Lab2

Multi-Choices

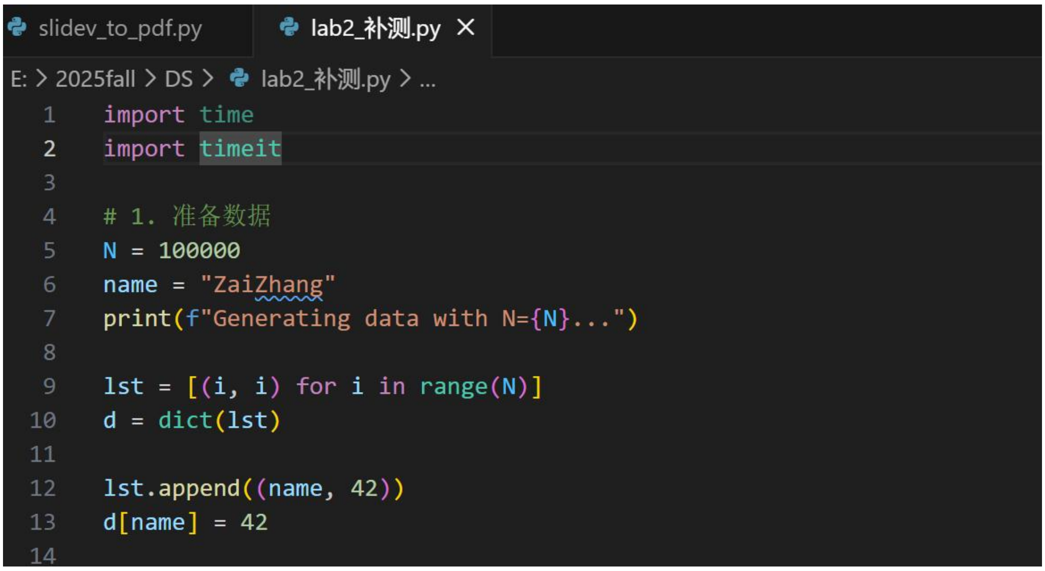
1. A，2.B，3.C

True or False

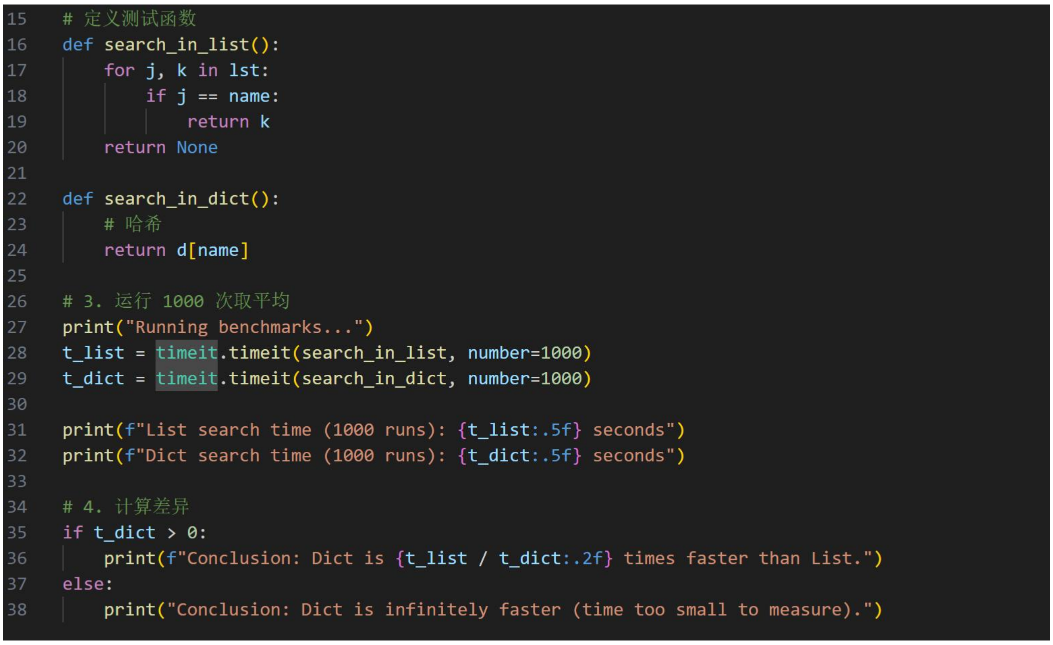
1. T，2.T，3.F

Evaluation

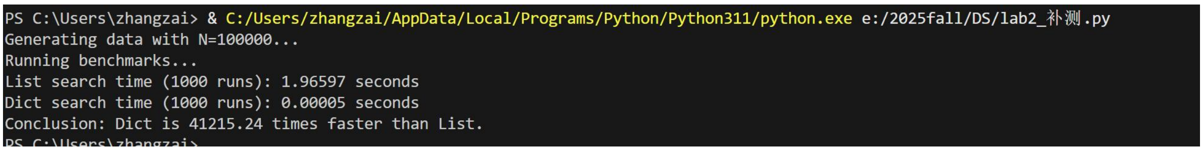
7.



```python
import time
import timeit

# 1. 准备数据
N = 100000
name = "ZaiZhang"
print(f"Generating data with N={N}...")

lst = [(i, i) for i in range(N)]
d = dict(lst)

lst.append((name, 42))
d[name] = 42

```

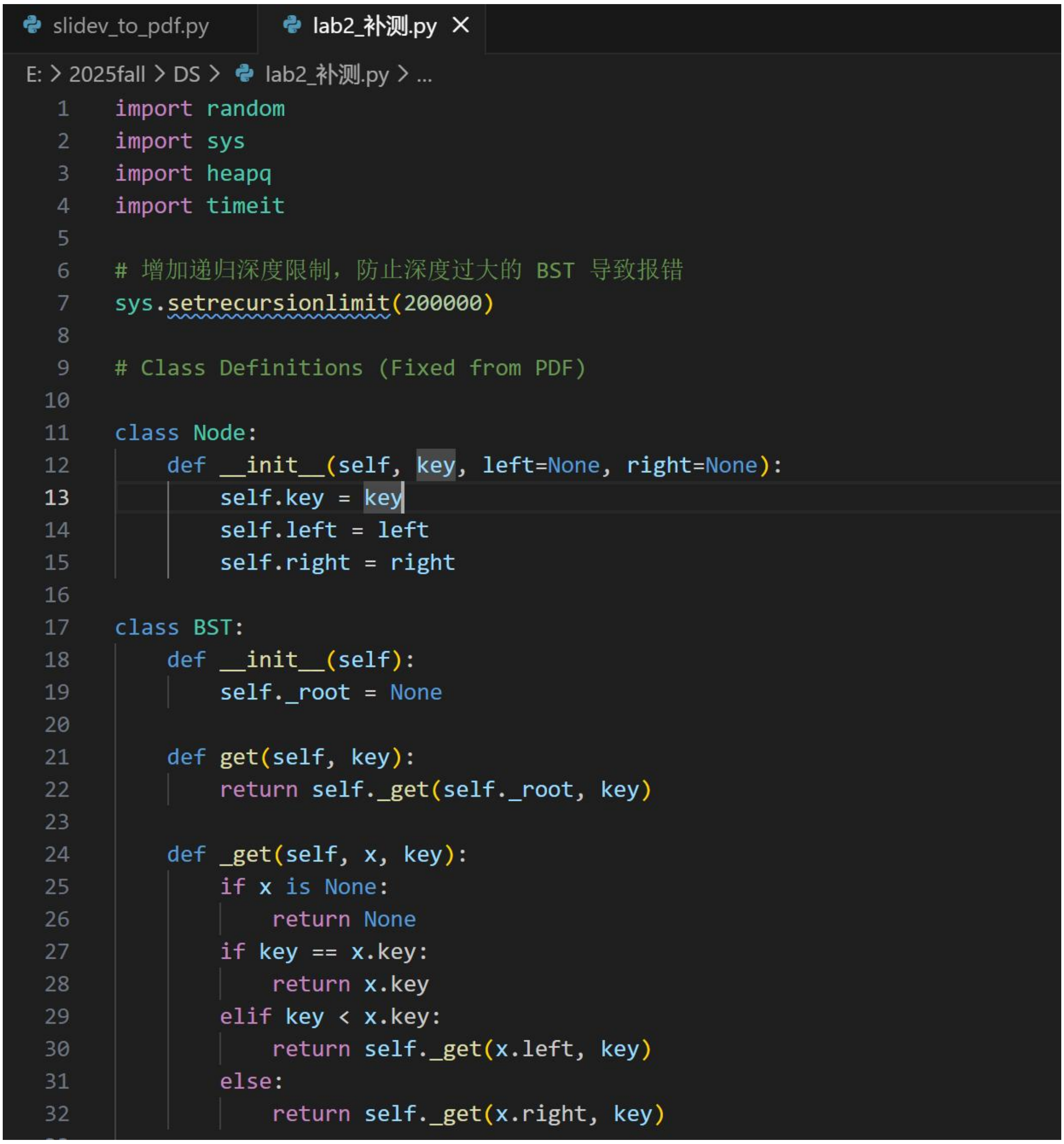


```python
# 定义测试函数
def search_in_list():
    for j, k in lst:
        if j == name:
            return k
    return None

def search_in_dict():
    # 哈希
    return d[name]

# 3. 运行 1000 次取平均
print("Running benchmarks...")
t_list = timeit.timeit(search_in_list, number=1000)
t_dict = timeit.timeit(search_in_dict, number=1000)

print(f"List search time (1000 runs): {t_list:.5f} seconds")
print(f"Dict search time (1000 runs): {t_dict:.5f} seconds")

# 4. 计算差异
if t_dict > 0:
    print(f"Conclusion: Dict is {t_list / t_dict:.2f} times faster than List.")
else:
    print("Conclusion: Dict is infinitely faster (time too small to measure).")
```

