Assignment #4: 排序、栈、队列和树

Updated 0005 GMT+8 March 11, 2024

2024 spring, Complied by ==同学的姓名、院系==

说明：

1）The complete process to learn DSA from scratch can be broken into 4 parts:

Learn about Time complexities, learn the basics of individual Data Structures, learn the basics of Algorithms, and practice Problems.

2）请把每个题目解题思路（可选），源码Python, 或者C++（已经在Codeforces/Openjudge上AC），截图（包含Accepted），填写到下面作业模版中（推荐使用 typora https://typoraio.cn ，或者用word）。AC 或者没有AC，都请标上每个题目大致花费时间。

3）提交时候先提交pdf文件，再把md或者doc文件上传到右侧“作业评论”。Canvas需要有同学清晰头像、提交文件有pdf、"作业评论"区有上传的md或者doc附件。

4）如果不能在截止前提交作业，请写明原因。

编程环境

==（请改为同学的操作系统、编程环境等）==

操作系统：macOS Ventura 13.4.1 (c)

Python编程环境：Spyder IDE 5.2.2, PyCharm 2023.1.4 (Professional Edition)

C/C++编程环境：Mac terminal vi (version 9.0.1424), g++/gcc (Apple clang version 14.0.3, clang-1403.0.22.14.1)

1. 题目

05902: 双端队列

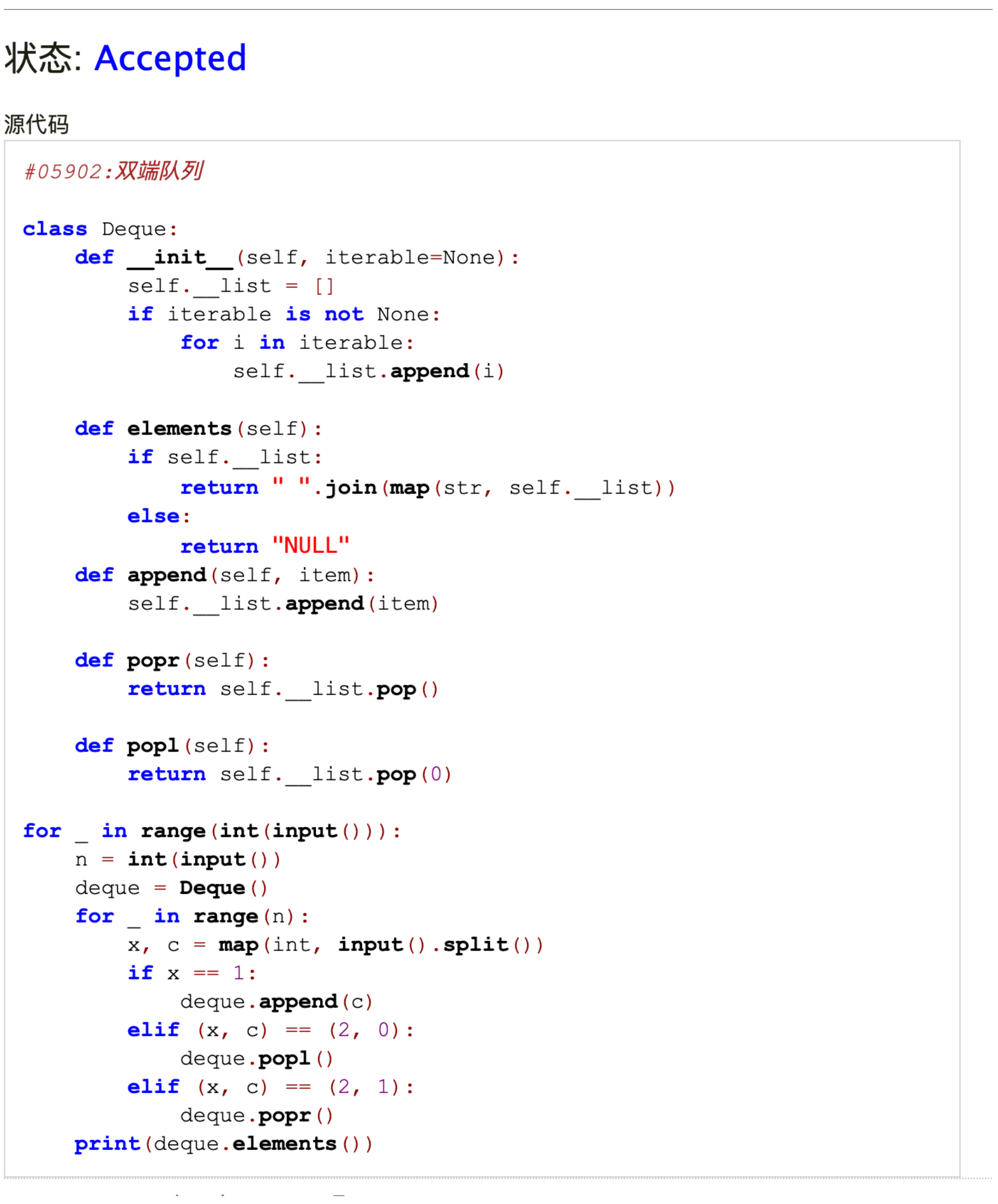
http://cs101.openjudge.cn/practice/05902/

