CS 234 Fall 2019
Naomi Nishimura

Assignment 1

Coverage: Python session 1; Modules 1 and 2.

This assignment consists of a written component and a programming component. Please read the course website carefully to ensure that you submit each component correctly.

All the questions in the written component should be submitted as a single pdf file with the name `a1written.pdf`. Although scanned documents are acceptable, please keep in mind that only legible writing will be marked (subject to the markers' discretion) and that MarkUs forces a limit of 5000 KB per file, which might be exceeded by high resolution scans.

Several of the questions will make use of the ADT Swaplist, which stores data of any type and supports the operations listed below. Unlike for other ADTs seen in class, the CREATE operation does not create an empty ADT, but rather an ADT storing data items provided in the input to the operation (a sequence containing the data items). Moreover, instead of additions or deletions, the changes that can be made are the swapping of positions of data items.

Name	Returns	Changes
CREATE*(Sequence)*	a new swaplist storing the data items in *Sequence*, where the data item in position *Pos* in *Sequence* is placed in position *Pos* in the swaplist (positions in both *Sequence* and the swaplist start at 0)	
LENGTH*(S)*	number of items stored in swaplist *S*	
LOOK_UP*(S, Pos)*	produces the item in position *Pos* in swaplist *S*, where $0 \leq Pos <$ LENGTH*(S)*	
SWAP*(S, Pos)*		swaps items in positions *Pos* and $Pos + 1$ in swaplist *S*, where $0 \leq Pos$ and $Pos + 1 <$ LENGTH*(S)*

Written component

For full marks, you are expected to provide a brief justification of any answer you provide.

W1. *[3 marks]* Explain what the pseudocode function FIND produces and, by referring to specific lines, how it produces what it produces. The lines have been numbered for your convenience.

```
FIND(S)
INPUT:     A swaplist S
OUTPUT:    For you to find out
1  Last ← LENGTH(S)
```

```
2  for Count from 0 to Last - 1
3      Value ← LOOK_UP(S, Count)
4      if Value == Count
5          return Count
6  return False
```

W2. *[10 marks]* In the subquestions below, n is the number of data items in a swaplist S.

(a) *[4 marks]* Assuming that the worst-case cost of LENGTH is L and the worst-case cost of LOOK_UP is U, express the worst-case cost of FIND in asymptotic notation (using Θ) in terms of L, U, and n. Be sure to account for each line in the function.

(b) *[2 marks]* Give the worst-case running time of FIND in asymptotic notation when the worst-case cost of LENGTH is in $\Theta(1)$ and the worst-case cost of LOOK_UP is in $\Theta(n)$. Briefly justify your answer. (You may refer to your solution to part (a).)

(c) *[2 marks]* Give the worst-case running time of FIND in asymptotic notation when the worst-case cost of LENGTH is in $\Theta(n)$ and the worst-case cost of LOOK_UP is in $\Theta(1)$. Briefly justify your answer. (You may refer to your solution to part (a).)

(d) *[2 marks]* Give the **best-case** running time of FIND in asymptotic notation when the best-case cost of LENGTH is in $\Theta(1)$ and the best-case cost of LOOK_UP is in $\Theta(1)$. Briefly justify your answer.

W3. *[5 marks]* Using the pseudocode interface for the ADT Swaplist given above, write a function JUMP_RIGHT that consumes a swaplist S and two positions *Old* and *New*, where $0 \leq Old < New <$ LENGTH*(S)*, moves the data item in position *Old* to position *New*, and moves each data item in positions *Old*+1 through *New* one position to the left. For example, suppose that in S position 4 contained 40, position 5 contained 50, position 6 contained 60, and position 7 contained 70. Then after executing JUMP_RIGHT*(S, 4, 7)*, position 4 would contain 50, position 5 would contain 60, position 6 would contain 70, position 7 would contain 40, and data items in all other positions would remain unchanged.

