复习题

这些问题都是关于算法分析的基础概念，下面是每个问题的答案：

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**：指的是算法执行所需要的计算工作量，通常用大O表示法来描述，比如O(n)、O(n^2)、O(log n)等。它描述了算法执行时间随输入规模增长的变化趋势。
   - **空间复杂度**：指的是算法执行过程中所需的存储空间，同样可以用大O表示法来描述，如O(1)、O(n)、O(n^2)等。它描述了算法执行过程中占用内存空间随输入规模增长的变化趋势。

   4.什么是算法：

- 算法是对可机械执行的一系列步骤精准而明确的规范"

  5.评价算法的效率方法
- 通过计算空间复杂度和时间复杂度

6. **如何去评判一个算法的复杂度？**
   - 评判算法复杂度通常涉及分析算法的时间复杂度和空间复杂度。可以通过分析算法中基本操作（如比较、交换、计算等）的执行次数来确定时间复杂度。空间复杂度则通过分析算法执行过程中所需的额外存储空间来确定。
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。**
   - **有穷性**：算法必须在有限的步骤后终止。
   - **确定性**：算法的每一步操作必须是明确的，无歧义的。
   - **可行性**：算法的每一步操作都必须是可执行的，即在当前的技术和资源条件下能够实现。
   - **指定输入**：算法可以有零个或多个输入，这些输入是算法执行所需的初始数据。
   - **指定输出**：算法至少有一个输出，输出是算法执行的结果。
8. **算法分析常用的理论方法有哪些？**
   - **渐进分析**：通过大O表示法来分析算法在最坏、平均或最好情况下的时间复杂度。
   - **摊还分析**：用于分析那些平均性能比最坏性能要好的算法。
   - **决策树模型**：通过构建决策树来分析算法的时间复杂度。
   - **概率分析**：当算法的性能依赖于随机事件时，使用概率分析来估计算法的平均性能。
   - **实验分析**：通过实际运行算法并测量其性能来分析算法。

实践题

1.

直接暴力出结果了，就判断一个数，不需要用筛法，如果是判断很多数是不是质数
可以用线性筛最大程度降低时间复杂度

```
def is_prime(n):
if n < 2:
return False
for i in range(2, n):
if n % i == 0:
return False
return True

n = int(input())
print(is_prime(n))
```

6.

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

n = int(input())
arr = list(map(int,input().split()))
print(select_sort(arr))
```

7.

```
def hanoi(n, a, b, c):
if n == 1:
print(a, '-->', c)
return
else:
hanoi(n-1, a, c, b)
print(a, '-->', c)
hanoi(n-1, b, a, c)


n=int(input())
hanoi(n, 'A', 'B', 'C')
```

8.

树排序其实就是是基于二叉搜索树的排序方法。它通过先构建一棵二叉搜索树，然后进行中序遍历来得到有序序列。树排序的关键在于维护树的结构，确保每个节点的键值都大于其左子树中所有节点的键值，且小于其右子树中所有节点的键值

具体解释放在注释里面

```
class node:
    def __init__(self, key):
        self.left = None  # 左子节点
        self.right = None  # 右子节点
        self.val = key     # 节点存储的值

def add(root, key):
    if root is None:  # 如果当前节点为空，创建一个新的节点
        return node(key)
    if root.val < key:  # 如果当前节点的值小于key，递归地在右子树中插入key
        root.right = add(root.right, key)
    else:  # 否则，在左子树中插入key
        root.left = add(root.left, key)
    return root  # 返回当前节点

def find(root, arr):
    if root:  # 如果当前节点不为空
        find(root.left, arr)  # 先递归遍历左子树
        arr.append(root.val)  # 访问当前节点，将其值添加到数组中
        find(root.right, arr)  # 然后递归遍历右子树

def sort(arr):
    if not arr:  # 如果数组为空，直接返回空数组
        return []
    root = None  # 初始化树的根节点为None
    for key in arr:  # 遍历数组中的每个元素
        root = add(root, key)  # 将元素插入到树中
    sorted_arr = []  # 初始化一个空数组来存储排序结果
    find(root, sorted_arr)  # 中序遍历树，将节点值添加到数组中
    return sorted_arr  # 返回排序后的数组

# 输入一个数组
arr = list(map(int, input().split()))  # 从用户输入获取一个整数数组
sorted_arr = sort(arr)  # 使用sort函数对数组进行排序
print(sorted_arr)  # 打印排序后的数组
```