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).

```
Algorithm reverse(n): #n: size of singly linked list
    head = last_node
    if n == 0:
        get(n).next = tail
    else:
        get(n).next = get(n-1)
        reverse(n-1)
```

---

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.

예시를 참조하여 Postfix 방법을 stack을 이용해 구현해보자. 구현에 있어 사용한 규칙은 다음과 같다.

1. operand의 경우 stack에 차례대로 push한다.
2. operator의 경우 stack에 있는 두 operand를 차례로 pop하여 이를 임의의 pop_list에 push한다. (example, S = [1, 2] pop_list = [2, 1]) 이후 pop_list에서 operand - operator(*postfix_list*) - operand 순의 pop를 통해 연산을 마친다. 이후 그 결과를 다시 stack에 push한다.

S = [5]

S = [5, 2]

S = [7] # '+' operater의 순서이기 때문에, S에서 두 oprend를 pop, pop_list에 push --> pop_list = [2, 5] 이후 pop_list.push(), operator, pop_list.push()를 통해 5 + 2 = 7 연산 실행. 마지막으로 7을 S에 push

S = [7, 8]

S = [7, 8, 3] # '-' operand 출현, 위의 규칙에 따라 진행

S = [7, 5]

S = [35]

S = [35, 4]

S = [35/4]

---

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

__getitem__의 경우 do_find(k)라는 메서드를 활용하여 구현하였다. do_find(k) 메서드는 최상단 height층에서 시작하여 현 node의 key값이 해당 level의 다른 node들의 key값과 비교하여 작을 경우, 해당 level의 한 계층 밑으로 내려가 같은 key값을 찾을 때까지 반복하는 알고리즘이다. 이를 활용하여 __getitem__을 구현하였다. __setitem__에서 random access를 통해 각 height를 구성하는 과정이 있기 때문에, 전체 height = log(n)을 따른다. 따라서 __getitem__ 메서드의 경우, 해당 작업을 실행하는데 O(logn) time이 소요된다. __delitem__의 경우 __getitem__의 아이디어에서 착안하여, 해당 node를 delete하는 과정이 추가되었으므로, O(logn) time에 실행 가능하다.

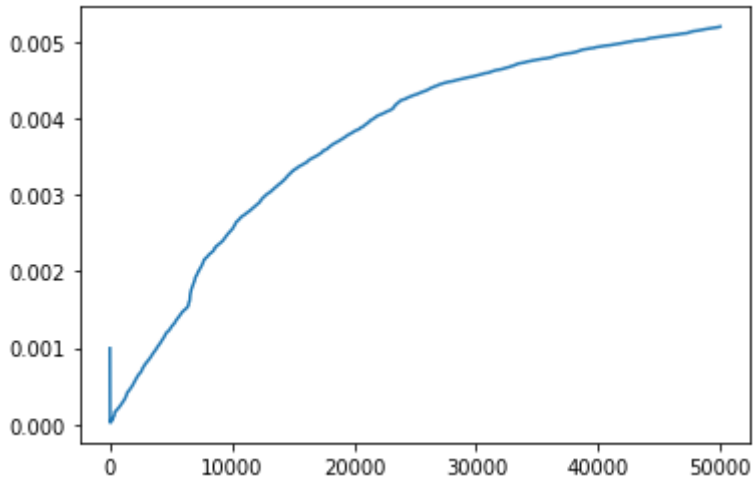
2. Experimentally show your `__setitem__` runs in O(logn) time. For example, measure the elapsed time of adding 100, 200, 300, ... , 10000 random items to your skiplist. The tested numbers may vary depend on your computer's performance. Plot them, with x-axis set to the problem size and y-axis set to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

실행 시간은 다음과 같다.

```
import time
import matplotlib.pyplot as plt
import math
import random

from skiplist import SkipList
SL = SkipList()
start = time.time()
x_axis = []
y_axis = []
for i in range(50000):
    key = random.randrange(10000)
    value = 1
    SL[key] = value
    if i % 10 == 0:
        x_axis.append(i)
        y_axis.append((time.time()-start)/(i+1))

plt.plot(x_axis, y_axis)
plt.show()
```



그래프의 진행 개형이 logn의 추세를 띠는 것으로 보인다. 따라서 실험적으로 이 Skiplist의 performance가 O(logn)임을 보일 수 있다.