思路：
手动实现了Deque类，但实际上的Deque实现要更复杂，此处未考虑性能问题。

代码

```python
class Deque:
    def __init__(self, iterable=None):
        self.__list = []
        if iterable is not None:
            for i in iterable:
                self.__list.append(i)

    def elements(self):
        if self.__list:
            return " ".join(map(str, self.__list))
        else:
            return "NULL"
    def append(self, item):
        self.__list.append(item)

    def popr(self):
        return self.__list.pop()

    def popl(self):
        return self.__list.pop(0)

for _ in range(int(input())):
    n = int(input())
    deque = Deque()
    for _ in range(n):
        x, c = map(int, input().split())
        if x == 1:
            deque.append(c)
        elif (x, c) == (2, 0):
            deque.popl()
        elif (x, c) == (2, 1):
            deque.popr()
    print(deque.elements())
```

代码运行截图 ==（至少包含有"Accepted"）==



状态: Accepted

源代码

```python
#05902:双端队列

class Deque:
    def __init__(self, iterable=None):
        self.__list = []
        if iterable is not None:
            for i in iterable:
                self.__list.append(i)

    def elements(self):
        if self.__list:
            return " ".join(map(str, self.__list))
        else:
            return "NULL"
    def append(self, item):
        self.__list.append(item)

    def popr(self):
        return self.__list.pop()

    def popl(self):
        return self.__list.pop(0)

for _ in range(int(input())):
    n = int(input())
    deque = Deque()
    for _ in range(n):
        x, c = map(int, input().split())
        if x == 1:
            deque.append(c)
        elif (x, c) == (2, 0):
            deque.popl()
        elif (x, c) == (2, 1):
            deque.popr()
    print(deque.elements())
```