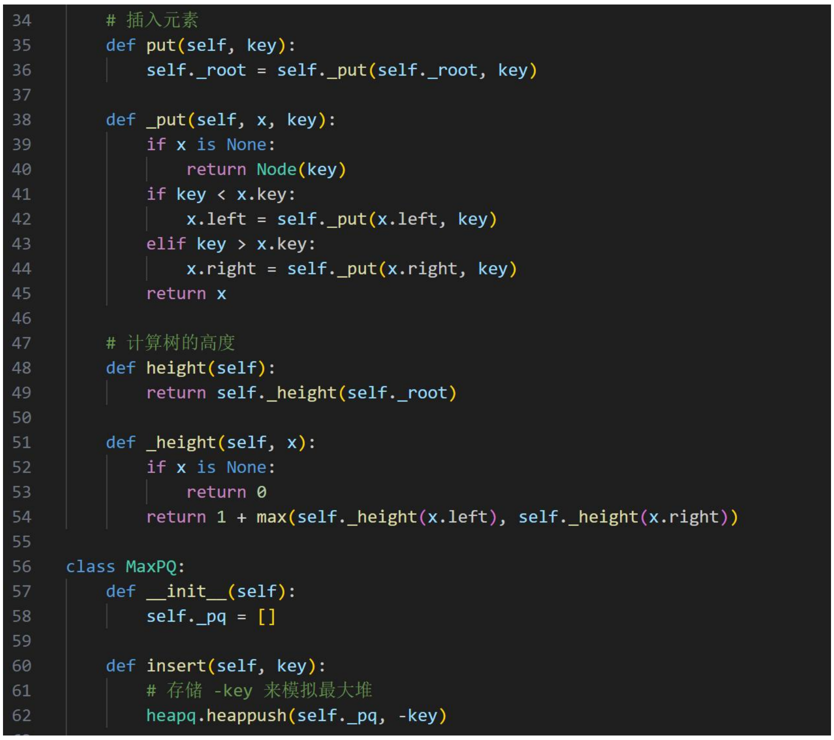


```
PS C:\Users\zhangzai> & C:/Users/zhangzai/AppData/Local/Programs/Python/Python311/python.exe e:/2025fall/DS/lab2_补测.py
Generating data with N=100000...
Running benchmarks...
List search time (1000 runs): 1.96597 seconds
Dict search time (1000 runs): 0.00005 seconds
Conclusion: Dict is 41215.24 times faster than List.
PS C:\Users\zhangzai>
```

