复习题

时间复杂度是指算法执行所需时间的量度。

空间复杂度是指算法在执行过程中所需内存空间的量度。

算法是解决特定问题的一组明确的、有限的步骤、规则、方法。

作用：用于描述解决某一类问题的方法步骤，由于方法已然确定，使得我们可以利用计算机来支撑算法从而实现用计算机解决某一类问题。

5、

时间复杂度分析：如排序算法中冒泡排序时间复杂度为O(n^2)，而快速排序时间复杂 度为O(nlogn)，则在运行时间层面上快速排序更优。

空间复杂度分析：如排序算法中冒泡排序空间复杂度为O(1)，而归并排序空间复杂度 为O(n)，则在运行空间占用层面上冒泡排序更优。

最坏情况、最佳情况与平均情况分析：对于冒泡排序的时间复杂度，最坏情况，平均情 况，最佳情况分别为O(n^2)，O(n)，O(n^2)。

6、

对于时间复杂度：先判断输入规模n是什么，再根据算法找出操作数和输入规模n的 代数关系。

对于空间复杂度：先判断输入规模n是什么，再根据算法找出空间需求和输入规模n 的代数关系。

7、

有序性：算法必须在有限的步骤内完成。

明确性：算法的每一步都必须是明确的，每个操作和步骤需要有明确的定义。

输入：算法可有0个或多个输入，是算法运行所需的数据。

输出：算法必须至少生成一个输出，是算法处理后得到的结果。

有效性：算法中的每一步操作都应该是可行的。

见习题

1、

a = int(input("input a number: "))

flag = True

for i in range(2, a):

    if a % i == 0:

        print("not prime")

        flag = False

        break

if flag: print("prime")

6、

import time

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

input\_data = input("请输入以空格分隔的整数数组: ")

arr = list(map(int, input\_data.split()))

print(f"排序前的数组: {arr}")

start\_time = time.time()

sorted\_array = selection\_sort(arr)

end\_time = time.time()

print(f"排序后的数组: {sorted\_array}")

print(f"执行时间: {end\_time - start\_time:.1000000f} 秒")

7、

def hnt(n, A, B, C):

    if n == 1:

        print(f"move {A} to {B}")

        return 0

    hnt(n-1, A, C, B)

    print(f"move {A} to {B}")

    hnt(n-1, C, B, A)

n = int(input("Enter the number of disks:"))

hnt(n, 'A', 'B', 'C')

1. 可以考虑非递归实现，手动使用数据结构（如堆栈）来模拟递归过程，从而避免栈溢出的问题。
2. 若只求所需步数，可以运用记忆化技术，用一个数组来存将k个盘转移到另一个柱子上所需步数从而减少递归数量。

8、

class TreeNode:

    def \_\_init\_\_(self, val=0, left=None, right=None):

        self.val = val

        self.left = left

        self.right = right

def insert\_into\_bst(root, val):

    if not root:

        return TreeNode(val)

    if val < root.val:

        root.left = insert\_into\_bst(root.left, val)

    else:

        root.right = insert\_into\_bst(root.right, val)

    return root

def build\_bst(lst):

    root = None

    for val in lst:

        root = insert\_into\_bst(root, val)

    return root

def left\_first\_traversal(root):

    if root:

        left\_first\_traversal(root.left)

        left\_first\_traversal(root.right)

        print(root.val)

# 获取用户输入并转换为列表

input\_list = list(map(int, input("请输入一个整数列表，以空格分隔: ").split()))

bst\_root = build\_bst(input\_list)

left\_first\_traversal(bst\_root)