数据结构　期末考试试卷

（2026 年春季学期 · 考后回忆整理版）

课程　数据结构　　**考试日期**　2026 年 6 月 25 日

任课教师　吴忠华　　**回忆整理**　侯思涵

题量　共 6 道大题　　**适用**　北京林业大学 / 严蔚敏教材

说明：本试卷为考后凭记忆复原，部分题目的具体数据（10 个待排数、二叉排序树结点、图的边权等）均由 Claude Opus 4.8 MAX 生成，结构与考查点与原卷一致，仅供复习自测之用。其中，代码部分不做要求，写出算法思路即可。文末附 Claude Opus 4.8 MAX 给出的**参考答案**，由人工核对，确认无误，建议先独立作答再对照。

第一部分　试 题

第一题（渐进复杂度判断）

设下列函数（n 为正整数，$n \to \infty$）：

$$f_1(n) = \log_2(n^2+1), \qquad f_2(n) = n\log_2 n, \qquad f_3(n) = n^{3/2}, \qquad f_4(n) = \sum_{i=1}^{n} i,$$

$$f_5(n) = f_5(n-1) + n^2 \ (f_5(0)=0), \qquad f_6(n) = 2^n + n^{100}, \qquad f_7(n) = n!, \qquad f_8(n) = n^n.$$

判断下列关于其渐进增长率的命题是否正确，正确打“✓”，错误打“×”：

(1) $f_2(n) = O\big(f_1(n)\big)$

(2) $f_3(n) = O\big(f_2(n)\big)$

(3) $f_4(n) = \Theta\big(f_3(n)\big)$

(4) $f_5(n) = O\big(f_4(n)\big)$

(5) $f_6(n) = O\big(f_5(n)\big)$

(6) $f_7(n) = O\big(f_6(n)\big)$

(7) $f_8(n) = O\big(f_7(n)\big)$

(8) $f_1(n) = \Omega\big(f_8(n)\big)$

第二题（线性表的插入与查找）

设有 n 个元素，存储地址（下标）从 0 到 $n-1$。

(1) 对于**顺序表**，在第 i 个位置插入元素 x，写出具体算法，并分析最坏情况下的时间复杂度。

(2) 对于**单链表**，在第 i 个结点处插入一个新结点，写出具体算法，分析最坏情况下的时间复杂度，并说明与顺序表插入的区别。

(3) 在一个**升序排列**的顺序表中，给出一种高效的查找方法，并分析其时间复杂度。

第三题（栈和队列）

(1) 给出栈（Stack）和队列（Queue）的定义。

(2) 说明如何用**两个栈**模拟一个队列，写出入队、出队的具体过程，并分别分析各元素入队、出队操作的**均摊（amortized）时间复杂度**。

第四题（外部排序与置换选择）

(1) 以 **2-路归并**为例，说明外部排序的具体过程，并解释**为什么需要归并**。

(2) 说明**置换选择排序**的具体过程，并解释**为什么生成初始归并段时不用普通内部排序，而采用置换选择排序**。

(3) 现有 10 个待排数据，按读入顺序为：

$$\boxed{29,\ 14,\ 61,\ 38,\ 15,\ 30,\ 51,\ 49,\ 39,\ 46}$$

设内存**工作区大小为 3**，请用置换选择排序生成全部初始归并段（结果应为 2 个归并段），写出生成过程。

第五题（二叉树与二叉排序树 / AVL 树）

(1) 已知某二叉树的先序遍历序列为 `abc###de##f##`（其中 `#` 表示空指针），请还原并画出该二叉树。

(2) 依次插入 7 个关键字：

$$\boxed{50,\ 30,\ 70,\ 20,\ 40,\ 60,\ 80}$$

画出建立的二叉排序树；写出查找 35 的具体过程，并说明二叉排序树查找的时间复杂度。

(3) 在上述序列之后再依次插入 10 和 90。若把整个插入序列建成一棵 **AVL（平衡二叉）树**，插入 10、90 时是否需要旋转？若需要旋转，请指出是 LL、RR、LR 还是 RL 型，并画出调整过程。

