, p, r)
3      QUICKSORT(A, p, q − 1)
4      QUICKSORT(A, q + 1, r)

PARTITION(A, p, r)
1  i ← p − 1
2  for j ← p to r − 1
4          i ← i + 1
7  return i + 1
```

1. Make the smallest possible change to the algorithm such that it would stop with a nearly-sorted array. This is accomplished by not sorting subarrays with $m$ elements or less. Mark your change on the copy of the algorithm given above.
2. With the change you made in part 1, and $m = 3$, how many recursive calls to Quicksort are required to nearly-sort the following array?

   4 7 2 5 8 3 9 6

3. With the change you made in part 1, $m = 3$, and the array given in part 2, compute the final array obtained by Quicksort.