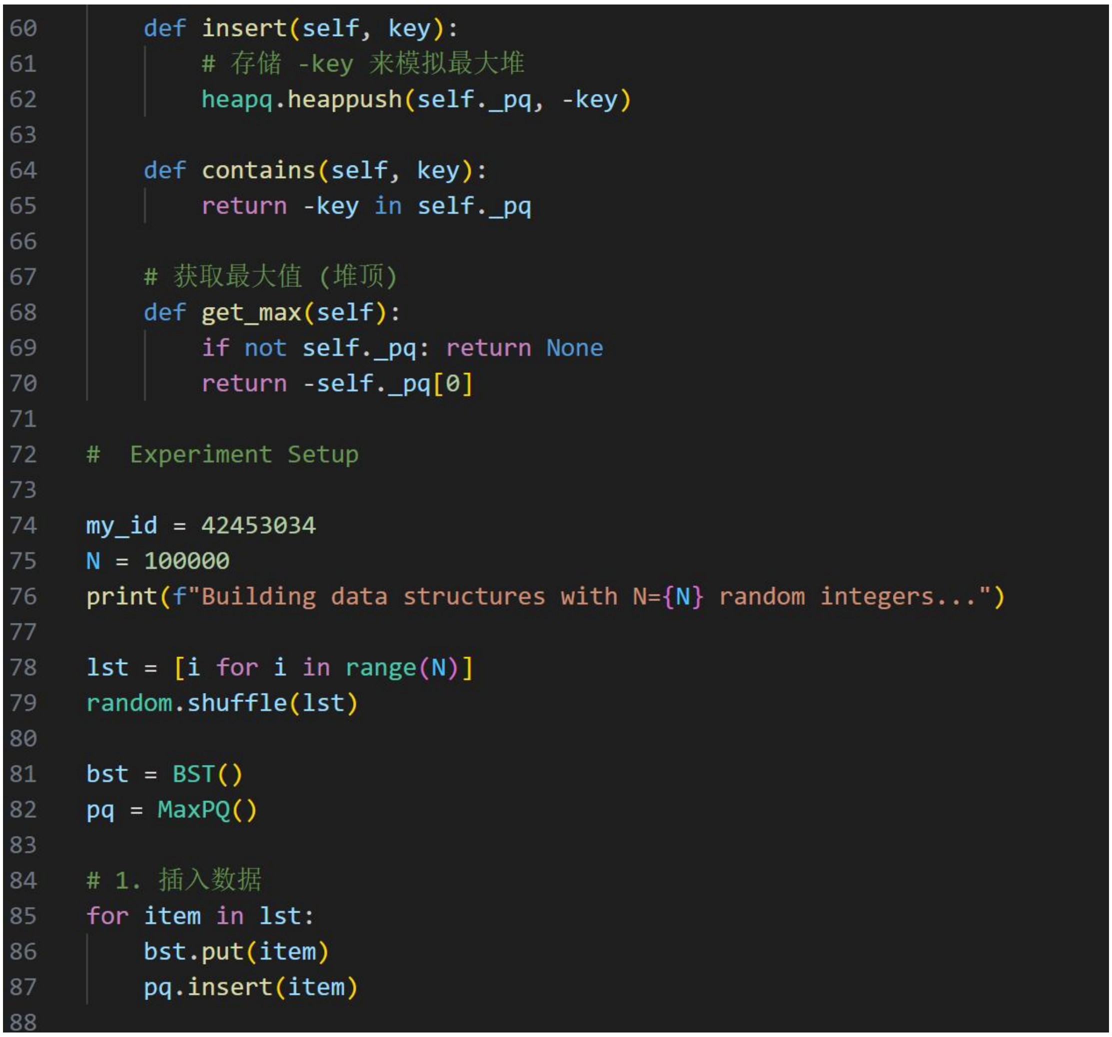
8.