第六题（图）

已知无向带权图 G，顶点集 $V = \{A, B, C, D, E, F\}$，边与权值如下：

边	A-B	A-C	B-C	B-D	C-D	C-E	D-E	D-F	E-F
权	7	3	8	5	2	6	9	4	1

(1) 从顶点 A 出发（同一结点的邻接点按**字母顺序**访问），写出**深度优先**与**广度优先**遍历序列。

(2) 用 **Kruskal 算法**求最小生成树，写出每一步所选的边及其权值，并画出最小生成树。

(3) **堆排序**：以上述各边权值构成数组（按上表从左到右）$[7, 3, 8, 5, 2, 6, 9, 4, 1]$。构造**大根堆**并画出；然后连续**两次**取堆顶元素并调整堆，画出过程图与两趟排序结果；最后分析堆排序的时间复杂度。

第二部分　参考答案

第一题 参考答案

先把各函数化为渐进阶：

$$f_1=\Theta(\log n),\quad f_2=\Theta(n\log n),\quad f_3=\Theta(n^{1.5}),\quad f_4=\Theta(n^2),$$
$$f_5=\Theta(n^3),\quad f_6=\Theta(2^n),\quad f_7=\Theta(n!),\quad f_8=\Theta(n^n).$$

其中 $f_4=\frac{n(n+1)}{2}=\Theta(n^2)$；$f_5=\sum_{i=1}^{n}i^2=\frac{n(n+1)(2n+1)}{6}=\Theta(n^3)$；$f_6$ 中 2^n 远大于多项式项 n^{100}，故 $f_6=\Theta(2^n)$。由此得到严格的增长率排序：

$$\boxed{f_1\prec f_2\prec f_3\prec f_4\prec f_5\prec f_6\prec f_7\prec f_8}$$

本题 8 个命题都把**增长更快**的函数错误地用**增长更慢**的函数来界定（前 7 题误用 O 作上界，第 8 题误用 Ω 作下界），所以**全部为假，答案全打** $\times$：

小题	命题	判断与理由
(1)	$f_2=O(f_1)$	$\times$　$n\log n$ 高于 $\log n$，不是其 O。
(2)	$f_3=O(f_2)$	$\times$　$n^{1.5}$ 高于 $n\log n$（$n^{0.5}\succ\log n$）。
(3)	$f_4=\Theta(f_3)$	$\times$　n^2 与 $n^{1.5}$ 不同阶。
(4)	$f_5=O(f_4)$	$\times$　n^3 高于 n^2。
(5)	$f_6=O(f_5)$	$\times$　指数 2^n 高于任意多项式。
(6)	$f_7=O(f_6)$	$\times$　$n!$ 自 $n\geq 4$ 起高于 2^n。
(7)	$f_8=O(f_7)$	$\times$　n^n 高于 $n!$。
(8)	$f_1=\Omega(f_8)$	$\times$　$\log n$ 最小，不可能是 n^n 的下界。

第二题 参考答案

(1) 顺序表插入

设逻辑位置 i（$1\leq i\leq n+1$）对应存储下标 $i-1$。要在位置 i 插入 x，需将原第 $i\sim n$ 个元素整体后移一位：

```
// Insert x at logical position i (1..n+1) in a sequential list of
    length n.
```

```
bool Insert(int A[], int &n, int i, int x) {
    if (i < 1 || i > n + 1) return false;      // illegal position
    for (int j = n; j >= i; --j)               // shift elements
        right
        A[j] = A[j - 1];
    A[i - 1] = x;                              // place the new
        element
    ++n;
    return true;
}
```

最坏时间复杂度：当 $i = 1$（在表头插入）时，需后移全部 n 个元素，比较/移动次数为 n，故为 $O(n)$。

(2) 单链表插入

带头结点的单链表，先从头结点出发找到第 $i-1$ 个结点，再“接线”插入：

```
struct Node { int data; Node* next; };

// Insert a new node with value x as the i-th node (1-based).
bool Insert(Node* head, int i, int x) {
    Node* p = head;                // dummy head
    int j = 0;
    while (p && j < i - 1) {        // locate the (i-1)-th node
        p = p->next; ++j;
    }
    if (!p) return false;          // position out of range
    Node* s = new Node{ x, p->next };
    p->next = s;                   // O(1) splice
    return true;
}
```

复杂度要分两部分看：

- 插入这一步本身（只改两个指针，不移动任何元素）是 $O(1)$——这正是链表相对顺序表的核心优势；
- 找到第 $i-1$ 个结点：链表不支持随机访问，必须从头逐个走，最坏（$i = n+1$，即在表尾插入）需 $O(n)$。