W4. *[2 marks]* Suppose you were given the choice of two possible implementations to use for an operation, where one had worst-case running time in $\Omega(n)$ and the other had worst-case running time in $O(n^2)$. Can you determine whether the first or the second is a better choice? If so, explain which is better and why. If not, explain why you do not have sufficient information to make a choice.

Programming component

Please read the course website carefully to ensure that you are using the correct version of Python and the correct style. For full marks, you are required not only to have a correct solution, but also to adhere to the requirements of the assignment question and the style guide, including aspects of the design recipe.

Although submitting tests is not required, it is highly recommended that you test your code. For each assignment question, create a testing file that imports your submission and tests the code. Do not submit your testing file.

For any of the programming questions in this assignment, you may import any of the following files: `check.py` and `swaplist.py`. The file `reversemodel.py` may be used only for the second question.

The programming questions for this assignment use the ADT Swaplist, which has been implemented for you in the provided module `swaplist.py`. Your code should work with the following code interface:

```
## Swaplist(items) produces a swaplist with the items as data.
## __init__: (listof Any) -> Swaplist
def __init__(self, items):

## print(self) prints the sequence of values stored in self.
## Effects: Prints value(s)
## __str__: Swaplist -> Str
def __str__(self):

## self == other produces True if the values and length
##     of self and other match.
## __eq__: Swaplist Swaplist -> Bool
def __eq__(self, other):

## self.length() produces the length of self.
## length: Swaplist -> Int
def length(self):

## self.look_up(position) produces the data item in the
##     given position in self.
## look_up: Swaplist Int -> Any
## Requires: 0 <= position < self.length()
def look_up(self, position):

## self.swap(position) swaps data items in positions
##     position and position + 1 in self.
## Effects: Mutates self.
## swap: Swaplist Int Int -> None
## Requires: 0 <= position < self.length()-1
def swap(self, position):
```

For example, if you wish to create a new swaplist S containing the values 3, 5, 2, 4, 6 in order, use

```
S = Swaplist([3, 5, 2, 4, 6])
```

and if you wish to find the data item in position 3, use

```
S.access(3)
```

Each of your solutions to the programming questions should include the line `from swaplist import *`. **For full marks, you must not directly manipulate the fields `_data` and/or `_length` in the implementation of swaplist; use only the provided methods for access.**

P1. *[10 marks]* Write a function `reverse` that consumes a Swaplist `items` and integers `first` and `last` such that $0 \leq$ `first` $<$ `last` and `last` is less than the length of `items`. The side effect of the function will be to reverse the items in positions `first` through `last`, inclusive. For example, suppose that in `items` position 4 contained 40, position 5 contained 50, position 6 contained 60, and position 7 contained 70. Then after executing `reverse(items, 4, 7)`, position 4 would contain 70, position 5 would contain 60, position 6 would contain 50, position 7 would contain 40, and data items in all other positions would remain unchanged.

Submit your work in a file with the name `reverse.py`.

P2. *[10 marks]* Write a function `big_swap` that consumes a swaplist `items` and positions `first` and `last` such that $0 \leq$ `first` $<$ `last` and `last` is less than the length of `items`. The side effect of the function will be to swap the items in positions `first` and `last`. For example, suppose that in `items` position 4 contained 40 and position 7 contained 70. Then after executing `big_swap(items, 4, 7)`, position 4 would contain 70, position 7 would contain 40, and data items in all other positions would remain unchanged.

If you wish to use `reverse` in your solution, you may do so even if you are not able to complete programming question 1. To use the model solution for `reverse`, import the file `reversemodel.py` to use our solution to P1 by including the line `from reversemodel import *`.

Note, however, that using `reverse` will make your program difficult to debug (you'll have to insert correct answers instead of relying on the function). Moreover, there is another approach that doesn't require `reverse`, so you might prefer that option.

Submit your work in a file with the name `bigswap.py`.