真题及解析（个人+AI辅助）

算法分析试卷（真题汇总）

1） 动态规划算法包括所有具有如下特征的算法：首先将原问题分成更小的子问题，保存这些子问题的解，并由它们来计算原问题的一个解。下列的问题求解中什么不能使用动态规划算法？ （ ）

A．最长公共子序列问题 B．图像无损压缩问题

C．0–1背包问题 D．二分搜索问题

动态规划算法的核心在于分解问题为重叠子问题，并通过存储子问题的解（避免重复计算）来高效求解原问题。以下是对各选项的分析：

A．最长公共子序列问题（LCS）

适用动态规划：可将问题分解为两个字符串的前缀子问题（如 \(\text{LCS}(i, j)\) 表示字符串 \(A[1..i]\) 和 \(B[1..j]\) 的最长公共子序列长度）。子问题存在重叠性（如不同路径可能多次计算相同的 \(\text{LCS}(i, j)\)），且可通过填表法存储中间结果，避免重复计算。结论：适用动态规划。

B．图像无损压缩问题

可能适用动态规划：图像压缩（如霍夫曼编码、行程长度编码等）虽常基于贪心或统计方法，但部分场景（如预测编码、分块压缩）可通过动态规划优化。例如，在分块压缩中，选择最优分块方式或预测模式时，可分解为子块的最优解问题，并存储中间结果。结论：存在适用动态规划的可能性（具体取决于压缩算法设计）。

C．0–1 背包问题

典型动态规划问题：问题可分解为“前 i 个物品、容量为 j 时的最大价值”子问题（状态 \(dp[i][j]\)）。子问题重叠且需避免重复计算，通过填表法可高效求解。结论：适用动态规划。

D．二分搜索问题

不适用动态规划：二分搜索的核心是分治策略（每次将问题规模减半），但子问题无重叠性（每次搜索仅处理一个子区间，无需存储中间结果）。动态规划的“存储子问题解”特性在此完全用不上，直接迭代或递归即可完成搜索。结论：不适用动态规划。

最终答案

D．二分搜索问题

（2）程序块（ ）是回溯法中遍历排列树的算法框架程序。

A.
```
void backtrack (int t)
{
  if (t>n) output(x);
    else
      for (int i=t;i<=n;i++) {
        swap(x[t], x[i]);
        if (legal(t)) backtrack(t+1);
        swap(x[t], x[i]);
      }
}
```

B.
```
void backtrack (int t)
{
  if (t>n) output(x);
    else
      for (int i=0;i<=1;i++) {
        x[t]=i;
        if (legal(t)) backtrack(t+1);
      }
}
```

C.
```
void backtrack (int t)
{
  if (t>n) output(x);
    else
      for (int i=0;i<=1;i++) {
        x[t]=i;
        if (legal(t)) backtrack(t-1);
      }
}
```

D.
```
void backtrack (int t)
{
  if (t>n) output(x);
    else
      for (int i=t;i<=n;i++) {
        swap(x[t], x[i]);
        if (legal(t)) backtrack(t+1);
      }
}
```

题目问的是“回溯法中遍历排列树的算法框架程序”。

我们重点分析排列树的特点：

排列树是用于生成全排列的回溯树，每个位置只能选还未使用的元素；

常用的方法是通过交换元素的位置来逐步构建排列；

树的深度是元素个数 n，每次递归向下走一层，表示固定当前位置；

所以核心框架应包括：交换元素、递归回溯、恢复现场（swap回来）。

故选择A

（3）用数量级形式表示的算法执行时间称为算法的：（ ）

A．时间复杂度 B．空间复杂度 C．处理器复杂度 D．通信复杂度

A

（4）n 个人拎着水桶在一个水龙头前面排队打水，水桶有大有小，水桶必须打满水，水流恒定。如下（ ）说法不正确?

A．让水桶大的人先打水，可以使得每个人排队时间之和最小。

B．让水桶小的人先打水，可以使得每个人排队时间之和最小。

C．让水桶小的人先打水，在某个确定的时间 t 内，可以让尽可能多的人打上水。

D．若要在尽可能短的时间内，n 个人都打完水，按照什么顺序其实都一样。

答案：A

解析：设 n 个人的水桶容量为 $t_1, t_2, \dots, t_n$（升序排列，$t_1 \leq t_2 \leq \dots \leq t_n$），排队时间为每个人等待前面所有人打水的时间之和。

选项分析：

A．错误：若让水桶大的人先打（降序排列），总排队时间为 $T_{\text{降序}} = (n-1)t_n + (n-2)t_{n-1} + \dots + t_2$，显然大于升序排列的总时间。

