MergeSort: make the split nearly equal (off by one) defun almost equal in LISP

a. Initial Part (L1):
    i. The first two elements of the list should be processed in the initial part (L1).
    ii. The first element ((car unprocessed)) is placed in l1, and the second element ((car (cdr unprocessed))) is placed in l2.
b. Remaining Part (L2):
    i. After removing the first two elements, the remaining part of the list is passed to the recursive call, i.e., (cdr (cdr unprocessed)).
    ii. The remaining part (L2) will be processed recursively by applying Process A again on this smaller list.
c. Recursive Combination:
    i. After processing the initial part (L1) and the remaining part (L2), the results are combined:
        1. A(L1) is the result of processing the first two elements of the list (i.e., the first item to l1, the second item to l2).
        2. A(L2) is the result of recursively applying Process A to the remaining elements of the list.
        3. These two parts are combined (concatenated) to form the final result.

Question 2 Answers:

1. Define a recursive function to do the partitioning. The basic partition strategy is to maintain the processed items as two lists, and at each stage take the next two unprocessed items and put one in each list (what if there are zero or one item in the unprocessed list?).
 ((null unprocessed) (list l1 l2))
 ((null (cdr unprocessed)) (list (cons (car unprocessed) l1) l2)) ;

2. The result of partitioning is two halves, As per Mergesort, what should we do to each Half?

a. Recursively apply Mergesort to $L1$
b. Recursively apply Mergesort to $L2$

3. Define a merge function that merges two sorted lists. Now the processed part is one sorted list, and the unprocessed part is two sorted lists. We need to pick the first item from one of the two unprocessed lists and append it to the processed part. What is the end Conditions?

a. Base Case 1: One of the lists becomes empty. In this case, append the remaining elements of the other list to the result.
b. Base Case 2: Both lists are empty (which will only happen at the end of the process). This signals that the entire merge process is complete.

Question 4
Implement Insertion sort in Lisp.

a. While insertion sort is in progress, we track 2 lists: the sorted items and the unsorted items. What should these look like when the process starts and when the process ends? Can we use that information to decide on termination?

Example input list: (4 3 1 5 2)

Starting Point
—-----------------------
Sorted list: ()
Unsorted list: (4 3 1 5 2)
Ending Point
—-------------------------
Sorted list: (1 2 3 4 5)
Unsorted list: ()

We would decide that the termination condition is that it would end when the unsorted list is empty

b. In each pass, we start with two lists. At the end of the pass, we would have moved one more item from the sorted to the unsorted list. When is this process trivially accomplished? What will the recursive call look like?

Sorted List (L1): a portion of the list already sorted
Unsorted List (L2): portion of list not sorted yet.

Process A (insertion sort) has been applied to list L if the following holds:

Process A has been applied to the initial part of the list (L1), recursively yielding A(L1) (the sorted part).

We can apply Process A recursively to the remaining part of the list (L2), recursively yielding A(L2) (the sorted remaining portion).

A(L1) and A(L2) can be combined to yield the final sorted list A(L).

At each step, we remove the first element from unsorted list and insert it into sorted list. Then we recursively call it until unsorted list is empty.

The call would look like this: ( insertion-sort ) (sorted-list (remaining-unsorted-list ) )

c. Moving requires an operation that can insert an item into a sorted list in ascending order. This can be done easily using the predefined LAST function in Common Lisp (look for this in the LISP text; LAST and BUTLAST are like the reverse of CAR and CDR). Again, to represent this insertion process, we need to track the items that have been examined, the items to be inserted, and the items yet to be examined. There are two ways in which the process can terminate – what are they?

We must
Track the items examined (examined)
The item to insert (currInsert)
Items yet to be examined (unexamined)
The two ways in which the process can terminate are
We find the correct position and insert it

We reach end of sorted list, then add item to end.

d. Implement the insertion sort using Lisp