# Challenge 3:
# Odd-even Transposition Sort

This sorting algorithm, developed originally for use on parallel processors, compares all odd-indexed list elements with their immediate successors in the list and, if a pair is in the wrong order (the first is larger than the second) swaps the elements. The next step repeats this for even-indexed list elements (and their successors). The algorithms iterates between these two steps until the list is sorted.

**Single Processor Solution**

The single-processor algorithm is simple, but not very efficient ($O(n^2)$). It can be considered a variation of the bubble sort algorithm. Here a zero-based index is assumed:

```
function oddEvenSort(list) {
  function swap(list, i, j) {
    var temp = list[i];
    list[i] = list[j];
    list[j] = temp;
  }

  var sorted = false;
  while(!sorted) {
    sorted = true;
    for(var i = 1; i < list.length-1; i += 2) {
      if(list[i] > list[i+1]) {
        swap(list, i, i+1);
        sorted = false;
      }
    }

    for(var i = 0; i < list.length-1; i += 2) {
      if(list[i] > list[i+1]) {
        swap(list, i, i+1);
        sorted = false;
      }
    }
  }
}
```

**Multi Processor Solution**

On parallel processors, with one value per processor and only local left–right neighbour connections, the processors all concurrently do a compare–exchange operation with their neighbours, alternating between odd–even and even–odd pairings in each step. The algorithm has linear runtime as comparisons can be performed in parallel.

A pseudocode implementation that uses message passing for synchronisation is presented in the following. The driver code spawns n processes, one for each array element and collects the results after termination.

```
process ODD-EVEN-PAR(n, id, myvalue)
  // n … the length of the array to sort
  // id … processors label (0 .. n-1)
  // myvalue … the value in this process
begin
  for i := 0 to n-1 do
  begin
    // alternate between left and right partner
    if i+id is even then
      if id has a right neighbour
        sendToRight(myvalue);
        othervalue = receiveFromRight();
        myvalue = min(myvalue, othervalue);
    else
      if id has a left neighbour
        sendToLeft(myvalue);
        othervalue = receiveFromLeft();
        myvalue = max(myvalue, othervalue);
  end for
end ODD-EVEN-PAR

for i := 0 to array.length-1
  process[i] := new ODD-EVEN-PAR(n, i, array[i])
end for

start processes and wait for them to finish

for i := 0 to array.length-1
  array[i] := process[i].myvalue
end for
```

**Verification Tasks:**

1. Specify and verify that the result of the even-odd sort algorithm is a sorted list.
2. Specify and verify that the result of the even-odd sort algorithm is a permutation of the input list.
3. Prove that the code terminates.

*Concurrency:* This algorithm was developed originally for parallel use. You should aim to have a parallel solution also if your tool allows.

*Synchronisation*: We have proposed a synchronisation scheme using messages between neighbouring processes. You are free to use a different scheme (semaphores, locks, …) if you wish.

*Caution*: The implementations shown above are for demonstration purposes only, they have not been thoroughly tested, let alone formally verified. That's your job!