复习题

3

**时间复杂度**是衡量算法执行时间随着输入规模增长而增长的速率。它通常表示为输入大小n的函数，并忽略低阶项和常数项，使用大O记法（O-notation）来表示。例如，O(n)表示线性时间复杂度，O(n^2)表示平方时间复杂度。

**空间复杂度**是算法执行过程中所需的最大存储空间量，同样表示为输入大小n的函数。这包括算法本身所占用的空间以及为算法运行而使用的额外空间（如调用栈、临时变量等）。空间复杂度也使用大O记法来表示。

4

**算法**：是一种解决特定问题的一系列明确、有限的步骤。它接受输入，通过一系列计算或操作，产生输出。算法是计算机程序的基础，是解决问题的精确描述。

**作用**：算法通过提供一系列明确的步骤，使得人们能够高效地解决复杂问题。它不仅提高了解决问题的速度和准确性，还使得解决相似问题变得更加容易和系统化。算法在计算机科学、数学、工程学等多个领域都有广泛应用。

5

时间复杂度分析，空间复杂度分析，最好情况，最坏情况，一般情况

6

时间复杂度：运行这个算法所需要的时间长短，可以估算执行次数，或者通过递归式计算

空间复杂度：运行这个算法所需要的空间大小，计算运行时消耗的空间

综合分析两种复杂度可以评判该算法复杂度

7

**有穷性**：算法必须在有限的时间内完成，即算法的执行步骤是有限的。

**确定性**：算法的每一步骤都必须有明确的定义，不能存在歧义或模糊性。

**输入**：算法具有零个或多个输入，这些输入是算法开始执行前的初始条件或已知数据。

**输出**：算法至少有一个输出，它是对输入数据加工处理后的结果。

**可行性**：算法的每一步都必须是可行的，即算法中描述的操作都是可以通过已经实现的基本运算执行有限次来实现的。

践习题

1 a = int(input())

*if* (a == 2):

    print("yes")

    exit()

*elif* (a <= 1):

    print("no")

    exit()

*for* i *in* range(2,a-1):

*if* (a%i == 0):

        print("no")

*break*

print("yes")

6

*import* time

*import* random

def selection\_sort(array):

    length = len(array)

*for* i *in* range(length):

        min\_index = i

*for* j *in* range(i + 1, length):

*if* array[j] < array[min\_index]:

                min\_index = j

        array[i], array[min\_index] = array[min\_index], array[i]

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

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

def measure\_sort\_time(array\_size):

    random\_array = generate\_random\_array(array\_size)

    start\_time = time.time()

    selection\_sort(random\_array)

    end\_time = time.time()

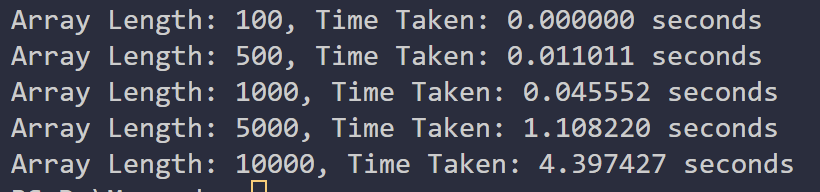
    elapsed\_time = end\_time - start\_time

    print(f"Array Length: {array\_size}, Time Taken: {elapsed\_time:.6f} seconds")

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

*for* size *in* array\_sizes:

    measure\_sort\_time(size)



7

def hnt(n, X, Y, Z):

*if* n == 1:

        print(f"Move disk 1 from {X} to {Y}")

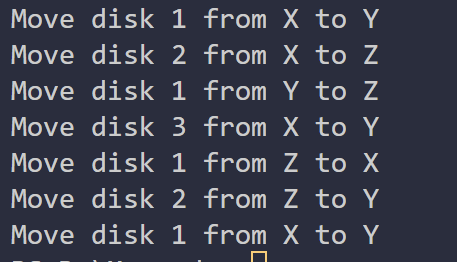
*return* 0

    hnt(n-1, X, Z, Y)

    print(f"Move disk {n} from {X} to {Y}")

    hnt(n-1, Z, Y, X)

hnt(3, 'X', 'Y', 'Z')



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)

input\_list = [7, 3, 18, 10, 22, 8, 11, 26]

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

left\_first\_traversal(bst\_root)