因此：若题目是“从头结点开始、在第 i 个位置插入”，整个操作的最坏复杂度是 $O(n)$，

瓶颈在定位而非插入；若题目已经给出指向插入位置（前驱）的指针，则插入就是 $O(1)$。

与顺序表的区别：顺序表插入必须*移动元素*（最坏 $O(n)$ 次移动）；链表插入*不移动元素*、只改指针（$O(1)$），代价从“移动元素”转移到了“查找位置”上。换言之，链表把插入做到了 $O(1)$，但失去了随机访问。

(3) 升序顺序表的高效查找——折半（二分）查找

```
// Binary search in ascending array A[0..n-1]; return index, or -1 if absent.
int BinarySearch(int A[], int n, int key) {
    int low = 0, high = n - 1;
    while (low <= high) {
        int mid = low + (high - low) / 2;
        if (A[mid] == key)      return mid;
        else if (A[mid] < key)  low  = mid + 1;
        else                    high = mid - 1;
    }
    return -1;
}
```

每次比较把查找区间减半，至多比较 $\lfloor \log_2 n \rfloor + 1$ 次，**时间复杂度** $O(\log n)$（要求表为顺序存储且有序）。

第三题 参考答案

(1) 定义

栈：限定仅在表的一端（栈顶）进行插入和删除的线性表，特性为**后进先出（LIFO）**。
队列：限定在表的一端（队尾）插入、另一端（队头）删除的线性表，特性为**先进先出（FIFO）**。

(2) 两个栈实现队列

用 `inStack` 负责入队、`outStack` 负责出队：入队总压入 `inStack`；出队时若 `outStack` 为空，则把 `inStack` 中元素全部倒入 `outStack`（顺序恰好反转，符合 FIFO），再从 `outStack` 弹出。

```
stack<int> inStack, outStack;

```

```
void enqueue(int x) {              // always O(1)
    inStack.push(x);
}

int dequeue() {                    // amortized O(1)
    if (outStack.empty()) {        // transfer only when out is empty
        while (!inStack.empty()) {
            outStack.push(inStack.top());
            inStack.pop();
        }
    }
    int x = outStack.top();
    outStack.pop();
    return x;
}
```

均摊复杂度分析：每个元素在其“一生”中至多经历 4 次单栈操作——压入 inStack、弹出 inStack、压入 outStack、弹出 outStack。n 个元素总操作量为 $O(n)$，平摊到每次入队 / 出队即为**均摊** $O(1)$。虽然某一次出队（恰好触发倒栈）最坏可达 $O(n)$，但这类昂贵操作不会连续发生，平摊后仍是常数。

第四题 参考答案

(1) 2-路归并的外部排序过程，及为什么要归并

当待排数据量远大于内存时，无法一次性读入内存排序，分两个阶段：

阶段 1： **生成初始归并段（顺串）**。把文件按内存容量分块读入，在内存中用内部排序排好后写回外存，得到若干段内部有序的“初始归并段”。

阶段 2： **逐趟归并**。2-路归并即每次取两个有序段，边读边比较、按序写出，合并成一个更长的有序段。每归并一趟，段数减半；设初始有 m 个段，则需 $\lceil \log_2 m \rceil$ 趟，直到只剩一个段（全局有序）。

为什么要归并：内存装不下全部数据，只能“分块排好再拼起来”。归并使用顺序读写外存，I/O 效率高；外部排序的瓶颈是访问外存的次数，逐趟归并正是把多个有序段高效合并、并尽量减少 I/O 趟数的手段。

(2) 置换选择排序过程，及为何优于普通内部排序

设工作区可容纳 w 个记录（本题 $w=3$），用一个“最小值选择”结构维护工作区，记 MINIMAX 为当前归并段已输出的最后一个关键字：

1. 从输入读入 w 个记录填满工作区；
2. 在工作区中选出关键字不小于 MINIMAX 的最小记录，输出之，并更新 MINIMAX；
3. 从输入读入下一个记录补入空位：若其关键字 $\geq$ MINIMAX，归入本段；否则冻结，留待下一段；
4. 重复 2～3；当工作区中已选不出 $\geq$ MINIMAX 的记录（全部被冻结）时，本段结束，解冻所有记录、重置 MINIMAX 开始新段；
5. 直到输入耗尽且工作区为空。

