

Exercise sheet 5

Problem 1: Array Lists and Linked Lists Python Implementation

Prepared by: Ashik Emon

(a) Array List insert(i, x)

Python implementation:

```
class ArrayList:
    def __init__(self, initial_capacity=4):
        self.cap = max(1, initial_capacity)
        self.arr = [None] * self.cap
        self.n = 0

    def __len__(self):
        return self.n

    def _ensure_capacity(self, min_capacity):
        if self.cap >= min_capacity:
            return
        new_cap = max(self.cap * 2, min_capacity, 1)
        new_arr = [None] * new_cap
        for k in range(self.n):
            new_arr[k] = self.arr[k]
        self.arr = new_arr
        self.cap = new_cap

    def insert(self, i, x):
        # Insert x at index i (0 ≤ i ≤ n). After insertion, element previously at i shifts to i+1.
        # Worst-case time: Θ(n)
        n = self.n
        if i < 0 or i > n:
            raise IndexError("insert index out of range")
        if n == self.cap:
            self._ensure_capacity(n + 1)
        for k in range(n - 1, i - 1, -1): # shift right
            self.arr[k + 1] = self.arr[k]
        self.arr[i] = x
        self.n += 1

    def to_list(self):
        return self.arr[:self.n]
```

Worst-case running time: $\Theta(n)$ due to element shifting (resizing is also $\Theta(n)$ but does not change the order).

(b) Swap a given node with its next in a singly-linked list

Python implementation:

```
class Node:
    __slots__ = ("key", "next")
    def __init__(self, key, next=None):
        self.key = key
        self.next = next

class SinglyLinkedList:
    def __init__(self, iterable=None):
        self.head = None
```

```

if iterable is not None:
    for x in reversed(list(iterable)):
        self.head = Node(x, self.head)

def to_list(self):
    out = []
    cur = self.head
    while cur:
        out.append(cur.key)
        cur = cur.next
    return out

def swap_with_next(self, p):
    # Swap node p with its next node in-place. Return True if swapped, else False.
    # Worst-case time:  $\Theta(n)$  to locate previous node ( $\Theta(1)$  if previous is known).
    if p is None or p.next is None:
        return False
    if self.head is p: # p is head
        a, b = p, p.next
        a.next = b.next
        b.next = a
        self.head = b
        return True
    prev = self.head
    while prev is not None and prev.next is not p:
        prev = prev.next
    if prev is None:
        return False # p not found
    a, b = p, p.next
    prev.next = b
    a.next = b.next
    b.next = a
    return True

def merge_inplace_from(self, other):
    # Merge sorted list 'other' into self (also sorted) to obtain a combined sorted list.
    # Afterward, self becomes the merged list and other becomes empty.
    # Worst-case time:  $\Theta(n_A + n_B)$ ; space:  $\Theta(1)$ .
    pa, pb = self.head, other.head
    dummy = Node(None)
    tail = dummy
    while pa is not None and pb is not None:
        if pa.key <= pb.key:
            tail.next = pa
            pa = pa.next
        else:
            tail.next = pb
            pb = pb.next
        tail = tail.next
    tail.next = pa if pa is not None else pb
    self.head = dummy.next
    other.head = None

```

Worst-case running time: $\Theta(n)$ for `swap_with_next` (search `prev`); $\Theta(n_A + n_B)$ for merge; both $\Theta(1)$ extra space.

Example usage (for quick verification)

```

# (a)
A = ArrayList()
for v in [1, 3, 7]:
    A.insert(len(A), v)
A.insert(1, 2) # A: [1, 2, 3, 7]

# (b)
L = SinglyLinkedList([10, 20, 30, 40])
p = L.head.next # node with key 20
L.swap_with_next(p) # L becomes [10, 30, 20, 40]

print(L.to_list()) # -> [10, 30, 20, 40]

# (c)
A_list = SinglyLinkedList([1, 3, 7])
B_list = SinglyLinkedList([2, 4, 5])
A_list.merge_inplace_from(B_list)

print(A_list.to_list()) # -> [1, 2, 3, 4, 5, 7]
print(B_list.to_list()) # -> []

```