02694: 波兰表达式

http://cs101.openjudge.cn/practice/02694/

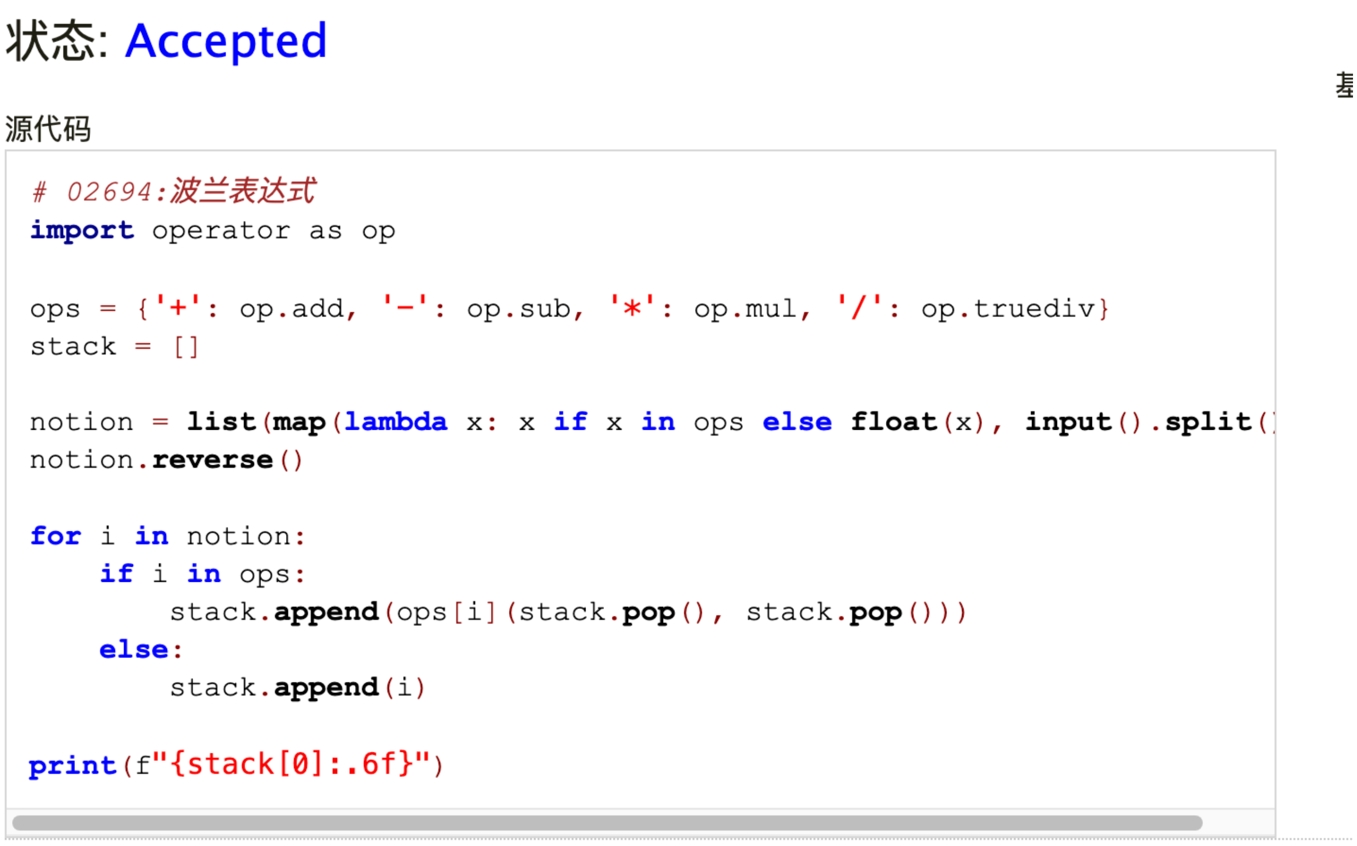
思路:

使用栈

代码

```python
import operator as op

ops = {'+': op.add, '-': op.sub, '*': op.mul, '/': op.truediv}
stack = []

notion = list(map(lambda x: x if x in ops else float(x), input().split()))
notion.reverse()

for i in notion:
    if i in ops:
        stack.append(ops[i](stack.pop(), stack.pop()))
    else:
        stack.append(i)

print(f"{stack[0]:.6f}")
```

代码运行截图 ==（至少包含有"Accepted"）==



状态: Accepted

源代码

```python
# 02694:波兰表达式
import operator as op

ops = {'+': op.add, '-': op.sub, '*': op.mul, '/': op.truediv}
stack = []

notion = list(map(lambda x: x if x in ops else float(x), input().split(
notion.reverse()

for i in notion:
    if i in ops:
        stack.append(ops[i](stack.pop(), stack.pop()))
    else:
        stack.append(i)

print(f"{stack[0]:.6f}")
```