B．正确：让水桶小的人先打（升序排列），总排队时间为 $T_{\text{升序}} = (n-1)t_1 + (n-2)t_2 + \dots + t_{n-1}$，这是最优排列（数学上可证明升序总时间最小）。

C．正确：前 k 个人打水的总时间为 $t_1 + t_2 + \dots + t_k$。升序排列时，前 k 项和最小，因此在确定时间 t 内，能完成打水的人数最多。

D．正确：所有人都打完水的总时间为 $t_1 + t_2 + \dots + t_n$，与打水顺序无关（加法交换律）。

（5） 旅行商问题的解可表示成解空间树，此解空间的状态空间有（ ） 个结点，此解空间树被称为（ ）。

A． nn B． n! C． 2n D． n E．排列树 F．子集树

答案：B、E

解析：

解空间状态数：旅行商问题（TSP）要求遍历 n 个城市且不重复，解的本质是 n 个城市的一个排列，因此解空间树的结点数为 $n!$（排列数）。

解空间树类型：排列问题的解空间树称为排列树（每个结点表示一个部分排列，叶子结点为完整排列）。

对比子集树：若问题解为子集（如背包问题选或不选物品），解空间树为子集树（结点数为 2^n），与 TSP 无关。

（6）解决问题时间的复杂性为多项式界的有：（ ）

A．快速排序算法 B．n–后问题 C．单源最短路径问题 D．骑士巡游问题

答案：A、C

解析：

快速排序算法：平均时间复杂度为 $O(n \log n)$，属于多项式时间算法。

n – 后问题：采用回溯法求解，时间复杂度为 $O(n!)$，属于指数级（非多项式）。

单源最短路径问题（如 Dijkstra 算法）：时间复杂度为 \(O(n^2)\) 或优化后的 \(O(m + n \log n)\)（m 为边数），均为多项式级。

骑士巡游问题：回溯法求解，时间复杂度高（非多项式）。

（7） 以下说法正确的是： （CD ）

A. 贪心法通过分阶段地挑选最优解，对所有问题都能很快获得问题的最优解。

B. 一个问题是否适合用动态规划算法要看它是否具有重叠子问题。

C. 分治法通过把问题化为较小的问题来解决原问题，从而简化或降低了原问题的复杂程度。

D. 回溯法是一种深度优先搜索算法。

- **A. 错误**：贪心法仅在满足**贪心选择性质**时才能得到最优解，并非对所有问题有效（如 0–1 背包问题贪心无法得到最优解）。
- **B. 错误**：动态规划要求问题具有**最优子结构**和**重叠子问题**，仅重叠子问题不充分。

（8）具有最优子结构的算法有：（ ）

A. 贪心算法 B. 回溯法 C. 分支限界法 D. 动态规划法

AD

（9） 以下说法错误的是： （ ）

A. 数值概率算法总能求解得到问题的一个解，而且所求得的解总是正确的。

B. 舍伍德算法不是避免算法的最坏情况，而是以较大的概率消除最坏情形。

C. 蒙特卡罗算法可以求得问题的一个解，但该解未必正确。

D. 拉斯维加斯算法有时以一定概率给出错误答案。

- **A. 错误**：数值概率算法（如蒙特卡罗）可能返回错误解，需结合误差控制（如重复计算）。
- **B. 正确**：舍伍德算法通过随机化消除最坏情况的**确定性**，使最坏情况以低概率出现。
- **C. 正确**：蒙特卡罗算法可能返回错误解，正确概率由算法设计决定。
- **D. 错误**：拉斯维加斯算法**不会给出错误答案**，但可能以一定概率返回“无解”（需重新运行）。

（10）适于递归实现的算法有：（ ）

A. 随机化算法 B. 近似算法 C. 分治法 D. 回溯法

CD

（11）分治法的适用条件是，所解决的问题一般具有这些特征：（ ）

A. 该问题的规模缩小到一定的程度就可以容易地解决；

B. 该问题不可以分解为若干个规模较小的相同问题；

C. 利用该问题分解出的子问题的解可以合并为该问题的解

D. 该问题所分解出的各个子问题不是相互独立的。

- **A. 正确**：问题规模缩小到基例（如 n=1）时可直接求解。
- **B. 错误**：分治法要求问题**可分解为若干规模较小的相同子问题**（如排序分解为子数组排序）。
- **C. 正确**：子问题的解需能**合并**为原问题的解（如归并排序的合并步骤）。
- **D. 错误**：分治法要求子问题**相互独立**（无重叠子问题），否则需用动态规划。

