1、复杂度排序：

$O(1) < O(\log\log n) < O(\log n) < O(\sqrt{n}) < O(n^{\frac{1}{3}}) < O(n)$

$< O(n\log n) < O(n\log^2 n) < O(n^2) < O(n^3) < O(2^n)$

$< O(n!) < O(n^n)$

精简版：$O(1) < O(\log n) < O(n) < O(n\log n) < O(n^2) < O(n^3) < O(2^n)$

$< O(n!) < O(n^n)$.

对数 < 多项式 < 指数 < 阶乘 < 幂塔

2、主定理 $T(n) = aT(\frac{n}{b}) + f(n)$

先求 $n^{\log_b a}$.，后比较 $f(n)$ vs $n^{\log_b a}$

子问题.

① 子问题更重要. $T(n) = \Theta(n^{\log_b a})$

② 势均力敌. $T(n) = \Theta(n^{\log_b a}\log n)$

③ 本层工作更多 $T(n) = \Theta(f(n))$.

经验复杂度

顺序查找 $O(n)$.

二分查找 $O(\log n)$

快速排序(平均). $O(n\log n)$

归并排序. $O(n\log n)$

冒泡/插入/选择 $O(n^2)$

BFS/DFS. $O(V+E)$

3、栈：

堆栈 ≠ 栈。 是历史/口语/混同名词

顺序栈、

链栈

下溢：存储结构为空，还继续删除/取出元素

上溢：存储结构已满，还插入

存储密度 = $\dfrac{\text{数据域所占存储空间}}{\text{数据域 + 结构域所占存储 space}}$

~~顺序~~存储结构：描述数据的逻辑结构在 computer 中的具体存储方式

逻辑结构：描述数据元素之间的逻辑关系，与如何存放无关

双栈共享空间时，栈满条件为 top1 + 1 = top2

即两栈中间差一格

单链表只有 next，无 prior；双链表 next，prior 都有

首元结点、和头指针

第1个有值节点 第1个之前的空结点

循环链表/队列，带头节点的头指针 front 指向头结点

十. 排序：

插入排序算法：把序列分成"已排序区"和"未排序区"
每次从未排序区取一个元素，插入到已排序区

希尔排序：① 选1个步长.(gap)
② 把原序列按照 gap 分成若干组
③ 对每一组进行排序.
④ 缩小 gap，直到 gap=1

最坏的时间复杂度：$O(N^2)$
不是稳定的
最好时间复杂度为 $O(N)$，已排好序

排序的稳定性：若两个元素关键字相等，排序后它们的相对顺序是否保持不变。

归并排序：1. 分、2. 治、3. 合.
1. 分：序列一分为二.
2. 治：1个元素天然有序
3. 合：两2个有序序列合成一个更大的

time 复杂度 最好/平均/最坏：$O(N \cdot \log N)$.
稳定：每趟归并工作量 $O(N)$.
归并趟数为 $\log_2 N$.

快速排序：
① 选枢轴 pivot. 常见：第1个/最后1个/随机元素.
② 划分. 左边 ≤ pivot，右边 ≥ pivot. pivot 放在最终位置.
③ 递归排序. 对左边、右边分别快速排序.

平均时间复杂度 $O(N\log N)$
最坏（序列已基本有序）$O(N^2)$
不稳定

递归次数：调用了多少次函数
递归栈深度：同时压在栈里的层数.
先排短分区能减少递归栈深度.

空间复杂度：最好/平均：$O(\log N)$
最坏：$O(N)$

块间有序：整块比较.

冒泡排序：相邻元素两两比较，逆序就交换，大的元素逐趟"冒"到后面.

最坏时间复杂度 $O(n^2)$

平均情况 $O(n^2)$.

最好情况 $O(n)$（有提前结束）

空间复杂度 $O(1)$.

稳定

简单选择排序：第i趟.在第i个位置及其之后的所有元素中选出最小的，与第i个位置交换.

时间复杂度 $O(n^2)$ $\frac{n(n-1)}{2}$.

最好/最坏/平均都是 $O(N^2)$.

