复习题：

3. 算法的时间复杂度和空间复杂度分别是什么？

时间复杂度是衡量算法执行时间随输入规模增长而增长的速率的一个概念。空间复杂度是衡量算法执行过程中所需存储空间大小的量度。

4. 算法是什么，有什么作用？

算法是对特定问题求解步骤的一种描述，它是有限序列的指令集，其中每一条指令表示一个或多个操作。

算法的作用有解决特定问题，同时提高效率，通过设计高效的算法，我们可以在更短的时间内解决更大的问题。此外，算法使得计算机能够自动执行一系列任务，计算机可以模拟人类的思维过程，实现自动化和智能化。

5. 算法分析的方法是多种多样的，常用的评判算法效率的方法有哪些？请举例说明。

时间复杂度分析，通过分析算法中的基本操作次数与输入规模之间的关系，可以确定算法的时间复杂度。

实际运行时间测试，通过在特定硬件上运行算法，并测量其执行时间，可以对算法的实际运行效率进行评估。

性能分析进一步考虑了算法在不同输入规模下的性能表现，包括响应时间、吞吐量、资源利用率等指标。通过对这些指标的评估，可以更全面地了解算法的性能特点。

6. 如何去评判一个算法的复杂度？

时间复杂度通常用大O表示法来表示，它只关注算法执行时间随输入规模增长的最主要趋势，忽略低阶项和常数项。通过确定基本操作，估算执行次数，分别考虑最好情况、最坏情况和平均情况下的执行次数，得到时间复杂度的表达式。

7. 算法在一般情况下被认为有五个基本属性，它们分别是什么？请简要说明。

（1）输入

算法应该能够接受输入数据。输入可以是一个或多个值，也可以是没有值（即零个输入）。（2）输出

算法应该能够产生输出结果。输出结果与输入数据有关联，是算法执行完毕后的结果或答案。算法至少应该具有一个输出，以表明其执行的结果。

1. 有穷性

算法应该在有限的步骤内终止，并产生输出结果。换句话说，算法不应该永远运行下去或陷入无限循环。

1. 确定性

算法的每一步操作必须有确定的结果。即无论在相同的输入下运行多少次，算法都应该产生相同的输出。确定性要求算法的每个步骤都是明确无误的，没有歧义。

1. 可行性

算法的每一步操作都应该是可行的，即可以被计算机执行或通过合理的方法手动执行。可行性要求算法中的操作不能超出计算机或执行者的能力范围。

践习题：

1. import math

def is\_prime(a):

    if a <= 1:

        return False

    for i in range(2, int(math.sqrt(a)) + 1):

        if a % i == 0:

            return False

    return True

try:

    a = int(input("请输入一个整数来判断是否为质数: "))

    if is\_prime(a):

        print(f"{a} 是质数")

    else:

        print(f"{a} 不是质数")

except ValueError:

    print("请输入一个有效的整数！")

6. import random

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]

def generate\_random\_array(length, min\_value=0, max\_value=100):

    return [random.randint(min\_value, max\_value) for \_ in range(length)]

def main():

    array\_lengths = [100, 500, 1000, 5000, 10000]

    for length in array\_lengths:

        arr = generate\_random\_array(length)

        print(f"Sorting an array of length {length}...")

        start\_time = time.time()

        selection\_sort(arr)

        end\_time = time.time()

        print(f"Execution time: {end\_time - start\_time:.6f} seconds")

if \_\_name\_\_ == "\_\_main\_\_":

    main()

7. def hanoi(n, source, target, auxiliary):

    if n == 1:

        print(f"Move disk 1 from {source} to {target}")

        return

    hanoi(n-1, source, auxiliary, target)

    print(f"Move disk {n} from {source} to {target}")

    hanoi(n-1, auxiliary, target, source)

n = 3

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

8. class TreeNode:

    def \_\_init\_\_(self, key):

        self.left = None

        self.right = None

        self.val = key

def insert(root, key):

    if root is None:

        return TreeNode(key)

    else:

        if root.val < key:

            root.right = insert(root.right, key)

        else:

            root.left = insert(root.left, key)

    return root

def inorder\_traversal(root, result=None):

    if result is None:

        result = []

    if root:

        inorder\_traversal(root.left, result)

        result.append(root.val)

        inorder\_traversal(root.right, result)

    return result

def tree\_sort(arr):

    if not arr:

        return []

    root = None

    for key in arr:

        root = insert(root, key)

    return inorder\_traversal(root)

arr = [3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5]

sorted\_arr = tree\_sort(arr)

print(sorted\_arr)