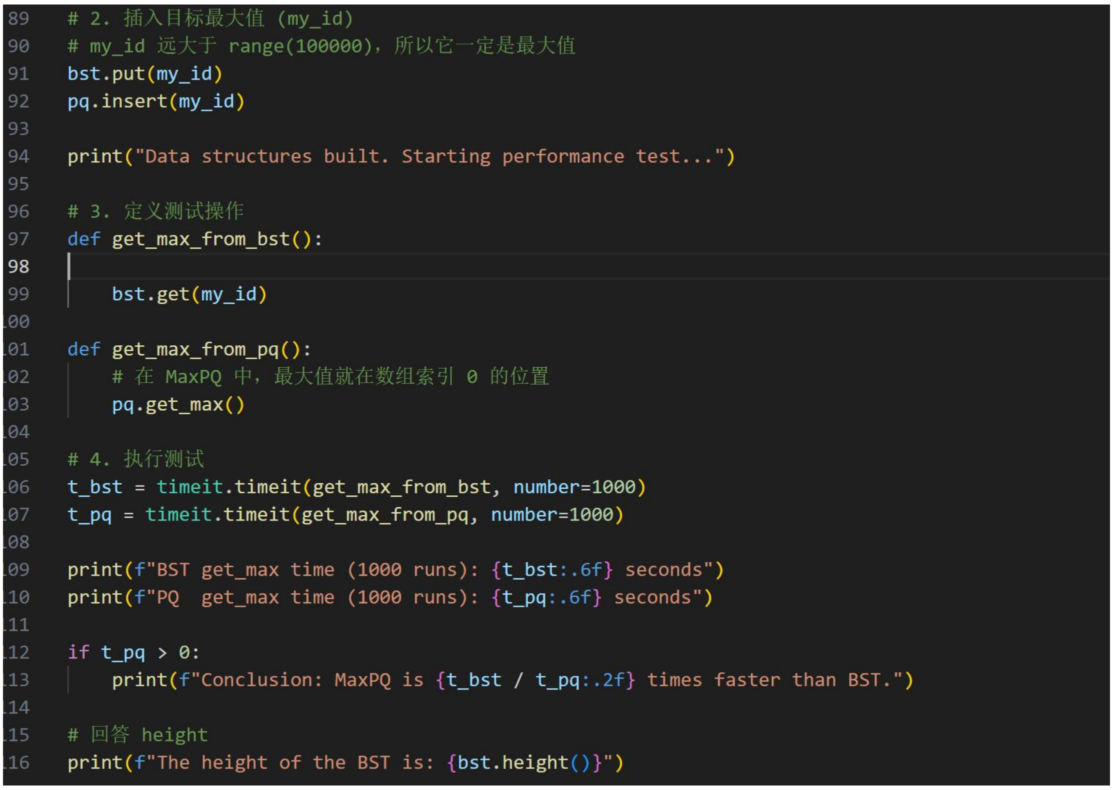
```python
import random
import sys
import heapq
import timeit

# 增加递归深度限制，防止深度过大的 BST 导致报错
sys.setrecursionlimit(200000)

# Class Definitions (Fixed from PDF)

class Node:
    def __init__(self, key, left=None, right=None):
        self.key = key
        self.left = left
        self.right = right

class BST:
    def __init__(self):
        self._root = None

    def get(self, key):
        return self._get(self._root, key)

    def _get(self, x, key):
        if x is None:
            return None
        if key == x.key:
            return x.key
        elif key < x.key:
            return self._get(x.left, key)
        else:
            return self._get(x.right, key)
```



