Problem 1

a) Search Algorithm:

```
function SEARCH_DOUBLY_LINKED_LIST(L, x)
    e ← L.head
    while e.key ≠ x and e ≠ NIL do
        e ← e.next
    end while
    return e
end function
```

Insert Algorithm:

```
function INSERT_DOUBLY_LINKED_LIST(L, x)
    e ← new element
    e.key ← x
    e.next ← L.head
    if e.next ≠ NIL then
        L.head.prev ← e
    end if
    L.head ← e
    return L
end function
```

b)

```
function DOUBLY_LINKED_LIST_APPEND(A, B)
    a_i ← A.head
    while a_i.next ≠ NIL do
        a_i ← a_i.next
    end while
    ▷ To keep list B intact, it seems to be not possible to simply link the
head of B to the end of A
```

good observation!

```
    b_i ← B.head
    while b_i.next ≠ NIL do
        e ← new element
        e.key ← b_i.key
        a_i.next ← e
        e.prev ← a_i
        a_i ← a_i.next
        b_i ← b_i.next
    end while
    e ← new element
    e.key ← b_i.key
    a_i.next ← e
    e.prev ← a_i
    return A
end function
```

The current algorithm has a asymptotic running time in $O(m+n)$, with the number of element of A being m and B being n.

Interestingly, if we don't care if B is intact or not, and meanwhile we

- point list L.head.prev to be the last element of L and in accordance ok
- point list L.head.prev.next to L.head,

the algorithm could be much more simpler:

function DOUBLY_LINKED_LIST_APPEND(A, B)
    $a \leftarrow A$.head.prev
    a.next $\leftarrow B$.head ▷ A's tail to B's head
    B.head.prev.next $\leftarrow A$.head ▷ B's tail to A's head
    A.head.prev $\leftarrow B$.head.prev ▷ A's head to B's tail
    B.head.prev $\leftarrow a$ ▷ B's head to A's tail
    return A
end function

This algorithm has an asymptotic running time in $O(1)$.

c)

function DOUBLY_LINKED_LIST_ZIP(A, B)
    $a_i \leftarrow A$.head
    $b_i \leftarrow B$.head
    while $a_i \neq$ NIL and $b_i \neq$ NIL **do**
        ▷ Repointing a_i.next, b_i.prev, b_i.next and a_{i+1}.prev
        $a_{i+1} \leftarrow a_i$.next
        a_i.next $\leftarrow b_i$ challenge: how can this
        b_i.prev $\leftarrow a_i$ be done with one less pointer?
        $b_{i+1} \leftarrow b_i$.next
        if $b_{i+1} \neq$ NIL **then**
            b_i.next $\leftarrow a_{i+1}$
            if $a_{i+1} \neq$ NIL **then**
                a_{i+1}.prev $\leftarrow b_i$
            end if
        end if
        ▷ Initializing the variables for the next iteration
        $a_i \leftarrow a_{i+1}$
        $b_i \leftarrow b_{i+1}$
    end while
    return A
end function