空间复杂度 $O(1)$. 不稳定.

合并排序：把序列不断拆分成更小的有序序列，
二归并排序　再把它们合并成一个大序列

基数排序：不比较大小，按"数位"来分配和收集.

按次位优先 LSD：（最低位优先）最低位开始排 → 个 → 十 → 百 → ...

MSD（最高位优先）

链式基数排序：用链表作为桶.

1. 分配：根据当前数位.按0~9.放对应桶（队列）

2. 收集：~~把~~按0~9.把元素接回链表

桶内保持稳定.

5. 树与二叉树:

$n = n_0 + n_1 + n_2$ 任何二叉树都成立

$n_0 = n_2 + 1$. 任何二叉树都成立.

$n - 1 = n_1 + 2n_2$. 任何二叉树都成立

二叉树中,父子关系是一对多.

☆完全二叉树中的前序序列中,若结点 u 是结点 v 的祖先,则 u 一定在 v 之前.

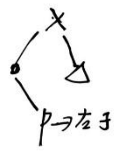
☆将一棵树转化成二叉树,则树 preorder 与对应二叉树的 preorder 序列相同.
(换成中序和后序都不成立)

但是☆.把一棵树换成二叉树后,

原树的 preorder = 二叉树 preorder

…… postorder = …… inorder

某结点的中序前驱 = 该结点左子树中"最右下"的结点



p→左子树中最右结点.

非空的二叉树一定满足:某结点若有左孩子,则其中序前驱一定没有右孩子.

完全二叉树的高度. $h=\lfloor \log_2(N+1) \rfloor$

☆计算. 对任意二叉树都成立:　　（前序相对次序）

1. 祖先一定在子孙之前.

2. 如果两个结点的LCA（最近公共祖先）是p, 并且一个在p的左子树，一个在p的右子树，则左子树先出现.

3. 如果一个在另一个子树, 按1判.

根在1号位: 左孩子: $2i$

右孩子: $2i+1$

父亲: $\lfloor i/2 \rfloor$

☆二叉树中有双子女的父结点，在中序遍历中后继一定是右孩子.

表达式树: · 叶子结点: 操作数（常量、变量）

内部结点: 运算符（+、-、×、÷ etc）

左右子数: 分别表示该运算符的左右操作数.

tips: 优先级低的运算符 → 离根更近，反之 → 离根远.

中序遍历 = 中缀表达式（通常需加括号）

前序遍历 = 前缀表达式（波兰式）

后序遍历 = 后缀表达式（逆波兰式）

后缀表达式 → 表达式树: ① 栈法，时间$O(n)$，空间$O(n)$（最好的）

② 递归分割: 时间最坏$O(n^2)$.

③ 先转中缀，再建树，时间$O(n)$. 但步骤冗余

可以唯一确定一棵二叉树的情况：

① 前序 + 中序
② 中序 + 后序 } → 生成二叉树要会
③ 层次 + 中序
④ 前序 + 后序 + 附加约束

6、堆（Heap）：① 完全二叉树

{ 大根堆：root ≥ 左右孩子
  小根堆：root ≤ 左右孩子

建堆的方法：1、逐个插入法（非最优）
① 每插入一个元素进行 up-heap / sift-up
② 每次插入调整 logN 层，插入 N 次，
总时间复杂度 O(N·logN)

2. 自底向上建堆（线性建堆）
① N 个元素直接按顺序放在数组
② 从最后一个非叶子结点开始，依次 down-heap / sift-down
③ 一直到根结点

时间复杂度为 O(N)

即 $\lfloor \frac{N}{2} \rfloor$

Tips：在最小堆中找最大元素，最坏时间复杂度是 O(N)
（就是把每个叶子结点都确定一下）

Tips：堆默认是层序遍历，默认删除根　　replace

Tips：DeleteMin 就把小根堆的根删了，用最后一个叶子
再调整

可合并堆：合并两个堆时间复杂度为 $O(\log N)$.

eg：左式堆、斜堆、二项堆/斐波那契堆

7. 查找

顺序查找：从头到尾依次比较关键字，直到找到查完.

