问题 1

请证明基于比较的排序算法计算复杂度下界为$\Omega(nlogn)$

对于一个比较排序算法，其决策过程等价于一个二叉决策树，对于 n 个元素，决策树至少需要 n!个叶子节点来覆盖所有可能的输入排序结果。

因为决策树的高度 h 代表最坏情况下的比较次数: $2^h \geqslant n!$ ⇒ $h \geqslant \log(n!)$

对 log(n!)使用 Stirling 近似公式展开: $\log(n!)=\Theta(n\log n)$

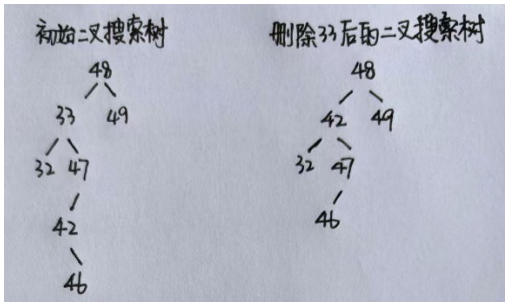
所以最坏情况下，任何基于比较的排序算法的时间复杂度是$\Omega(n\log n)$

问题 2

构建二叉搜索树

> 1. 给定一个数组 [48, 33, 49, 47, 42, 46, 32], 构建一个二叉搜索树存储这些数据，请**绘制**所构建的**二叉搜索树**（标明结点的值）。
> 2. 从第1问的二叉搜索树中删除33，请**绘制**删除33后的**二叉搜索树**（标明结点的值）

- 推荐使用graphviz包绘制树的结构



问题 3

> 下面是九门课程的开始和结束时间:
>
> [(9:00,12:30), (11:00,14:00), (13:00, 14:30), (9:00,10:30),(13:00, 14:30),(14:00,16:30), (15:00,16:30), (15:00,16:30), (9:00,10:30)]
>
> 请使用贪婪算法为这九门课分配教室，要求在同一天内安排这些课，并且每个教室同一时间只能安排一门课。

请问最少需要几间教室，罗列出每个教室安排的课程

最少需要 3 间教室

教室 1:

09:00 - 12:30

13:00 - 14:30

15:00 - 16:30

教室 2:

09:00 - 10:30

11:00 - 14:00

14:00 - 16:30

教室 3:

09:00 - 10:30

13:00 - 14:30

15:00 - 16:30

问题 4

> 爬楼梯问题：假设爬楼梯时你每次只能爬一阶或者爬两阶，问爬上n阶的楼梯，你一共有多少种方法

请设计算法完成该问题，分析算法设计思路，计算时间复杂度，并基于python编程实现

算法伪代码：

```
def climb_stairs(n):
    if n == 0:
        return 1
    if n == 1:
        return 1
    dp = [0] * (n + 1)
    dp[0], dp[1] = 1, 1
    for i in range(2, n + 1):
        dp[i] = dp[i - 1] + dp[i - 2]
    return dp[n]
```

计算时间复杂度：O(n)

初始化 dp[0] 和 dp[1]：O(1)，通过循环计算 dp[2] 到 dp[n]：每个 dp[i] 的计算需要一次加法操作（dp[i-1] + dp[i-2]），时间为 O(1)。总循环次数 n-1 次（从 i=2 到 i=n）。

∴总时间复杂度为：T(n)=O(1) (初始化)+(n−1)×O(1) =O(n)

问题 5

> 0-1背包问题：现在有4块大理石，每个大理石对应的重量和价值使用一个元组表示, 即（重量，价值），4块大理石的重量和价值为：[(5,10), (4,40), (6,30), (3, 50)], 假设你有一辆最大承重为9的小推车，请问使用这个小推车装大理石的最大价值为多少