（12）下面那条规则是正确的？（ ）

A. $\{f(n) = O(F(n)), g(n) = O(G(n))\} \Rightarrow f(n)/g(n) = O(F(n)/G(n))$

B. $\{f(n) = O(F(n)), g(n) = O(G(n))\} \Rightarrow f(n)/g(n) = \Theta(F(n)/G(n))$

C. $\{f(n) = \Omega(F(n)), g(n) = \Omega(G(n))\} \Rightarrow f(n)/g(n) = \Omega(F(n)/G(n))$

D. $\{f(n) = \Theta(F(n)), g(n) = \Theta(G(n))\} \Rightarrow f(n)/g(n) = \Theta(F(n)/G(n))$

D，不确定

二．算法设计分析题（61 分）

1、（12 分）用 O、Ω、Θ 表示函数 f 与 g 之间的关系：

(1) f(n)=50　　g(n)= 3n

(2) $f(n)=6n+10n\lfloor \log n \rfloor$　　$g(n)= 2\sqrt{n}$

(3) f(n)= n/logn-1　　$g(n)= \log_3 n$

(4) $f(n)=n^2$　　$g(n)=3^n$

f(n) = O(g(n))：f(n) 的增长率小于或等于 g(n) 的增长率（f 是 g 的上界）。

f(n) = Ω(g(n))：f(n) 的增长率大于或等于 g(n) 的增长率（f 是 g 的下界）。

f(n) = Θ(g(n))：f(n) 的增长率等于 g(n) 的增长率（f 和 g 同阶）。

---

(1) f(n) = 50, g(n) = 3n

关系: `f(n) = O(g(n))`

解释: f(n) 是一个常数函数，其值不随 n 的增长而改变。g(n) 是一个线性函数，随 n 的增长而线性增长。因为常数函数的增长率远小于线性函数，所以 f(n) 的增长率小于 g(n)。因此，f(n) 是 g(n) 的一个渐进上界。

---

(2) $f(n) = 6n + 10n\lfloor \log n \rfloor$, $g(n) = 2\sqrt{n}$

关系: `f(n) = Ω(g(n))`

解释:

对于函数 f(n)，当 n 很大时，`10n⌊log n⌋` 这一项是主导项，它的增长率决定了整个函数的增长率。所以 f(n) 的增长率可以近似看作 `n log n` 。

函数 g(n) 的增长率为 `√n` 。

比较 `n log n` 和 `√n` ，`n log n` 的增长速度远快于 `√n` 。

因此，f(n) 的增长率大于 g(n)，f(n) 是 g(n) 的一个渐进下界。

---

(3) $f(n) = n/\log n - 1$, $g(n) = \log_3 n$

关系: `f(n) = Ω(g(n))`

解释:

根据对数换底公式，`g(n) = log₃ n = log n / log 3` 。由于 `log 3` 是一个常数，所以 g(n) 的增长率与 `log n` 相同。

函数 f(n) 的增长率为 `n/log n` 。

比较 `n/log n` 和 `log n` 。任何形式为 `n/polylog(n)` 的函数都比任何 `polylog(n)` 函数增长得快。显然，`n/log n` 的增长速度远快于 `log n` 。

因此，f(n) 的增长率大于 g(n)，f(n) 是 g(n) 的一个渐进下界。

---

(4) $f(n) = n^2$, $g(n) = 3^n$

关系: `f(n) = O(g(n))`

解释: f(n) 是一个多项式函数（二次方），而 g(n) 是一个指数函数。指数函数 (`aⁿ` , a>1) 的增长速度要远快于任何多项式函数 (`n^k`)。因此，f(n) 的增长率远小于 g(n)。所以，f(n) 是 g(n) 的一个渐进上界。

2、 (6 分）代码填空题：

Hanoi(n,a,b,c)

if （n==1） move(a,c)；

else

{ _________________1；

_________________2；

Hanoi(n–1,b, a, c);

}

1. `Hanoi(n-1, a, c, b)`
2. `move(a, c)`

3、（8 分）一个机器人每步可以走 1 米、2 米或 3 米。要计算机器人走 n 米，有多少种走法。写出相应的递归求解方程即可。

答案:

设 `f(n)` 为走 `n` 米的走法总数，则递归方程为：$f(n) = \begin{cases} 1, & \text{if } n = 0 \\ 0, & \text{if } n < 0 \\ f(n-1) + f(n-2) + f(n-3), & \text{if } n \geq 1 \end{cases}$

解析:

- 边界条件：`n=0` 时只有一种走法（不走）；`n<0` 时无解。
- 递归式：最后一步可能走 **1** 米、**2** 米或 **3** 米，因此总走法为三种情况之和。