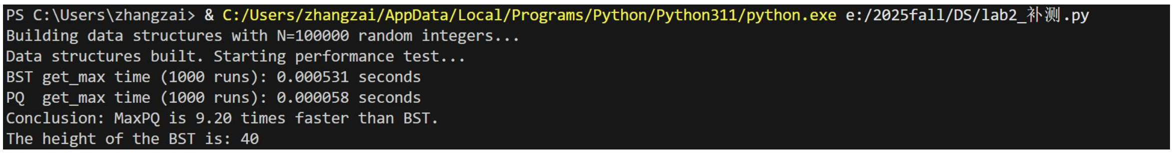
```python
    # 插入元素
    def put(self, key):
        self._root = self._put(self._root, key)

    def _put(self, x, key):
        if x is None:
            return Node(key)
        if key < x.key:
            x.left = self._put(x.left, key)
        elif key > x.key:
            x.right = self._put(x.right, key)
        return x

    # 计算树的高度
    def height(self):
        return self._height(self._root)

    def _height(self, x):
        if x is None:
            return 0
        return 1 + max(self._height(x.left), self._height(x.right))

class MaxPQ:
    def __init__(self):
        self._pq = []

    def insert(self, key):
        # 存储 -key 来模拟最大堆
        heapq.heappush(self._pq, -key)
```



```python
    def insert(self, key):
        # 存储 -key 来模拟最大堆
        heapq.heappush(self._pq, -key)

    def contains(self, key):
        return -key in self._pq

    # 获取最大值 (堆顶)
    def get_max(self):
        if not self._pq: return None
        return -self._pq[0]

#  Experiment Setup

my_id = 42453034
N = 100000
print(f"Building data structures with N={N} random integers...")

lst = [i for i in range(N)]
random.shuffle(lst)

bst = BST()
pq = MaxPQ()

# 1. 插入数据
for item in lst:
    bst.put(item)
    pq.insert(item)
```



```python
# 2. 插入目标最大值 (my_id)
# my_id 远大于 range(100000), 所以它一定是最大值
bst.put(my_id)
pq.insert(my_id)

print("Data structures built. Starting performance test...")

# 3. 定义测试操作
def get_max_from_bst():

    bst.get(my_id)

def get_max_from_pq():
    # 在 MaxPQ 中, 最大值就在数组索引 0 的位置
    pq.get_max()

# 4. 执行测试
t_bst = timeit.timeit(get_max_from_bst, number=1000)
t_pq = timeit.timeit(get_max_from_pq, number=1000)

print(f"BST get_max time (1000 runs): {t_bst:.6f} seconds")
print(f"PQ  get_max time (1000 runs): {t_pq:.6f} seconds")

if t_pq > 0:
    print(f"Conclusion: MaxPQ is {t_bst / t_pq:.2f} times faster than BST.")

# 回答 height
print(f"The height of the BST is: {bst.height()}")
```



```
PS C:\Users\zhangzai> & C:/Users/zhangzai/AppData/Local/Programs/Python/Python311/python.exe e:/2025fall/DS/lab2_补测.py
Building data structures with N=100000 random integers...
Data structures built. Starting performance test...
BST get_max time (1000 runs): 0.000531 seconds
PQ  get_max time (1000 runs): 0.000058 seconds
Conclusion: MaxPQ is 9.20 times faster than BST.
The height of the BST is: 40
```