为什么用置换选择而非普通内部排序：普通内部排序每段长度恰等于内存容量 w，段数较多；而置换选择由于不断补入新记录，**初始归并段平均长度可达** $2w$（E. F. Moore 的“扫雪机”模型）。段更长 $\Rightarrow$ 段数 m 更少 $\Rightarrow$ 归并趟数 $\lceil \log_2 m \rceil$ 更少 $\Rightarrow$ 读写外存次数更少，外排总时间更短。

(3) 对给定 10 个数据的置换选择（$w=3$）

读入顺序：$29, 14, 61, 38, 15, 30, 51, 49, 39, 46$。带 $*$ 表示该记录被冻结（属于下一段）。

步	读入	工作区（读入后）	MINIMAX	输出	说明
初	29,14,61	$\{29, 14, 61\}$	$-\infty$	—	填满工作区，开始段 1
1	38	$\{29, 38, 61\}$	14	**14**	选 14；38≥14 入本段
2	15	$\{15^*, 38, 61\}$	29	**29**	选 29；15<29 冻结
3	30	$\{15^*, 30^*, 61\}$	38	**38**	选 38；30<38 冻结
4	51	$\{15^*, 30^*, 51^*\}$	61	**61**	选 61；51<61 冻结
	—	全部冻结	—	—	**段 1 结束**，解冻
5	49	$\{49, 30, 51\}$	$-\infty$	**15**	开始段 2；选 15；49 入本段
6	39	$\{49, 39, 51\}$	15	**30**	选 30；39 入本段
7	46	$\{49, 39, 51\}$	30	**39**	选 39；46 入本段
8	（空）	$\{49, 46, 51\}$	39	**46**	选 46；输入耗尽
9	（空）	$\{49, \_, 51\}$	46	**49**	选 49
10	（空）	$\{\_, \_, 51\}$	49	**51**	选 51；工作区空，**段 2 结束**

最终结果：2 个初始归并段

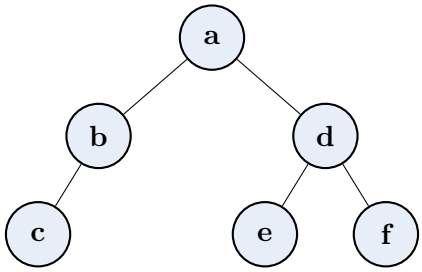
归并段 1：14, 29, 38, 61　　（长度 4）

归并段 2：15, 30, 39, 46, 49, 51　　（长度 6）

第五题 参考答案

(1) 由先序序列还原二叉树

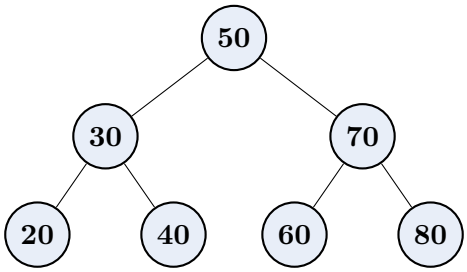
按先序“根—左—右”递归读取，# 即空子树，得到：



即 a 的左子树为 b（b 仅有左孩子 c），右子树为 d（d 的左、右孩子为 e, f）。6 个结点对应 7 个空指针，与序列中 7 个 # 吻合。

(2) 二叉排序树、查找 35 与复杂度

按 $50, 30, 70, 20, 40, 60, 80$ 依次插入，得到一棵 3 层满二叉排序树：



查找 35 的过程（与各结点比较）：

$$35 < 50\ (\text{向左}) \ \rightarrow\ 35 > 30\ (\text{向右}) \ \rightarrow\ 35 < 40\ (\text{向左}) \ \rightarrow\ \text{空，查找失败.}$$