请设计一个算法解决该问题，分析算法设计思路，计算时间复杂度，并基于python编程实现

定义一个一维数组 dp[w]表示容量为 w 时的最大价值，状态转移方程为：

$$dp[w]=\max\left(dp[w],dp[w-weight[i]+value[i]]\right)$$

算法伪代码：

```
def knapsack_01(weights, values, capacity):
    n = len(weights)
    dp = [0] * (capacity + 1)
    for i in range(n):
        for w in range(capacity, weights[i] - 1, -1):
            dp[w] = max(dp[w], dp[w - weights[i]] + values[i])
    return dp[capacity]
```

时间复杂度：O(nw)=O(4*9)=O(36)

问题 6

给定 `n` 个非负整数表示每个宽度为 `1` 的柱子的高度图，计算按此排列的柱子，下雨之后能接多少雨水。

示例 1：

> **输入：** height = [0,1,0,2,1,0,1,3,2,1,2,1]
> **输出：** 6
> **解释：** 上面是由数组 [0,1,0,2,1,0,1,3,2,1,2,1] 表示的高度图，在这种情况下，可以接 6 个单位的雨水（蓝色部分表示雨水）。

示例 2：

> **输入：** height = [4,2,0,3,2,5]
> **输出：** 9

请分析算法设计思路，计算时间复杂度，并基于python编程实现

每个位置能接的水量由该位置左边的最高柱子和右边的最高柱子决定，状态方程表示为：

$$water[i] = \min\left(left\_\max[i], right\_\max[i] - height[i]\right)$$

算法伪代码如下：

```
def trap(height):
    if not height:
        return 0
    left, right = 0, len(height) - 1
    left_max, right_max = height[left], height[right]
    res = 0
    while left < right:
        if height[left] < height[right]:
            left += 1
            left_max = max(left_max, height[left])
            res += max(0, left_max - height[left])
        else:
            right -= 1
            right_max = max(right_max, height[right])
            res += max(0, right_max - height[right])
    return res
```

时间复杂度：每个元素最多被访问一次（left 和 right 各扫描一遍）

问题 7

股票投资组合优化：假设你是一位投资者，想要在不同的股票中分配你的资金，以最大化你的投资回报。每只股票都有不同的预期收益率和风险。你的目标是选择一些股票，使得总投资金额不超过你的预算，并且预期收益最大化。

在这个情况下，你可以将每只股票视为一个“物品”，其重量为投资金额，价值为预期收益率。然后，你可以使用分级背包问题的方法来选择部分股票，以便在预算内获得最大的预期收益。

以下是一个简化的例子：

假设你有以下三只股票可供选择：

1. 股票 A：投资金额 5000 美元，预期收益率 10%
2. 股票 B：投资金额 3000 美元，预期收益率 8%
3. 股票 C：投资金额 2000 美元，预期收益率 12%

请设计算法找到最优投资方案，分析算法设计思路，计算时间复杂度，并基于python编程实现

dp[i][j]:前 i 只股票,在总投资不超过 j 时能获得的最大收益，状态方程表示为：

$$dp[i][j] = \max\left(dp[i-1][j], dp[i-1][j-\cos t[i]] + profit[i]\right)$$

若 j < cost[i]，不能选择第 i 支股票。若能选，则取：选与不选两者最大值

算法伪代码如下：

```
def investment_optimization(costs, returns, capital):
    n = len(costs)
    dp = [0] * (capital + 1)
    path = [[False] * (capital + 1) for _ in range(n)]
    for i in range(n):
        for j in range(capital, costs[i] - 1, -1):
            if dp[j - costs[i]] + returns[i] > dp[j]:
                dp[j] = dp[j - costs[i]] + returns[i]
                path[i][j] = True
    selected = []
    j = capital
    for i in range(n - 1, -1, -1):
        if path[i][j]:
            selected.append(i)
            j -= costs[i]
    return dp[capital], selected[::-1]
```

时间复杂度：O(nW)，其中 n 是股票数量，W 是最大投资金额

问题 8

给你二叉搜索树的根节点 root ，该树中的恰好两个节点的值被错误地交换。请在不改变其结构的情况下，恢复这棵树 。设计算法解决该问题，分析算法设计思路，计算时间复杂度，并基于python编程实现