4、(13 分)在黑板上写了 N 个正整数作成的一个数列，进行如下操作:每一次擦去其中的两个数 a 和 b，然后在数列中加入一个数 ax b+1， 如此下去直至黑板上剩下一个数。 在所有按这种操作方式最后得到的数中，最大的记作 max，最小的记作 min，该数列的极差定义为 M=max－min。请写出算法计算极差。

答案:

运行

Python

```python
def calculate_range(numbers):
    # 计算max：每次选最小的两个数相乘+1（升序排列）
    min_list = numbers.copy()
    while len(min_list) > 1:
        min_list.sort()
        a, b = min_list.pop(0), min_list.pop(0)
        min_list.append(a * b + 1)
    max_val = min_list[0]

    # 计算min：每次选最大的两个数相乘+1（降序排列）
    max_list = numbers.copy()
    while len(max_list) > 1:
        max_list.sort(reverse=True)
        a, b = max_list.pop(0), max_list.pop(0)
        max_list.append(a * b + 1)
    min_val = max_list[0]

    return max_val - min_val
```

解析：

- **max 值**：每次选择当前最小的两个数 `a` 和 `b` （升序排列），乘积 + 1 后加入数列，最终结果最大。
- **min 值**：每次选择当前最大的两个数 `a` 和 `b` （降序排列），乘积 + 1 后加入数列，最终结果最小。
- **极差**： `max_val - min_val` 。

5、(12 分)请在不排序的情况下，用分治法解最大最小问题，写出算法伪代码，并描述数组A=(48,12,61,3,5,19,32,7) 中求最大最小的过程

Plain Text

```
算法 MaxMin(A, low, high)
输入：数组A[low..high]
输出：(max, min)

if low = high then
    return (A[low], A[low])  // 只有一个元素，最大最小都是它
else if high = low + 1 then
    if A[low] < A[high] then
        return (A[high], A[low])
    else
        return (A[low], A[high])
    end if
else
    mid ← ⌊(low + high)/2⌋
    (max1, min1) ← MaxMin(A, low, mid)      // 左半部分递归
    (max2, min2) ← MaxMin(A, mid+1, high)   // 右半部分递归
    max ← max(max1, max2)                   // 合并最大值
    min ← min(min1, min2)                   // 合并最小值
    return (max, min)
end if
```

过程见期中考答案。

6、（10 分） n 个人参加拔河比赛，每个人有自己的重量。现在需要把他们分成两组进行比赛，每个人属于其中的一个组，两组的人员个数相差不能超过 1。为使比赛公平，请设计算法找出分配方案，使两组重量差最小。写出算法设计思想，列出伪代码并配备必要的注释，并分析算法的复杂度。

算法设计思想：

1. **问题转化**：将总重量 `sum` 分为两组，使两组重量差最小，且每组人数接近 `n/2` 。

2. **动态规划**：使用三维数组 `dp[i][j][s]` 表示前 `i` 个人中选 `j` 个人，总重量是否可以为 `s` 。
3. **状态转移**：对于第 `i` 个人，可选或不选，更新状态。
4. **结果查找**：遍历所有可能的重量 `s` ，找到最接近 `sum/2` 且人数为 `n/2` 或 `n/2+1` 的解。

伪代码：

```Plain Text
算法 TugOfWar(A[1..n])
输入：数组A[1..n]，表示n个人的重量
输出：两组的重量差最小值

sum ← 0
for i ← 1 to n do
    sum ← sum + A[i]
end for

// 初始化动态规划数组
dp[0][0][0] ← true
for i ← 1 to n do
    for j ← 0 to i do
        for s ← 0 to sum do
            dp[i][j][s] ← dp[i-1][j][s]  // 不选第i个人
            if j > 0 and s ≥ A[i] and dp[i-1][j-1][s-A[i]] then
                dp[i][j][s] ← true  // 选第i个人
            end if
        end for
    end for
end for

// 查找最优解
target ← sum / 2
min_diff ← +∞
k ← n / 2  // 目标人数

// 检查k和k+1（处理n为奇数的情况）
for s ← 0 to sum do
    if (dp[n][k][s] or (n is odd and dp[n][k+1][s])) then
        diff ← |sum - 2s|
        if diff < min_diff then
            min_diff ← diff
        end if
    end if
end for

return min_diff
```

复杂度分析：

- **时间复杂度**：$O(n^2 \cdot sum)$，其中 `sum` 是所有人的总重量。
- **空间复杂度**：$O(n^2 \cdot sum)$，主要用于动态规划数组。