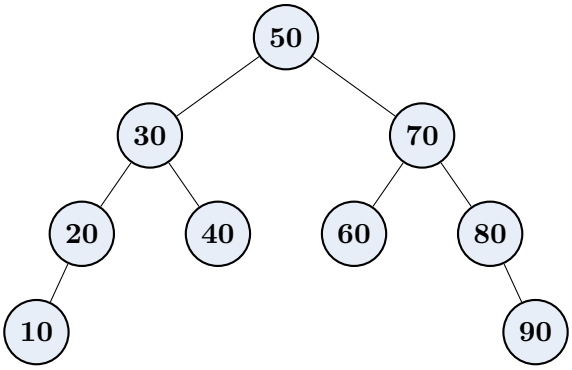
比较序列为 $50 \rightarrow 30 \rightarrow 40$，共 3 次比较后落到 40 的空左指针，故 35 不在树中。

时间复杂度：二叉排序树查找的比较次数等于被查结点（或失败位置）的层数。平均情况下树较平衡，为 $O(\log n)$；最坏情况（按有序序列插入退化成单支链）为 $O(n)$。

(3) 再插入 10、90，AVL 树是否旋转

结论：都不需要旋转。原树是 3 层满树，所有结点平衡因子均为 0，留有“缓冲”：插入新的最小值 10 只把最左路径加深一层，沿途结点平衡因子至多变为 $+1$；插入新的最大值 90 只把最右路径加深一层，沿途结点平衡因子至多变为 -1。全程 $|\mathrm{BF}| \le 1$，未破坏平衡条件，故无需任何旋转。

依次插入 10（成为 20 的左孩子）、90（成为 80 的右孩子）后的 AVL 树：

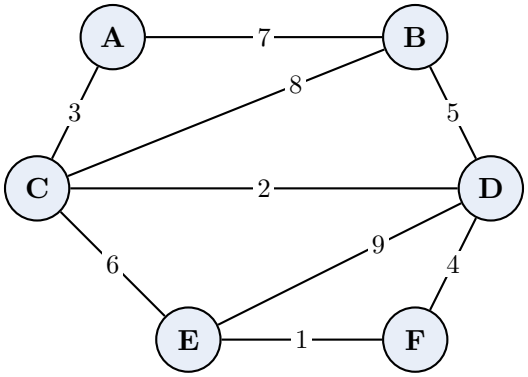


平衡因子验证（$\mathrm{BF} = h_{左} - h_{右}$）：插入 10 后，结点 $20, 30, 50$ 的 BF 依次为 $+1, +1, +1$；再插入 90 后，结点 $80, 70$ 的 BF 为 $-1, -1$，而根 50 因左右子树同时增高、BF 又回到 0。两次插入后所有 $|\mathrm{BF}| \le 1$，平衡始终满足。

小结：只有当插入使某结点 BF 达到 ± 2 时才需旋转（LL / RR / LR / RL）；本题向一棵满树的两端各加一个极值，恰好用掉“缓冲”而未越界，因此不触发旋转。

第六题 参考答案

各顶点邻接表（按字母序）：A:B, C；B:A, C, D；C:A, B, D, E；D:B, C, E, F；E:C, D, F；F:D, E。图示如下：



(1) 深度优先与广度优先遍历

DFS（从 A，邻接点取字母最小未访问者）：$A \to B \to C \to D \to E \to F$。
路径说明：A 选 B；B 选 C；C 选 D；D 选 E；E 选 F；F 邻点皆已访问，回溯结束。

BFS（从 A，同层按字母序入队）：$A \to B \to C \to D \to E \to F$。
说明：A 出队后入队 B, C；B 出队入队 D；C 出队入队 E；D 出队入队 F；其后无新顶点。

（本图较稠密且按字母序处理，两种遍历序列恰好相同，均为 A, B, C, D, E, F。）

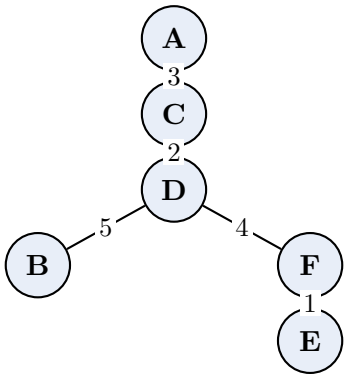
(2) Kruskal 最小生成树

把所有边按权值升序排序，依次选取“不构成回路”的边，直到选满 $|V| - 1 = 5$ 条：

次序	候选边	权	是否选取	当前连通情况
1	E-F	1	✓	$\{E, F\}$
2	C-D	2	✓	$\{C, D\}$
3	A-C	3	✓	$\{A, C, D\}$
4	D-F	4	✓	$\{A, C, D, E, F\}$
5	B-D	5	✓	$\{A, B, C, D, E, F\}$，已连通，结束