24591: 中序表达式转后序表达式

http://cs101.openjudge.cn/practice/24591/

思路：我尝试使用树的结构和递归处理，但输出答案的运算符顺序与标答有所不同，最终参考了题解。

我的代码：和题解思路不同导致输出正确但顺序与测试数据不同的答案

```python
#24591:中序表达式转后序表达式

class Expression:
    def __init__(self, is_op, *args):
        self.__is_op = is_op
        if self.__is_op:
            self.__op = args[0]
            self.__paras = (args[1], args[2])
        else:
            self.__value = args[0]

    def display(self):
        if self.__is_op:
            return f"{self.__paras[0].display()} {self.__paras[1].display()} {self.__op}"
        if isinstance(self.__value, Expression):
            return self.__value.display()
        return self.__value

def str_parser(string):
    res = []
    tmp = []
    for char in string:
        if char in {'+', '-', '*', '/', '(', ')'}:
            if tmp:
                res.append(''.join(tmp))
            res.append(char)
            tmp = []
        else:
            tmp.append(char.strip())
    if tmp:
        res.append(''.join(tmp))
    return res

def expression_parser(exp):
    length = len(exp)
    if length == 1:
        return Expression(False, exp[0])
    for index in range(length):
        if exp[index] == '(':
            count = 1
            for rindex in range(index + 1, length):
                if exp[rindex] == '(':
                    count += 1
                elif exp[rindex] == ')':
                    count -= 1
                    if count == 0:
                        return expression_parser([expression_parser(exp[index + 
1:rindex])] + exp[rindex + 1:])
        if exp[index] in {'+', '-'}:
            return Expression(True, exp[index], expression_parser(exp[:index]), 
expression_parser(exp[index + 1:]))
        if exp[index] in {'*', '/'}:
            return Expression(True, exp[index], expression_parser(exp[:index]), 
expression_parser(exp[index + 1:]))

test = str_parser("1+1+1+1+1+2*3")
test = expression_parser(test)
print(test.display())
```

使用的题解代码：

```
def infix_to_postfix(expression):
    precedence = {'+':1, '-':1, '*':2, '/':2}
    stack = []
    postfix = []
    number = ''

    for char in expression:
        if char.isnumeric() or char == '.':
            number += char
        else:
            if number:
                num = float(number)
                postfix.append(int(num) if num.is_integer() else num)
                number = ''
            if char in '+-*/':
                while stack and stack[-1] in '+-*/' and precedence[char] <= 
precedence[stack[-1]]:
                    postfix.append(stack.pop())
                stack.append(char)
            elif char == '(':
                stack.append(char)
            elif char == ')':
                while stack and stack[-1] != '(':
                    postfix.append(stack.pop())
                stack.pop()

    if number:
        num = float(number)
        postfix.append(int(num) if num.is_integer() else num)
    while stack:
        postfix.append(stack.pop())

    return ' '.join(str(x) for x in postfix)

n = int(input())
for _ in range(n):
    expression = input()
    print(infix_to_postfix(expression))
```

代码运行截图 ==（AC代码截图，至少包含有"Accepted"）==

#44305495提交状态

状态: Accepted

源代码

```python
def infix_to_postfix(expression):
    precedence = {'+':1, '-':1, '*':2, '/':2}
    stack = []
    postfix = []
    number = ''

    for char in expression:
        if char.isnumeric() or char == '.':
            number += char
        else:
            if number:
                num = float(number)
                postfix.append(int(num) if num.is_integer() else num)
                number = ''
            if char in '+-*/':
                while stack and stack[-1] in '+-*/' and precedence[char
                    postfix.append(stack.pop())
                stack.append(char)
            elif char == '(':
                stack.append(char)
            elif char == ')':
                while stack and stack[-1] != '(':
                    postfix.append(stack.pop())
                stack.pop()

    if number:
        num = float(number)
        postfix.append(int(num) if num.is_integer() else num)

    while stack:
        postfix.append(stack.pop())

    return ' '.join(str(x) for x in postfix)

n = int(input())
for _ in range(n):
    expression = input()
    print(infix_to_postfix(expression))
```

