SE274 Midterm Exam (40 points total)

Use only pen and paper (or word processor) for answering the questions 1, 2, 3. For the question 4, you should present a complete working python code. You may look up the textbook, lecture slides, and materials, but **DO NOT SEARCH the internet** for answering those questions, **ON YOUR HONOR.**

For reporting, submit a `StudentID-Name.zip` package with two files in:

1. A PDF document that answers the following questions.
2. A `skiplist.py` for the question 4.

The submission due date is May 4th, 23:59.

0. Honor Code (5 points)

Submit an honor code (check the announcement on the course web page), if you haven't done it. If you already submit it, you've already earned 5 points.

1. Linked List (5 points)

Give a recursive algorithm for reversing a singly linked list (present a pseudo-code).

2. Stack (5 points)

Postfix notation is an unambiguous way of writing an arithmetic expression without parentheses. It is defined so that if ***(exp1) op (exp2)*** is a normal, fully parenthesized expression whose operation is op, the postfix version of this is ***exp1 exp2 op***.

For example, the *postfix* version of **((5+2) ∗ (8−3))/4** is **5 2 + 8 3 − ∗ 4 /**.

Describe a nonrecursive way of evaluating an expression in *postfix* notation, using a stack S.

3. Heap (5 points)

Using an array-based heap representation, fill in the items in the array after the following priority-queue operations, one-by-one.

1. `add(4)      -> [   ]`
2. `add(9)      -> [   ,   ]`
3. `add(1)      -> [   ,   ,   ]`
4. `add(5)      -> [   ,   ,   ,   ]`
5. `remove_min -> [   ,   ,   ] / (   ) is returned.`
6. `add(6)      -> [   ,   ,   ,   ]`
7. `add(3)      -> [   ,   ,   ,   ,   ]`
8. `add(7)      -> [   ,   ,   ,   ,   ,   ]`
9. `remove_min -> [   ,   ,   ,   ,   ] / (   ) is returned.`
10. `remove_min -> [   ,   ,   ,   ] / (   ) is returned.`
11. `remove_min -> [   ,   ,   ] / (   ) is returned.`

4. Skip List (20 points)

On `skiplist.py`, fully implement a skip list that completes `MutableMapping` ADT. Specifically, fill in the `__getitem__`, `__setitem__`, and `__delitem__`. Feel free to add supplimentatry methods if needed.

With your implementation, report the following questions.

1. Analyze your logic in terms of time complexity. get, set, del functions should be run in O(logn)O(logn) time.
2. Experimentally show your `__setitem__` runs in O(logn)O(logn) time. For example, measure the elapsed time of adding 100, 200, 300, ... , 10000 random items to your skiplist. The tested numbers may vary depend on your computer's performance. Plot them, with x-axis set to the problem size and y-axis set to the elapsed time. The trend should follow lognlogn shape.

A guide for the implementation

`_Node` is a class for representing one column in a skip list. It has `_key` and `_value` as a key-value pair for a Map. Additionally, it has `_next` list that stores links to the next nodes.

For example, We can consider a skip list having 8 items (+2 items for -inf and +inf):

```
At height #3, -inf -------------------------------------------> +inf
At height #2, -inf ------------------------------> 16 -------> +inf
At height #1, -inf -> 0 -> 1 ----------------> 7 -> 16 -------> +inf
At height #0, -inf -> 0 -> 1 -> 2 -> 3 -> 6 -> 7 -> 16 -> 17 -> +inf
```

This could be represneted with 10 nodes, as follow.

```
Node     key    h=0 h=1 h=2 h=3
-------------------------------
Node0 (-inf):   1   1   7   9
Node1    (0):   2   2   .   .
Node2    (1):   3   6   .   .
Node3    (2):   4   .   .   .
Node4    (3):   5   .   .   .
Node5    (6):   6   .   .   .
Node6    (7):   7   7   .   .
Node7   (16):   8   9   9   .
Node8   (17):   9   .   .   .
Node9 (+inf):   .   .   .   .
-------------------------------
* . represents None
```

All the nodes should be the same height (= the size of `_next` list should be maintained same all across the nodes).

On `Node0`, `_next` list holds object references to the next node at each height, `[Node1, Node1, Node7, Node9]`.

On `Node1`, `_next` list holds object references to `[Node2, Node2, None, None]`, and so on.

Note that the higest height (= #3 in the example) should hold only two special keys, -inf -> inf, and the lowest height (= #0) should hold all the items.

The height in the nodes (=the size of `_next` list) should be increased or decreased according to the add/remove operations, if needed.