（其后 C-E(6)、A-B(7)、B-C(8)、D-E(9) 均会成环，舍弃。）

最小生成树（总权值 $1 + 2 + 3 + 4 + 5 = \boxed{15}$）：

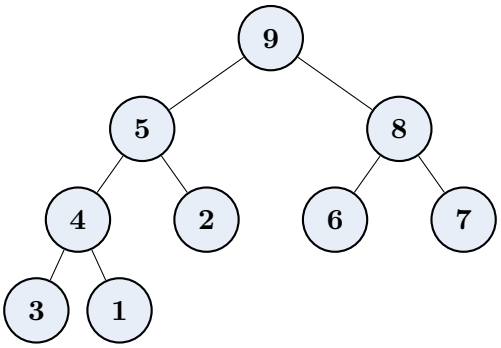


(3) 堆排序

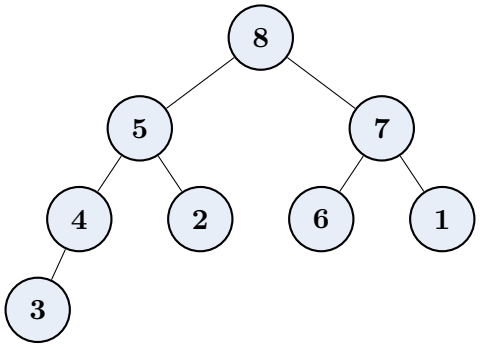
对数组 $[7, 3, 8, 5, 2, 6, 9, 4, 1]$（下标 $1 \sim 9$）建**大根堆**：从最后一个非叶结点 $\lfloor 9/2 \rfloor = 4$ 开始，自底向上反复“筛下”（sift-down）。建堆过程：

处理结点	数组状态（粗体为本步发生交换处）
初始	$[7,3,8,5,2,6,9,4,1]$
$i=4$	$5 \geq \{4,1\}$，不变
$i=3$	$8 \leftrightarrow 9$：$[7,3,\mathbf{9},5,2,6,\mathbf{8},4,1]$
$i=2$	$3 \leftrightarrow 5 \leftrightarrow 4$：$[7,\mathbf{5},9,\mathbf{4},2,6,8,\mathbf{3},1]$
$i=1$	$7 \leftrightarrow 9 \leftrightarrow 8$：$[\mathbf{9},5,\mathbf{8},4,2,6,\mathbf{7},3,1]$

得到大根堆 $[9,5,8,4,2,6,7,3,1]$：

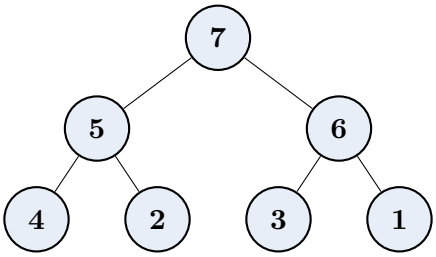


第 1 次取堆顶：堆顶 9 与末元素 1 交换，9 归位（有序区 $[9]$），堆规模减为 8，对新堆顶 1 筛下（$1 \leftrightarrow 8 \leftrightarrow 7$），得堆 $[8,5,7,4,2,6,1,3]$：



此趟排序结果（有序区在右端）：$\ldots \mid \mathbf{9}$

第 2 次取堆顶：堆顶 8 与当前末元素 3 交换，8 归位（有序区 $[8,9]$），堆规模减为 7，对新堆顶 3 筛下（$3 \leftrightarrow 7 \leftrightarrow 6$），得堆 $[7,5,6,4,2,3,1]$：



此趟排序结果（有序区在右端）：$\ldots \mid \mathbf{8\ 9}$

如此继续，每次把堆顶（当前最大值）换到末尾，可得**升序**序列。

时间复杂度：建堆为 $O(n)$；其后 $n-1$ 次“取堆顶 + 筛下调整”，每次筛下 $O(\log n)$，合计 $O(n\log n)$。故堆排序总体为

$$\boxed{O(n)+O(n\log n)=O(n\log n)}$$

最好、平均、最坏均为 $O(n\log n)$；空间 $O(1)$（原地排序），且为不稳定排序。