Problem 1.

1. A recursive procedure is characterized by a build-up of a chain of deferred (pending) operations, employing recursive-style programming, where the function calls itself, while the state of the overall computation (process) is hidden away and maintained internally by the interpreter, not explicitly contained in the program variables. Typically, the longer the chain, the more information must be maintained by the interpreter (the interpreter keeps track of the operations to be performed later on), which is resource-wise heavy on both space and time. However, recursive procedures usually have more straightforward code, are easier on the eye and more understandable.
2. In contrast, an iterative procedure (though also employing recursive-style programming in Scheme), is one where the current state of computation is explicitly stored in program variables, having a fixed rule that describes how the state variables should be updated in the chain of computation and possibly a base case that describes when the procedure should terminate. In this type of procedures, you don't defer operations for later, but instead pass the intermediary results to the next step, which aids in space complexity, although this doesn't necessarily say anything about time complexity. Iterative procedures are usually harder to read than recursive procedures, because of all the program variables needed to be tracked.

Problem 2A.

```
(define (even-odd sequence)
   (if (null? sequence) 0
       ((if (even? (car sequence)) + -)
        (even-odd (cdr sequence))
        (car sequence))))
```

Problem 2B.

```
(define (even-odd-iter sequence)
   (define (iter sum seq)
     (if (null? seq) sum
         (iter ((if (even? (car seq)) + -) sum (car seq)) (cdr seq))))
  (iter 0 sequence))
```

Problem 3.

```
(define (swap i j lst)
  (define (iter ind seq)
        (if (null? seq) nil
         (cons (cond ((= ind i) (list-ref lst j))
                     ((= ind j) (list-ref lst i))
                     (else (car seq)))
               (iter (+ 1 ind) (cdr seq)))))
  (iter 0 lst))
```