时间复杂度. ~~OC~~ 最好 $O(1)$. 最坏 $O(n)$. 平均 $O(n)$

适用条件：不要求有序，顺序表/链表都 OK.

折半查找/二分查找：在有序表，通过不断取中间元素，将查找区间缩小一半.

时间复杂度：最好 $O(1)$. 最坏 $O(\log n)$ 平均 $O(\log n)$.

适用条件：数据必须有序、通常要求顺序存储.

Tips：衡量查找算法~~的~~效率的主要标准是平均查找长度.

8. 二叉排序树/二叉查找树(BST)

BST是一棵二叉树，对任意结点：

左子树中所有关键字小于该节点关键字.

右子树、、、 大于 、、 ——

(若允许相等，题目会说明放在左边/右边)

性质：① 中序遍历有序：

对BST中序遍历，得到的序列是递增的.

② 查找效率为 $O(h)$

最好 $O(\log n)$. 最坏 $O(n)$

③ 左<根<右.(全局有序)

④ 插入结果与插入顺序有关.

不同的插入顺序，BST可能完全不同.

操作：① 查找 ·—

② 插入. ∴查找失败的位置，插入叶子.

③ 删除：(1) 叶子结点：直接删.

(2) 只有1个孩子：用孩子替代

(3) 有两个孩子：用中序前驱/中序后继替代.

AVL树（自平衡二叉排序树）

性质：① 是 BST.

② 平衡因子. $BF = h(左) - h(右)$

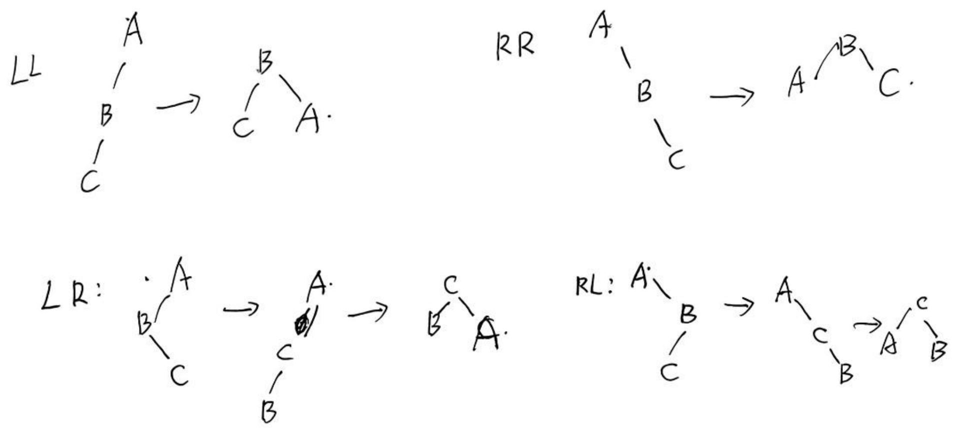
$\in \{-1, 0, 1\}$.

③ AVL树 $h = O(\log n)$，查找时间为 $O(h) = O(\log n)$

④：当 $|BF| = 2$ 时 需旋转来恢复.

多画图一下 这里找到本质才行

LL 型
RR 型
LR 型
RL 型
} → 自己记了得



Tips: AVL树的高度.

$N(h)$ = 高度为 h 的AVL所包含的最少结点树

$N(h) = N(h-1) + N(h-2) + 1$.

9. ~~生成树~~ 图.

生成树的判定: 原图: $G=(V,E)$. $|V|=n$. $G'=(V,E')$.
子图

☆ $V(G')=V(G)$ + G'连通图 + 边数为$n-1$ / $|E'|=n-1$

⇒ 子图G'为G的生成树.

对包含全部顶点的G', 以下两两组合都能判定是生成树:

① 连通 + $|E'|=n-1$ ⇒ 生成树.