对二叉搜索树进行中序遍历，记录中间值对（判断是否逆序）。在中序序列中定位出：第一个异常点 x（比后一个大）第二个异常点 y（比前一个小），交换 x 和 y 的值即可恢复。

算法伪代码如下：

```
def recoverTree(root):
    x = y = prev = None
    def inorder(node):
        nonlocal x, y, prev
        if not node:
            return
        inorder(node.left)
```

```
        if prev and node.val < prev.val:
            y = node
            if not x:
                x = prev
            else:
                return
        prev = node
        inorder(node.right)
    inorder(root)
    if x and y:
        x.val, y.val = y.val, x.val
```

时间复杂度：由于需要完整中序遍历一遍树，故时间复杂度为 O(n)

问题 9

给你一个二叉树的根节点 root ，树中每个节点都存放有一个 0 到 9 之间的数字。每条从根节点到叶节点的路径都代表一个数字：

> 例如，从根节点到叶节点的路径 1 -> 2 -> 3 表示数字 123 。

设计一个算法计算从根节点到叶节点生成的所有数字之和 ，分析算法设计思路，计算时间复杂度，并基于python编程实现

叶节点:是指没有子节点的节点。

使用深度优先搜索进行递归

在每一步

维护一个当前路径上的数字 current_number，每访问一个节点就更新为：

$$current\_number = current\_number * 10 + node.val$$

当走到叶节点时，把当前数字加入总和。

算法伪代码如下：

```
def dfs(node, current_number):
    if node is None:
        return 0
    current_number = current_number * 10 + node.val
    if node is leaf:
        return current_number
    return dfs(left) + dfs(right)
```

时间复杂度：由于访问每个节点一次，故时间复杂度为 O(n)。

问题 10

给你一个二叉树的根节点 root ， 检查它是否轴对称。

1. 分析算法设计思路，计算时间复杂度，并基于python编程实现
2. * 设计使用递归和迭代两种方法解决这个问题，计算时间复杂度，并基于python编程实现

比较**左子树**和**右子树**是否为“镜像树”，对称要求是，左子树的左边=右子树的右边，左子树的右边=右子树的左边。

递归实现算法伪代码，时间复杂度是 O(n)：

```
def is_mirror(t1, t2):
    if t1 is None and t2 is None: return True
    if t1 is None or t2 is None: return False
    return (t1.val == t2.val and is_mirror(t1.left, t2.right) and is_mirror(t1.right, t2.left))
```

问题 11

给你一个由 '1'（陆地）和 '0'（水）组成的的二维网格，请你计算网格中岛屿的数量。岛屿总是被水包围，并且每座岛屿只能由水平方向和/或竖直方向上相邻的陆地连接形成。

此外，你可以假设该网格的四条边均被水包围。

分析算法设计思路，计算时间复杂度，并基于python编程实现

要在二维数组中，统计连通的 '1' 区域的数量。

每当发现一个 '1'：

把它“淹掉”变成 '0'（表示访问过）

同时用深度优先遍历把所有连通的 '1' 都变成 '0'

岛屿数加一

算法伪代码如下：

```
def numIslands(grid):
    if not grid:
        return 0
    rows, cols = len(grid), len(grid[0])
    def dfs(r, c):
        if r < 0 or r >= rows or c < 0 or c >= cols or grid[r][c] == '0':
            return
        grid[r][c] = '0'   # 标记为访问过
        dfs(r + 1, c)
        dfs(r - 1, c)
        dfs(r, c + 1)
        dfs(r, c - 1)
    count = 0
    for r in range(rows):
        for c in range(cols):
            if grid[r][c] == '1':
                dfs(r, c)
                count += 1
    return count
```

时间复杂度为：二维网格的长为 m，宽为 n，每个格子最多被访问一次，故时间复杂度为 O(mn)。

注：本作业在完成时使用了 chatgpt