一般每秒能执行约 10^8 次运算（Python 可能要除以 10），可以据此估计能通过的时间复杂度，如下表所示。

数据范围	允许的时间复杂度	适用算法举例
$n \leq 10$	$O(n!)$ 或 $O(C^n)$	回溯、暴力搜索
$n \leq 20$	$O(2^n)$	状态压缩 DP
$n \leq 40$	$O(2^{n/2})$	折半枚举
$n \leq 10^2$	$O(n^3)$	三重循环的 DP、Floyd
$n \leq 10^3$	$O(n^2)$	二重循环的 DP、背包
$n \leq 10^5$	$O(n \log n)$	各类常用算法
$n \leq 10^6$	$O(n)$	线性 DP、滑动窗口
$n \leq 10^9$	$O(\sqrt{n})$	判断质数
$n \leq 10^{18}$	$O(\log n)$ 或 $O(1)$	二分、快速幂、数学公式

单调栈，例题：柱状图中最大的矩形

```
def largestRectangleArea(heights):
    heights.append(0) # 添加哨兵（高度0）确保所有柱子都能被弹出
    stack = [-1]  # 存储索引的栈，初始放入哨兵-1
    max_area = 0
    for i in range(len(heights)):
        # 当前柱子高度小于栈顶柱子高度时
        while stack and heights[i] < heights[stack[-1]]:
            # 弹出栈顶柱子作为矩形高度
            h = heights[stack.pop()]
            # 计算宽度：右边界i - 左边界(新栈顶) - 1
            w = i - stack[-1] - 1
            max_area = max(max_area, h * w)
        # 当前柱子入栈
        stack.append(i)
    return max_area
```

滑动窗口：维持左右边界都不回退的一段范围，来求解很多子数组的相关问题
关键：找到 **范围** 和 **答案指标** 之间的 **单调性关系**
过程：可以用简单变量或者结构来维护信息
大流程：求子数组在每个位置 开头或结尾的情况下的答案

```
for 枚举右边界:
    while 枚举左边界:
    if 条件:
        ans更新
```

lca问题（寻找最早公共祖先）

```
def lowestCommonAncestor(root,p,q):
    #递归函数返回值的含义是：在当前子树中是否找到p或q，或是它们的LCA。具体有3种状态：
    #返回None：当前子树中不存在p或q
    #返回非None节点：可能是p或q本身（找到其中一个）,可能是已找到的LCA(若pq都在)
    # 检查当前节点是否是p或q
    if root is None or root == p or root == q:
        return root
    l=lowestCommonAncestor(root.left,p,q)
    r=lowestCommonAncestor(root.right,p,q)
    # 左右子树均非空：说明 p 和 q 分布在当前节点两侧 → 当前节点是 LCA
    if l is not None and r is not None:
        return root
    # 左右子树均为空：子树中无目标节点 → 返回 None
    if l is None and r is None:
        return None
    # 一侧非空一侧空：返回非空子树结果（包含 LCA 或其中一个目标节点）
    return l if l is not None else r
```

Trie，判断nums中有没有一个是另一个的前缀

```
def insert(nums: List[str]) -> bool:
    trie = {}
    for i in nums:
        cur = trie
        for j in i:
            if j in cur and cur[j] == {}:
                return False
            if j not in cur:
                cur[j] = {}
            cur = cur[j]
    return True
```

二叉搜索树： 一棵二叉树，其中每个节点的值都大于其左子树中所有节点的值，且小于其右子树中所有节点的值。
修剪二叉搜索树：给定一个边界，移除树中所有值不在该范围内的节点，并保持剩余节点的二叉搜索树性质

```
def trimBST(cur,low,high): # 移除[low,high]的节点
    if cur is None:
        return None
    if cur.val<low:
        return trimBST(cur.right,low,high)
    if cur.val>high:
        return trimBST(cur.left,low,high)
    cur.left=trimBST(cur.left,low,high)
    cur.right=trimBST(cur.right,low,high)
    return cur
```

二叉树的序列化：使用特殊标记（如#）表示空节点，前序/后序遍历。

完全二叉树：一棵二叉树，除了最后一层（叶子节点所在的那一层）可能不是满的以外，其他所有层都被完全填满，并且最后一层的节点尽可能集中在左边。

对于一个索引为 i 的节点： 它的父节点索引是 floor((i-1)/2) (如果根在索引 0) 或 floor(i/2) (如果根在索引 1)。
它的左孩子索引是 2*i + 1 (根在 0) 或 2*i (根在 1)。
它的右孩子索引是 2*i + 2 (根在 0) 或 2*i + 1 (根在 1)。

01矩阵中面积最大的长方形：枚举每一行，以每一行作为底去进行单调栈即可（不连续就变成0，还要记得复用上一行的数据）。将二维矩阵问题转化为一系列一维直方图最大矩形问题。
对于矩阵的每一行，计算一个高度数组heights，heights[j]表示从当前行向上连续1的个数（包括当前行），如果当前元素是0，则heights[j] = 0。对每一行得到的高度数组，使用单调栈算法计算最大矩形面积。

```
# 括号嵌套树
class Node:
    def __init__(self, val, children):
        self.val = val
        self.children = children
alphabet = lambda x : isinstance(x,str) and x in 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
stack = []
s = input()
for i in s:
    if alphabet(i):
        stack.append(Node(i,[]))
    if i == '(':
        stack.append(i)
    if i == ',':
        pass
    if i == ')':
        children = []
        while stack[-1] != '(':
            children.append(stack.pop())
        stack.pop() # remove '('
        stack[-1].children = children[::-1]
root = stack[-1]
if alphabet(root):
    root = Node(root,[])
```