② 无环 + $|E'|=n-1$ ⇒ 生成树

③ 连通 + 无环 ⇒ 生成树

④ 连通 + 删除任一条边就不连通(极小连通) ⇒ 生成树

⑤ 无环 + 加任意一条边就出现环(极大无环) ⇒ 生成树

MST 定理: 在任意一个环中, 权值最大的边不属于任何MST.

but. 某一环边权值各不同, 权值最小的边一定在某一个MST中 错X

定理: 若连通图各边权值均不相同, 则该图的MST唯一。

(MST) 最小生成树: 在一个带权无向连通图中, 选$n-1$条边, 使所有顶点连通且边权之和最小的那棵树

☆ 图的遍历:

DFS (深度 first) 对应 二叉树的 preorder.
Depth

BFS (广度 first) 对应 二叉树的 层序遍历
Breadth

Tips: 在图中, 访问标志位 visited[] 是防止"重复访问"的唯一保证: ①可能有环, ②即使无环, 也可能多路径相汇.

☆☆☆

	BFS.	DFS.
基本策略	由近及远, 一层一层.	由深到浅, 一条路到底.
优先访问	距离起点最近的顶点	当前路径最深的顶点
类比	树的层序 order	树的 preorder
需要的数据结构	queue	stack/递归
时间复杂度	$O(V+E)$	$O(V+E)$

简单图：不含自环、也不含重边的图。

多重图（不允许自环）

伪图.

Tips：无向图中，若边数 ≥ 顶点数，则一定有环.

对一棵无向树：边 = 顶点 −1.

无环且连通 ⟺ 树 ⟺ 边 = 顶点 −1.

在无向图中，若任意两个顶点之间都存在一条路径，则称该图为连通的

邻接矩阵：$A = (a_{ij})$；

缺点，空间复杂度为 $O(n^2)$

无边：$a_{ij} = 0$ 或 ∞.

有边：无权图：$a_{ij} = 1$. 有权图：a_{ij} = 权值.

判断强连通：在有向图中，若任意两个顶点 u、v，都存在 $u \to v$ 和 $v \to u$ 的路径，则该图是强连通的.

任意两点，双向可达

可达矩阵.（用 Warshall 算法获得）

$R[i][j] = R[i][j] \vee (R[i][k] \wedge R[k][j])$

邻接矩阵 → 可达矩阵

规则：若存在边 $i \to j$：$R[i][j] = 1$.

不存在边 ~~i→j~~：$R[i][j] = 0$

主对角线：$R[i][i] = 1$.

拓扑序：对一个有向无环图（DAG）的所有顶点进行线性排序，使得若存在有向边 $u \to v$，则在序列中 u 一定排在 v 之前

Tips：拓扑序不一定唯一！

拓扑排序怎么求？

① Kahn法（入度法）

(1) 找入度为0的顶点，输出

(2) 删除它的出边

(3) 重复

② DFS

(1) 对每个未访顶点 DFS

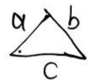
(2) DFS过程中：当一个点所有出边邻接点都处理完，把这个点压入栈/放到结果末尾

(3) 最后把"压栈序列"逆序输出，就是拓扑序

Dijkstra算法：在"边权非负"的图中，求某一源点到其余各顶点的最短路径

核心思路：每一步从"尚未确定的点"中，选一个当前距离源点最近的顶点，把短路定死

时间复杂度：n个顶点，e条边. $O(n^2)$



最短路径三角形不等式：

在无向图（或有向图按方向成立）

$$a \le b+c$$

$$a \ge |b-c|$$

第61

Prim 算法：从一个顶点出发，逐步扩展，始终选取"当前树到外部的最小权边"，构造 MST

Process：1. 任选1个起点，加入生成树集合 T.
2. 在所有"T内顶点 → T外顶点"的边中，选权值最小的一条
3. 把这条边和对应顶点加入 T.
4. 重复，直到所有顶点都在 T 中

Kruskal 算法：按"边权从小到大排序"，只要不形成环，就加入生成树

Process：1. 把所有边按权值从小到大排序
2. 从最小边扫描
3. 加入该边不形成环 → 选
4. 若加入该边形成环 → 过.
5. 选满 n−1 条边.

prim 和 Kruskal 适用于：无向图、带权图（可相等、可为负）、连通图.