22068: 合法出栈序列

http://cs101.openjudge.cn/practice/22068/

思路：

模拟实现，判断该序列是否能反向回到原序列

代码

```python
def check(i=-1, j=-1, length=-1):
    if i > minimum:
        if stack and stack[-1] == standard[i]:
            stack.pop()
            i -= 1
        else:
            try:
                stack.append(test[j])
                j -= 1
            except IndexError:
                pass
    else:
        return "YES"
    if len(stack) == length:
        if length == 0:
            return "YES"
        return "NO"
    return check(i, j, len(stack))

standard = input()
while 1:
    try:
        test = input()
        if len(test) != len(standard) or set(test) != set(standard):
            print("NO")
        else:
            minimum = -len(standard)
            stack = []
            i = -1
            j = -1
            print(check())
    except EOFError:
        break
```

代码运行截图 ==（AC代码截图，至少包含有"Accepted"）==

状态: Accepted

源代码

```python
def check(i=-1, j=-1, length=-1):
    if i > minimum:
        if stack and stack[-1] == standard[i]:
            stack.pop()
            i -= 1
        else:
            try:
                stack.append(test[j])
                j -= 1
            except IndexError:
                pass
    else:
        return "YES"
    if len(stack) == length:
        if length == 0:
            return "YES"
        return "NO"
    return check(i, j, len(stack))

standard = input()
while 1:
    try:
        test = input()
        if len(test) != len(standard) or set(test) != set(standard):
            print("NO")
        else:
            minimum = -len(standard)
            stack = []
            i = -1
            j = -1
            print(check())
    except EOFError:
        break
```

06646: 二叉树的深度

http://cs101.openjudge.cn/practice/06646/

思路：

实现树的数据结构，递归

代码

```python
class Tree:
    structure = {}
    def __init__(self, number, *nodes):
        self.__number = number
        Tree.structure[self.__number] = self
        self.__nodes = nodes
    def depth(self):
        if self.__nodes == ('-1', '-1'):
            return 1
        return 1 + max(Tree.structure[i].depth() for i in self.__nodes if i != '-1')

n = int(input())
for i in range(1, n + 1):
    Tree(str(i), *input().split())

print(Tree.structure['1'].depth())
```

