

Assignment 1 - Data Structures & Algorithms

Tom Goodman

1 Question 1

1.1 Brief

You need to insert the numbers 2, 5, 3, 6, one at a time in that order into to an initially empty queue.

Represent that process using the standard constructors *push* and *EmptyQueue*.

Show, in the standard two-cell notation, the resulting queue.

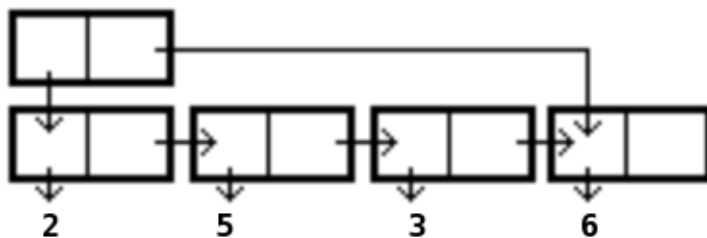
What is the result of the operation *top* on that queue?

What is the result of the operation *pop* on the original queue?

What is the result of the operation *pop* followed by *pop* followed by *top* on the original queue?

1.2 Answer

`push(6, push(3, push(5, push(2, EmptyQueue))))`



2

[5, 3, 6]

3

2 Question 2

2.1 Brief

In the lecture notes (Section 3.2) we looked at a procedure `last(L)` that returned the last item in the given list `L`. By performing the simplest possible modification of that procedure, create a recursive procedure `secondlast(L)` that returns the second to last item in a given list `L`.

What is the time complexity of your algorithm?

Now perform a more general modification of the `last(L)` procedure to give a recursive procedure `getItem(i,L)` that returns the *i*th item in a list `L`, where *i* is an integer greater than zero.

2.2 Answer

```

let secondlast(L) =
  if isEmpty(L) then
    error "The list is Empty."
  else if isEmpty(rest(L)) then
    error "The list only has 1 element."
  else if isEmpty(rest(rest(L))) then
    return first(L)
  else
    return secondlast(rest(L))

```

Time Complexity : $O(n)$

```

let getItem(i, L) =
  if i >= 0 then
    error "i should be greater than 0."
  else if i = 1 then
    return first(L)
  else
    return getItem(i--, rest(L))

```

3 Question 3

3.1 Brief

It is often useful to know whether two given lists are the equal, i.e. contain the same items in the same order. Write a recursive procedure `equalList(L1,L2)` that returns true if the two lists L1 and L2 are the same, and false if they are not. The only other procedures it may call are the standard primitive list operators *first*, *rest* and *isEmpty*.

What is the time complexity of your algorithm?

3.2 Answer

```

let equallist(L1, L2) =
  if isEmpty(L1) xor isEmpty(L2) then
    false
  else
    return (first(L1) = first(L2)) and (equallist(rest(L1), rest(L2)))

```

Time Complexity : $O(n)$

4 Question 4

4.1 Brief

A set can be represented as a list in which repeated items are not allowed and the order of the 2 items does not matter. Suppose you have sets S1 and S2 represented as linked-lists, and access to the standard list operators *first*, *rest*, and *isEmpty*.

Write a recursive procedure `member(x,S1)` that returns true if item x is in set S1, and false if it is not.

Now write a recursive procedure `subset(S1,S2)` that returns true if set S1 is a subset of set S2, and false if it is not. It is only allowed to call the standard primitive list operators *first*, *rest* and *isEmpty* and your *member* procedure.

Finally, write a procedure `equalset(S1,S2)` that returns true if set S1 is equal to set S2, and false if it is not. It is only allowed to call the standard list operators *first*, *rest* and *isEmpty* and your *member* and *subset* procedure.

4.2 Answer

```
let member(x, S1) =
  if isEmpty(S1) then
    return false
  else if x = first(S1) then
    return true
  else
    return member(x, rest(S1))

let subset(S1, S2) =
  if isEmpty(S1) then
    return true
  else if member(first(S1), S2) then
    return subset(rest(S1), S2)
  else
    return false

let equalset(S1, S2) =
  return subset(S1, S2) and subset(S2, S1)
```