1. 分治法：寻找「绝对众数」

题目

设计分治算法判断数组中是否存在绝对众数（出现次数 > n/2），若存在则返回该值，否则返回-1。要求时间复杂度为O(n log n)，并分析其正确性。

解题思路

1. **分治策略**
    - **分割**：将数组分为左右两半。
    - **解决**：递归地在左右子数组中寻找候选众数。
    - **合并**：比较左右子数组的候选众数，统计它们在当前整个数组中的出现次数，选择出现次数更多的作为候选。
2. **关键步骤**
    - 递归终止条件：当子数组长度为1时，该元素即为候选众数。
    - 合并时需验证左右候选众数在整个数组中的实际出现次数。
    - 最终全局验证候选众数是否满足绝对众数条件。
3. **正确性分析**
    - 绝对众数必定是某一半数组的众数或跨越中点的众数。
    - 分治合并过程确保了所有可能情况被覆盖。

```
def majority_element(nums):
    def helper(left, right):
        if left == right:
            return nums[left]
        mid = (left + right) // 2
        left_maj = helper(left, mid)
        right_maj = helper(mid+1, right)

        if left_maj == right_maj:
            return left_maj

        count_left = sum(1 for i in range(left, right+1) if nums[i] == left_maj)
        count_right = sum(1 for i in range(left, right+1) if nums[i] == right_maj)

        return left_maj if count_left > count_right else right_maj

    candidate = helper(0, len(nums)-1)
    return candidate if nums.count(candidate) > len(nums)//2 else -1
```

2. 回溯法：括号生成

题目

设计回溯算法生成n对合法括号的所有组合。例如n=3时输出["((()))","(()())","(())()","()(())","()()()"]。

解题思路

1. **回溯设计**
   - **选择路径**：每一步可选择添加左括号或右括号。
   - **约束条件**：
     - 左括号数 ≤ n
     - 右括号数 ≤ 左括号数
   - **终止条件**：字符串长度达到2n时保存结果。
2. **剪枝优化**
   - 仅当左括号未达上限时添加左括号。
   - 仅当右括号数小于左括号数时添加右括号。
3. **递归树分析**
   - 每个节点代表当前部分解，叶子节点为合法组合。

```python
def generateParenthesis(n):
    res = []
    def backtrack(s, left, right):
        if len(s) == 2*n:
            res.append(s)
            return
        if left < n:
            backtrack(s+'(', left+1, right)
        if right < left:
            backtrack(s+')', left, right+1)
    backtrack('', 0, 0)
    return res
```

3. 贪心算法：无重叠区间

题目

给定区间集合，求最多能选多少个互不重叠的区间。证明贪心选择最早结束的区间是最优策略。

解题思路

1. **贪心策略**
    - 按区间结束时间排序，优先选择结束最早的区间。
    - 每次选择后排除与其重叠的区间。
2. **正确性证明**
    - **最优子结构**：选择最早结束的区间后，剩余问题仍为最优子问题。
    - **反证法**：假设存在更优解，则可通过替换第一个区间为最早结束的区间，得到不劣于原解的结果。

```
def eraseOverlapIntervals(intervals):
    if not intervals: return 0
    intervals.sort(key=lambda x: x[1])
    count = 1
    end = intervals[0][1]
    for interval in intervals[1:]:
        if interval[0] >= end:
            count += 1
            end = interval[1]
    return len(intervals) - count
```

4. 动态规划：最长递增子序列

题目

设计DP算法求数组最长递增子序列长度。例如[10,9,2,5,3,7,101,18]的最长递增子序列长度为4（如[2,5,7,101]）。

解题思路

1. **状态定义**
   - `dp[i]`：以`nums[i]`结尾的最长递增子序列长度。
2. **状态转移方程**
   - 对于每个`i`，遍历`j < i`，若`nums[i] > nums[j]`，则`dp[i] = max(dp[i], dp[j] + 1)`。
3. **初始化与结果**
   - 初始值全为1（每个元素自身构成长度为1的子序列）。
   - 最终结果为`dp`数组的最大值。

```
def lengthOfLIS(nums):
    dp = [1] * len(nums)
    for i in range(1, len(nums)):
        for j in range(i):
            if nums[i] > nums[j]:
                dp[i] = max(dp[i], dp[j]+1)
    return max(dp)
```

6. 拉斯维加斯算法：随机快速选择

题目

设计拉斯维加斯算法在未排序数组中查找第k小元素，保证结果正确但运行时间随机。

解题思路

1. **算法设计**
   - 随机选择pivot，将数组分为三部分：小于、等于、大于pivot的子数组。
   - 根据k所在的区间递归处理对应子数组。
2. **关键特性**
   - 结果必然正确，但时间效率依赖随机选择的质量。

```python
import random

def quick_select(arr, k):
    pivot = random.choice(arr)
    less = [x for x in arr if x < pivot]
    eq = [x for x in arr if x == pivot]
    great = [x for x in arr if x > pivot]

    if k <= len(less):
        return quick_select(less, k)
    elif k <= len(less)+len(eq):
        return pivot
    else:
        return quick_select(great, k-len(less)-len(eq))
```

