

数据结构与算法

作业 2: 复杂度分析

您的姓名 您的学号

November 13, 2025

问题一 循环结构分析

请分析以下 Python 代码片段的时间复杂度，并使用大 O 表示法给出结果。请简要说明你的推导过程。

```
1 def exercise_one(n):
2     total = 0
3     # 外层循环
4     for i in range(n):
5         # 内层循环
6         for j in range(n * n):
7             total += i * j
8     return total
```

解:

问题二 连续与条件结构分析 (规则 3 & 4)

请分析以下 Python 函数的整体时间复杂度。请分别指出代码中第一部分和第二部分 (**if/else** 结构) 的复杂度，并根据规则推导出最终的整体复杂度。

```
1 def exercise_two(n):
2     # 第一部分
3     result = 0
4     for i in range(n // 2):
5         result += i
6
7     # 第二部分
```

```
8     if n > 100:
9         # if 分支
10        for i in range(n):
11            for j in range(n):
12                result += (i + j)
13    else:
14        # else 分支
15        result += n
16
17    return result
```

解:

问题三 对数复杂度分析

与课程中分析的循环不同, 有些循环的计数器 (counter) 并不是线性增加的。请分析以下代码片段的时间复杂度, 并解释为什么它的增长趋势与普通循环不同。

```
1  def exercise_three(n):
2      i = 1
3      count = 0
4      while i < n:
5          i = i * 2  # 关键步骤
6          count += 1
7      return count
```

解:

问题四 主定理直接应用 (规则 5)

一个递归算法的运行时间由以下递归关系式描述:

$$T(n) = 8T\left(\frac{n}{2}\right) + O(n^2) \quad (1)$$

请根据主定理 (Master Theorem) 判断该算法的时间复杂度属于哪一种情况, 并写出最终的复杂度结果。

解:

问题五 从代码到递归关系式 (规则 5)

请分析以下递归函数 `recursive_func` 的时间复杂度。请先写出描述其运行时间的递归关系式, 然后使用主定理求解。

```
1 def recursive_func(n):
2     if n <= 1:
3         return 1
4
5     # 递归调用
6     result = recursive_func(n / 3) + recursive_func(n / 3)
7         ↪ + recursive_func(n / 3)
8
9     # 非递归部分的开销
10    for i in range(n):
11        result += i
12
13    return result
```

解:

问题六 算法比较

假设有两个算法 A 和 B 用于解决同一个问题。算法 A 的时间复杂度为 $O(n^2)$, 算法 B 的时间复杂度为 $O(n \log n)$ 。当处理一个规模为 $n = 1,000,000$ 的超大规模数据集时, 你会选择哪个算法? 为什么? 这种选择体现了渐进分析的什么核心思想?

解:

问题七 综合分析

请综合运用时间复杂度分析的各项规则, 分析以下 `comprehensive_analysis` 函数的整体时间复杂度, 并给出详细的推导过程。

```
1 def comprehensive_analysis(n):
2     # 第一部分
```

```
3     for i in range(n):
4         for j in range(i):
5             # O(1) 操作
6             pass
7
8     # 第二部分
9     if n % 3 == 0:
10        # 递归调用
11        def recursive_part(x):
12            if x <= 1:
13                return 1
14            return 4 * recursive_part(x / 2)
15        recursive_part(n)
16    else:
17        # 简单循环
18        for i in range(n):
19            # O(1) 操作
20            pass
21
22    return n
```

解:

问题八 扩展: 反向思考

请设计一个 Python 函数, 使其时间复杂度恰好为 $O(n\sqrt{n})$ 。提示: 可以考虑使用嵌套循环, 其中内层循环的迭代次数与外层循环的变量 i 相关。

解: