2024.9.23作业

复习题

- T3

  算法的时间复杂度是一个函数，定性描述该算法的运行时间；而空间复杂度是对一个算法在运行过程中临时占用存储空间大小的量度。

- T4

  算法是把数据从信息中提取出来的手段，并且能通过特定的手段来实现某些特定问题。算法的作用是定义解决问题的具体步骤，以实现计算目标、提高计算效率并优化资源利用。

- T5

  1. 时间复杂度分析

     比如对排序算法，我们会比较各个排序的时间复杂度是O(nlogn)，亦或是O(n^2)

  2. 空间复杂度分析

     对于不同算法我们可能会分析其空间占用，如果开拓了过大的空间可能会对性能有影响。

  3. 最差/最好时间复杂度分析

     在一些情况下我们可能要考虑到算法的稳定性，以及在任何可能出现的特殊情况下的运行性能，我们就要对算法进行最好/最差时间复杂度分析。

- T6

  从时间复杂度和空间复杂度两个维度出发，并且进行综合评判。

- T7

  1. **输入**：算法开始执行前的必要数据或条件。
  2. **输出**：算法执行完毕后的结果或答案。
  3. **有穷性**：算法在有限时间内完成，步骤有限，不陷入无限循环。
  4. **确定性**：相同输入下，算法执行过程和结果唯一确定。
  5. **可行性**：算法在实际应用是否中有效，资源利用合理。

践习题

- T1

```
def judge(num):
    flag = 1
    if(num <= 1):
        return 0
    for i in range(2,num//2+1):
        if num%i == 0:
            flag = 0
    return flag

n = int(input())
x = judge(n)
if x:
    print("是质数")
else:
```

```
    print("不是质数")
```

- T6

```
def selection_sort(arr):
    n = len(arr)
    for i in range(n):
        min_index = i
        for j in range(i + 1, n):
            if arr[j] < arr[min_index]:
                min_index = j
        arr[i], arr[min_index] = arr[min_index], arr[i]
    return arr

array = map(int,input().split())
sorted_array = selection_sort(array)
print("排序后的数组：", sorted_array)
```

5个数字的时候大约几毫秒就能完成，如果数组有20个数字就要花更久时间，大约几十毫秒，一百个数字就会到几百毫秒甚至几秒

- T7

```
def hanoi(n, source, target, auxiliary):
    if n == 1:
        print(f"move {source} to {target}")
    else:
        hanoi(n - 1, source, auxiliary, target).
        print(f"move {source} to {target}")
        hanoi(n - 1, auxiliary, target, source)
n = int(input("请输入正整数 N（环的数量）："))
hanoi(n, 'A', 'B', 'C')
```

我们可以考虑用动态规划存储的方式来优化

- T8

```
class TreeNode:
    def __init__(self, value):
        self.value = value
        self.left = None
        self.right = None

def insert(node, value):
    if node is None:
        return TreeNode(value)
    if value < node.value:
        node.left = insert(node.left, value)
    else:
        node.right = insert(node.right, value)
    return node

def second_visit_traversal(node):
    if node is None:
        return
    second_visit_traversal(node.left)
```

```
    print(node.value, end=' ')
    second_visit_traversal(node.right)

root = None
values = map(int,input().split())
for value in values:
    root = insert(root, value)
print("树排序结果：", end='')
second_visit_traversal(root)
```