This is a test code and its printed results, after the skiplist is fully implemented. Make sure all the edge cases are handled, exception should be raised when error occurs (look at the comments in the code), and all the Map functions are functional. You can assume that all the keys are integers.

Test Code

```
from skiplist import SkipList

L = SkipList()
last_key = 0
for i in range(30):
    key = random.randrange(20)
    value = random.randrange(10000)
    print(f'Adding L[{key}] = {value}')
    L[key] = value
    last_key = key
L.print_tree()

print(L[last_key], "# This should be the same to the lastly inserted value")
L[last_key] = 123456789
print(L[last_key], "# This should be 123456789")

print('\nLet\'s iterate through the keys')
for key in L.keys():
    print(key)

print('\nLet\'s delete all the keys, one by one')

del_order = random.sample(list(L), int(len(L)/2))
for k in del_order:
    print(f"Deleting the key {k}")
    del L[k]

L.print_tree()

del_order = random.sample(list(L), len(L))
for k in del_order:
    print(f"Deleting the key {k}")
    del L[k]
L.print_tree()
```

Sample Result

```
Adding L[12] = 6655
Adding L[12] = 9613
Adding L[7] = 2793
Adding L[2] = 4267
Adding L[9] = 8344
Adding L[17] = 4223
Adding L[3] = 4269
Adding L[11] = 8662
Adding L[12] = 7621
Adding L[6] = 2392
Adding L[15] = 1788
Adding L[12] = 3202
Adding L[5] = 6223
Adding L[12] = 7495
Adding L[10] = 5739
Adding L[14] = 7538
Adding L[8] = 9488
Adding L[6] = 7008
Adding L[10] = 6564
Adding L[15] = 6987
Adding L[8] = 8383
Adding L[14] = 4467
Adding L[6] = 8983
Adding L[2] = 8682
Adding L[14] = 3287
Adding L[2] = 5148
Adding L[2] = 7246
Adding L[14] = 3402
Adding L[13] = 6375
Adding L[4] = 3698
^^^^^^^^^^^^^^^^^^^^^^^^^^
#   -   -   -   -   -   -
#   2   2   2   .   .   .
#   3   .   .   .   .   .
#   4   4   4   .   .   .
#   5   .   .   .   .   .
#   6   6   .   .   .   .
#   7   7   7   .   .   .
#   8   8   8   .   .   .
#   9   .   .   .   .   .
#   10  .   .   .   .   .
#   11  .   .   .   .   .
#   12  .   .   .   .   .
#   13  .   .   .   .   .
#   14  .   .   .   .   .
#   15  .   .   .   .   .
#   17  17  17  17  17  .
#   +   +   +   +   +   +
At height #5, -inf -> inf ->
At height #4, -inf -> 17 -> inf ->
At height #3, -inf -> 17 -> inf ->
At height #2, -inf -> 2 -> 4 -> 7 -> 8 -> 17 -> inf ->
At height #1, -inf -> 2 -> 4 -> 6 -> 7 -> 8 -> 17 -> inf ->
At height #0, -inf -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> 8 -> 9 -> 10 -> 11 -> 12 ->
13 -> 14 -> 15 -> 17 -> inf ->
vvvvvvvvvvvvvvvvvvvvvvvvvv
3698 # This should be the same to the lastly inserted value
123456789 # This should be 123456789

Let's iterate through the keys
2
3
4
5
6
7
8
9
```

```
10
11
12
13
14
15
17

Let's delete all the keys, one by one
Deleting the key 13
Deleting the key 5
Deleting the key 17
Deleting the key 4
Deleting the key 15
Deleting the key 9
Deleting the key 14
^^^^^^^^^^^^^^^^^^^^^^^^^^
#   -   -   -   -
#   2   2   2   .
#   3   .   .   .
#   6   6   .   .
#   7   7   7   .
#   8   8   8   .
#   10  .   .   .
#   11  .   .   .
#   12  .   .   .
#   +   +   +   +
At height #3, -inf -> inf ->
At height #2, -inf -> 2 -> 7 -> 8 -> inf ->
At height #1, -inf -> 2 -> 6 -> 7 -> 8 -> inf ->
At height #0, -inf -> 2 -> 3 -> 6 -> 7 -> 8 -> 10 -> 11 -> 12 -> inf ->
vvvvvvvvvvvvvvvvvvvvvvvvvv
Deleting the key 11
Deleting the key 2
Deleting the key 8
Deleting the key 3
Deleting the key 6
Deleting the key 7
Deleting the key 12
Deleting the key 10
^^^^^^^^^^^^^^^^^^^^^^^^^^
#   -
#   +
At height #0, -inf -> inf ->
vvvvvvvvvvvvvvvvvvvvvvvvvv
```