9. 动态规划：零钱兑换

题目

用动态规划求组成amount金额的最少硬币数。例如coins=[1,2,5], amount=11，应返回3（5+5+1）。

解题思路

1. **状态定义**
    - `dp[i]`：组成金额i所需的最少硬币数。
2. **状态转移方程**
    - `dp[i] = min(dp[i - coin] + 1)`，对所有coin ≤ i。
3. **初始化与边界**
    - `dp[0] = 0`，其他初始化为无穷大。
    - 若最终`dp[amount]`仍为无穷大，返回-1。

```python
def coinChange(coins, amount):
    dp = [float('inf')] * (amount+1)
    dp[0] = 0
    for i in range(1, amount+1):
        for coin in coins:
            if i >= coin:
                dp[i] = min(dp[i], dp[i-coin]+1)
    return dp[amount] if dp[amount] != float('inf') else -1
```

10. 蒙特卡洛算法：π近似计算

题目

用蒙特卡洛方法估算π值：在单位正方形内随机采样，统计落在1/4圆内的比例。

解题思路

1. **采样方法**
   - 在[0,1]×[0,1]范围内生成随机点(x,y)。
   - 判断是否满足$x^2 + y^2 \le 1$。
2. **概率计算**
   - 落在1/4圆内的概率为π/4，故π ≈ 4 * (命中数 / 总样本数)。

```
import random

def estimate_pi(n):
    count = 0
    for _ in range(n):
        x, y = random.random(), random.random()
        if x**2 + y**2 <= 1:
            count +=1
    return 4 * count / n
```

13. 贪心算法：活动选择问题

题目

给定一组活动（开始和结束时间），求最多能参加多少个不重叠的活动。证明贪心选择最早结束的活动是最优策略。

解题思路

1. **贪心策略**
    - 按结束时间排序活动，依次选择不冲突的最早结束活动。
2. **正确性证明**
    - **归纳法**：假设前k个选择是最优的，则第k+1个选择保持最优性。

```
def max_activities(activities):
    activities.sort(key=lambda x: x[1])
    count = 0
    end = -float('inf')
    for s, e in activities:
        if s >= end:
            count += 1
            end = e
    return count
```

14. 动态规划：编辑距离

题目

设计动态规划算法计算两个字符串的最小编辑距离（插入、删除、替换操作）。

解题思路

1. **状态定义**
    - `dp[i][j]`：将word1前i个字符转换为word2前j个字符的最小操作数。
2. **状态转移**
    - 若字符相同：`dp[i][j] = dp[i-1][j-1]`
    - 否则：`dp[i][j] = 1 + min(dp[i-1][j], dp[i][j-1], dp[i-1][j-1])`

```
def minDistance(word1, word2):
    m, n = len(word1), len(word2)
    dp = [[0]*(n+1) for _ in range(m+1)]
    for i in range(m+1):
        dp[i][0] = i
    for j in range(n+1):
        dp[0][j] = j
    for i in range(1, m+1):
        for j in range(1, n+1):
            if word1[i-1] == word2[j-1]:
                dp[i][j] = dp[i-1][j-1]
            else:
                dp[i][j] = 1 + min(dp[i-1][j], dp[i][j-1], dp[i-1][j-1])
    return dp[m][n]
```

16. 拉斯维加斯算法：随机化快速排序

题目

实现拉斯维加斯算法的随机化快速排序，保证结果正确但运行时间随机。

解题思路

1. **算法设计**
    - 随机选择pivot，分割数组为小于、等于、大于三部分。
    - 递归排序左右子数组。

```python
import random

def quicksort(arr):
    if len(arr) <= 1:
        return arr
    pivot = random.choice(arr)
    less = [x for x in arr if x < pivot]
    eq = [x for x in arr if x == pivot]
    great = [x for x in arr if x > pivot]
    return quicksort(less) + eq + quicksort(great)
```

20. 蒙特卡洛算法：估算积分

题目

用蒙特卡洛方法估算定积分$\int_0^1 x^2\,dx$，并分析误差。

解题思路

1. **采样方法**
   - 在[0,1]×[0,1]区域随机采样，统计落在$y \le x^2$曲线下的比例。
2. **积分计算**
   - 积分值≈命中比例（几何概率法）。

```
import random

def estimate_integral(n):
    count = 0
    for _ in range(n):
        x = random.uniform(0, 1)
        y = random.uniform(0, 1)
        if y <= x**2:
            count += 1
    return count / n
```

22. 回溯法：数独求解

题目

设计回溯算法填充数独，要求满足每行、每列、每个3x3宫格包含1-9不重复。

解题思路

1. **回溯框架**
    - 遍历每个空格，尝试填入1-9的数字。
    - 检查当前填入是否满足数独规则。
    - 若合法则递归填充下一个空格。
2. **剪枝条件**
    - 当前数字在行、列、宫格中已存在时跳过。

```python
def solveSudoku(board):
    def is_valid(row, col, num):
        for i in range(9):
            if board[row][i] == num or board[i][col] == num:
                return False
        start_row, start_col = 3*(row//3), 3*(col//3)
        for i in range(3):
            for j in range(3):
                if board[start_row+i][start_col+j] == num:
                    return False
        return True

    def backtrack():
        for i in range(9):
            for j in range(9):
                if board[i][j] == '.':
                    for num in '123456789':
                        if is_valid(i, j, num):
                            board[i][j] = num
                            if backtrack():
                                return True
                            board[i][j] = '.'
                    return False
        return True
    backtrack()
```