复习题

三、算法的时间复杂度和空间复杂度分别是什么?

1. 时间复杂度 (Time Complexity)

时间复杂度是指算法在输入规模 n 增加时，执行所需的时间如何增长。它用来衡量算法的效率，描述的是算法的渐近运行时间，即当输入规模变得非常大时，算法的执行时间大致上是怎样的。

2. 空间复杂度 (Space Complexity)

空间复杂度是指算法在运行过程中所需的额外内存空间（不包括输入本身），也用**大O记号**来表示。它衡量的是算法占用的内存空间随输入规模 n 增长的情况。

四、算法是什么，有什么作用?

图灵奖获得主瑞士科学家Niklaus Wirth，Pascal语言的发明者和 结构化程序设计创始者，在1976年出版了《算法 + 数据结构 = 程序设计》，提出了著名的公式： Program = DS + Algorithm • ***算法：解决问题的计算方法（步骤）*** • 数据结构：数据存储的模型 • 语言：描述算法和数据结构的工具 • 程序：用语言描述出来的算法和数据结构

算法是指解决问题的一系列明确的、可执行的步骤或指令。通俗地讲，算法就是为了解决某个问题设计的一个精确流程，用来在有限的时间内得到问题的解答。算法通常由一组有限的操作构成，按照一定的顺序执行，并且每一步都有明确的操作规则。

1. 算法的定义

算法可以被定义为：

- **有穷性**：算法必须在有限的步骤内终止。
- **确定性**：每一步骤的操作必须是明确的，不会产生歧义。
- **可行性**：算法中的每个步骤都能在有限的时间内完成。
- **输入**：算法接收一定数量的输入数据（可能为零或多个）。
- **输出**：算法在执行后会产生至少一个输出结果。

2. 算法的作用

算法的主要作用是解决各类问题并提供效率优化。无论是简单的计算问题还是复杂的系统设计，算法都是核心工具，具体作用如下：

(1) **提高效率**(2) **自动化任务**(3) **解决复杂问题**(4) **优化资源分配**(5) **提高安全性**(6) **解决重复性问题**

五、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。

1. **时间复杂度 (Time Complexity)**
2. **空间复杂度 (Space Complexity)**
3. **最坏、最佳和平均情况分析**
4. **渐近分析 (Asymptotic Analysis)**

六、如何去评判一个算法的复杂度?

统计操作次数：分析算法的每个操作，并估算执行的次数。

使用大O符号：用大O符号表示算法的时间和空间复杂度，忽略常数和低阶项。

分析递归算法：通过递归关系求解时间复杂度，常用递归树或主定理。

最坏、最佳和平均情况：全面分析算法在不同输入条件下的表现。

渐近分析：分析算法在输入规模非常大的时候的表现趋势。

七、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。

算法在一般情况下被认为有五个基本属性，分别是**有穷性**、**确定性**、**可行性**、**输入**和**输出**。这五个属性确保算法可以正确且有效地解决问题。下面是对每个属性的简要说明：

1. 有穷性 (Finiteness)

算法必须在有限的步骤内终止，不能无限执行。

2. 确定性 (Definiteness)

算法的每一个步骤必须是明确的、无歧义的。

3. 可行性 (Feasibility)

算法的每一步操作都应该是可行的

4. 输入 (Input)

一个算法应能够接收零个或多个输入，这些输入是算法执行所需的数据。

5. 输出 (Output)

算法应有一个或多个输出结果，这些结果是对给定输入的正确解答。

践习题

1、

```
def is_prime(n):
    n=int(n)
    if n <= 1:
        return False
    for i in range(2, int(n ** 0.5) + 1):
        if n % i == 0:  # 如果能被整除，说明不是质数
            return False
    return True

a=input("请输入一个数：")
if is_prime(a):
    print(f"{a}是质数")
else :
```

```
    print(f"{a}不是质数")
```

6、

```
PS C:\Users\13803\Desktop\大二作业\数据科学> & C:/Users/13803/AppData/Local/Microsoft/WindowsApps/python3.12.exe c:/Users/13803/Desktop/大二作业/数据科学/代码/2.py
数组长度: 100, 排序时间: 0.00000 秒
数组长度: 1000, 排序时间: 0.04314 秒
数组长度: 5000, 排序时间: 0.68078 秒
数组长度: 10000, 排序时间: 2.89604 秒
数组长度: 20000, 排序时间: 11.14921 秒
```

```python
import random
import time

# 选择排序算法
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
        # 交换最小值和当前位置的值
        arr[i], arr[min_index] = arr[min_index], arr[i]

def test_sorting_time(array_length):
    # 生成随机数组
    arr = [random.randint(0, 10000) for _ in range(array_length)]

    # 记录开始时间
    start_time = time.time()

    # 对数组进行选择排序
    selection_sort(arr)

    # 记录结束时间
    end_time = time.time()

    # 计算并返回执行时间
    return end_time - start_time

# 测试不同长度的数组
array_lengths = [100, 1000, 5000, 10000, 20000]

# 输出不同长度的数组排序所需的时间
for length in array_lengths:
    exec_time = test_sorting_time(length)
    print(f"数组长度: {length}, 排序时间: {exec_time:.5f} 秒")
```

七、

```python
def hanoi(n, source, target, auxiliary):
    if n == 1:
        print(f"将盘子 1 从 {source} 移动到 {target}")
        return
```

```
    # 将 n-1 个盘子从源柱子移动到辅助柱子
    hanoi(n - 1, source, auxiliary, target)
    # 将第 n 个盘子从源柱子移动到目标柱子
    print(f"将盘子 {n} 从 {source} 移动到 {target}")
    # 将 n-1 个盘子从辅助柱子移动到目标柱子
    hanoi(n - 1, auxiliary, target, source)

# 测试汉诺塔函数
num_disks = 3  # 盘子的数量
hanoi(num_disks, 'A', 'C', 'B')
```

八、

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

class BinarySearchTree:
    def __init__(self):
        self.root = None

    def insert(self, key):
        if self.root is None:
            self.root = TreeNode(key)
        else:
            self._insert_recursively(self.root, key)

    def _insert_recursively(self, node, key):
        if key < node.val:
            if node.left is None:
                node.left = TreeNode(key)
            else:
                self._insert_recursively(node.left, key)
        else:
            if node.right is None:
                node.right = TreeNode(key)
            else:
                self._insert_recursively(node.right, key)

    def inorder_traversal(self, node, sorted_list):
        if node:
            self.inorder_traversal(node.left, sorted_list)
            sorted_list.append(node.val)
            self.inorder_traversal(node.right, sorted_list)

def tree_sort(arr):
    bst = BinarySearchTree()
    for value in arr:
        bst.insert(value)

    sorted_list = []
    bst.inorder_traversal(bst.root, sorted_list)
```

```
    return sorted_list

# 测试树排序
if __name__ == "__main__":
    data = [5, 3, 8, 1, 4, 7, 10]
    sorted_data = tree_sort(data)
    print("排序后的数据:", sorted_data)
```