



## Problem 1

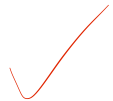
a) Search Algorithm:

```
1: function SEARCH_DOUBLY_LINKED_LIST( $L, x$ )
2:    $e \leftarrow L.head$ 
3:   while  $e.key \neq x$  and  $e \neq NIL$  do
4:      $e \leftarrow e.next$ 
5:   end while
6:   return  $e$ 
7: end function
```



Insert Algorithm:

```
1: function INSERT_DOUBLY_LINKED_LIST( $L, x$ )
2:    $e \leftarrow$  new element
3:    $e.key \leftarrow x$ 
4:    $e.next \leftarrow L.head$ 
5:   if  $e.next \neq NIL$  then
6:      $L.head.prev \leftarrow e$ 
7:   end if
8:    $L.head \leftarrow e$ 
9:   return  $L$ 
10: end function
```



b)

```
1: function DOUBLY_LINKED_LIST_APPEND( $A, B$ )
2:    $a_i \leftarrow A.head$ 
3:   while  $a_i.next \neq NIL$  do
4:      $a_i \leftarrow a_i.next$ 
5:   end while
6:    $\triangleright$  To keep list B intact, it seems to be not possible to simply link the
      head of B to the end of A
7:    $b_i \leftarrow B.head$ 
8:   while  $b_i.next \neq NIL$  do
9:      $e \leftarrow$  new element
10:     $e.key \leftarrow b_i.key$ 
11:     $a_i.next \leftarrow e$ 
12:     $e.prev \leftarrow a_i$ 
13:     $a_i \leftarrow a_i.next$ 
14:     $b_i \leftarrow b_i.next$ 
15:   end while
16:    $e \leftarrow$  new element
17:    $e.key \leftarrow b_i.key$ 
18:    $a_i.next \leftarrow e$ 
19:    $e.prev \leftarrow a_i$ 
20:   return  $A$ 
21: end function
```

good observation!

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
- point list  $L$ .head.prev.next to  $L$ .head,

ok

the algorithm could be much more simpler:

```

1: function DOUBLY_LINKED_LIST_APPEND( $A, B$ )
2:    $a \leftarrow A$ .head.prev
3:    $a$ .next  $\leftarrow B$ .head
4:    $B$ .head.prev.next  $\leftarrow A$ .head
5:    $A$ .head.prev  $\leftarrow B$ .head.prev
6:    $B$ .head.prev  $\leftarrow a$ 
7:   return  $A$ 
8: end function

```

$\triangleright$  A's tail to B's head  
 $\triangleright$  B's tail to A's head  
 $\triangleright$  A's head to B's tail  
 $\triangleright$  B's head to A's tail

✓

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

c)

```

1: function DOUBLY_LINKED_LIST_ZIP( $A, B$ )
2:    $a_i \leftarrow A$ .head
3:    $b_i \leftarrow B$ .head
4:   while  $a_i \neq \text{NIL}$  and  $b_i \neq \text{NIL}$  do
5:      $\triangleright$  Repointing  $a_i$ .next,  $b_i$ .prev,  $b_i$ .next and  $a_{i+1}$ .prev
6:      $a_{i+1} \leftarrow a_i$ .next
7:      $a_i$ .next  $\leftarrow b_i$ 
8:      $b_i$ .prev  $\leftarrow a_i$ 
9:      $b_{i+1} \leftarrow b_i$ .next
10:    if  $b_{i+1} \neq \text{NIL}$  then
11:       $b_i$ .next  $\leftarrow a_{i+1}$ 
12:      if  $a_{i+1} \neq \text{NIL}$  then
13:         $a_{i+1}$ .prev  $\leftarrow b_i$ 
14:      end if
15:    end if
16:     $\triangleright$  Initializing the variables for the next iteration
17:     $a_i \leftarrow a_{i+1}$ 
18:     $b_i \leftarrow b_{i+1}$ 
19:  end while
20:  return  $A$ 
21: end function

```

challenge: how can this be done with one less pointer?