代码运行截图 ==（AC代码截图，至少包含有"Accepted"）==

02299: Ultra-QuickSort

http://cs101.openjudge.cn/practice/02299/

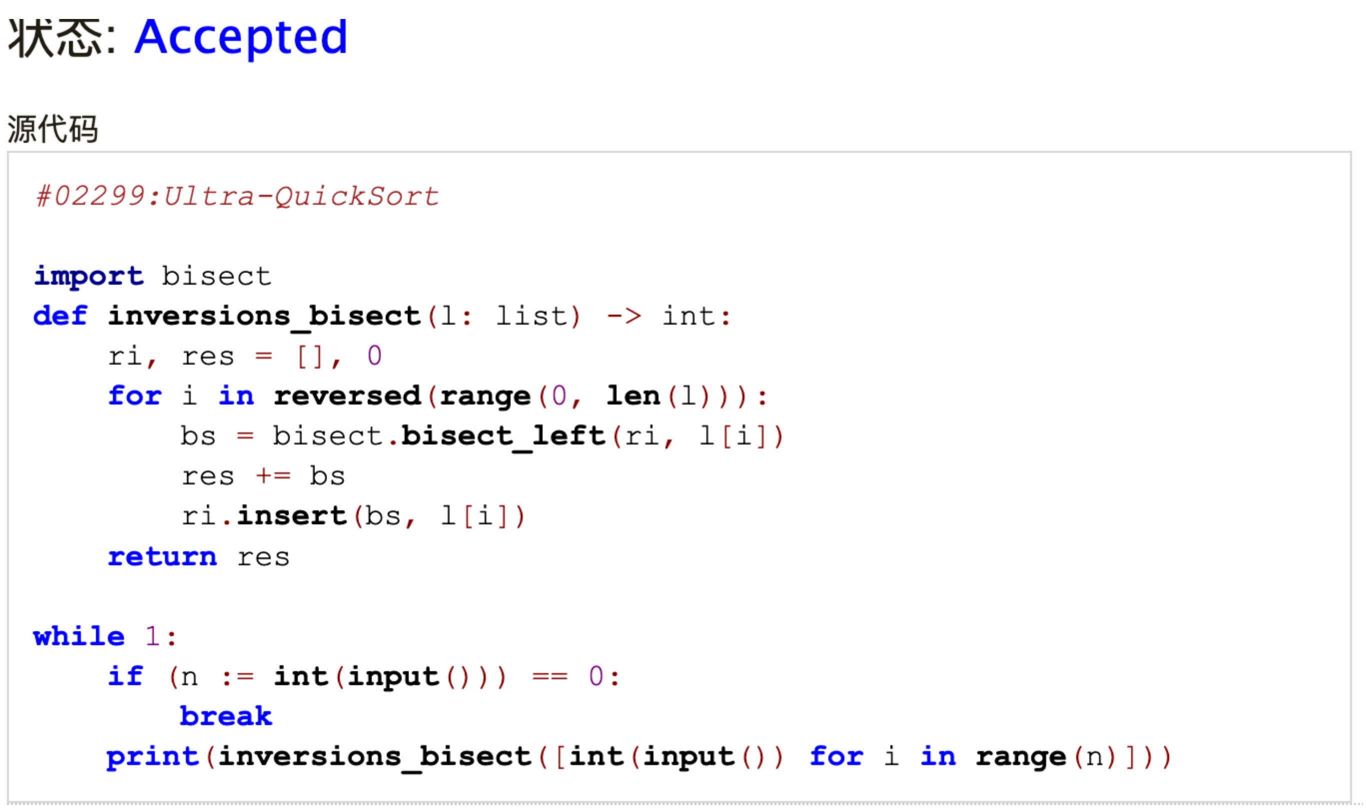
思路：

本质求逆序数，使用二分查找求解，勉强没有超时。

代码

```python
import bisect
def inversions_bisect(l: list) -> int:
    ri, res = [], 0
    for i in reversed(range(0, len(l))):
        bs = bisect.bisect_left(ri, l[i])
        res += bs
        ri.insert(bs, l[i])
    return res

while 1:
    if (n := int(input())) == 0:
        break
    print(inversions_bisect([int(input()) for i in range(n)]))
```

代码运行截图 ==（AC代码截图，至少包含有"Accepted"）==



状态: Accepted

源代码

```python
#02299:Ultra-QuickSort

import bisect
def inversions_bisect(l: list) -> int:
    ri, res = [], 0
    for i in reversed(range(0, len(l))):
        bs = bisect.bisect_left(ri, l[i])
        res += bs
        ri.insert(bs, l[i])
    return res

while 1:
    if (n := int(input())) == 0:
        break
    print(inversions_bisect([int(input()) for i in range(n)]))
```

2. 学习总结和收获

==如果作业题目简单，有否额外练习题目，比如：OJ“2024spring每日选做”、CF、LeetCode、洛谷等网站题目。==

学到了用递归处理树的问题，在中序表达式转后序表达式和合法出栈序列的细节上卡的时间较长。