

一、选择题 (单选, 15 分)

1. 可以用 (C) 定义一个完整的数据结构。
A. 数据元素 B. 数据对象 C. 数据关系 D. 抽象数据类型
2. 在长度为 $n > 1$ 的带头节点的单链表 h 上, 另设有尾指针 r 指向尾节点, 则执行 (B) 操作与链表长度有关。
A. 删除单链表的第一个元素 B. 删除单链表的最后一个元素
C. 在单链表第一个元素前插入一个新元素 D. 在单链表的最后一个元素后插入一个新元素
3. 设线性表中有 $2n$ 个元素, (D) 在单链表上实现要比在向量 (顺序表) 上实现效率更高。
A. 删除所有值为 x 的元素
B. 在最后一个元素的后面插入一个新元素
C. 顺序输出前 k 个元素
D. 交换第 i 个元素和第 $2n-i-1$ 个元素的值 ($i=0, \dots, n-1$)
4. 元素 a, b, c, d, e 依次进入初始为空的栈, 若元素进栈后可停留、可出栈, 直到所有元素都出栈, 则在所有可能的出栈序列中, 以元素 d 开头的序列个数是 (B) d, a, b, c
A. 3 B. 4 C. 5 D. 6
5. 下列哪个应用使用到队列 (B) ~~BC~~
A. 括号匹配 B. 迷宫求解 C. 缓冲区 D. 进制转换 E. 递归
6. 表达式 $a*(b+c)-d$ 的后缀表达式为 (B)
A. $abcd*+-$ B. $abc*+d-$ C. $abc*+d-$ D. $-+*abcd$
7. 在一棵度为 4 的树 T 中, 若有 20 个度为 4 的节点, 10 个度为 3 的节点, 1 个度为 2 的节点, 10 个度为 1 的节点, 则树的叶节点个数为 (B)
A. 41 B. 82 C. 113 D. 122
8. 已知一棵二叉树的先序遍历为 $ABCDEF$, 中序遍历结果为 $CBAEDF$, 则后序遍历结果为 ()
A. $CBEFDA$ B. $FEDCBA$ C. $CBEFDA$ D. 不能确定
9. 含有 5 层节点的 AVL 树至少有多少个节点 (B)
A. 10 B. 12 C. 15 D. 17
10. 求解最短路径的 Floyd 算法的时间复杂度为 (C)
A. $O(n)$ B. $O(n^2)$ C. $O(n^3)$ D. $O(n^4)$
11. 根据 n 个元素建立一棵二叉搜索树时, 其时间复杂度大致为 (D)
A. $O(1)$ B. $O(\log_2 n)$ C. $O(n)$ D. $O(n \log_2 n)$
12. 快速排序算法在 (ABD) 情况下不利于发挥其长处
A. 要排序的数据量过大 C. 要排序的数据中含有多个相同值
B. 要排序的数据个数为奇数个 D. 要排序的数据已基本有序
13. 若序列的原始状态为 $\{1, 2, 3, 4, 5, 10, 6, 7, 8, 9\}$, 则要想使得排序过程中元素比较次数最少, 应该使用 (B) ~~BD~~ A
A. 插入排序 B. 选择排序 C. 希尔排序 D. 冒泡排序
14. 以下排序算法中, 时间复杂度为 $O(n \log_2 n)$ 且为稳定的排序算法是 (C)
A. 堆排序 B. 快速排序 C. 归并排序 D. 直接插入排序
15. 用邻接表表示的图进行广度优先遍历时, 通常采用 (B) 结构实现算法。
A. 栈 B. 队列 C. 二叉树 D. 图

二、填空题 (25 分)

1. 8 个节点 (节点关键码由 1,2,3,4,5,6,7,8 组成) 组成的二叉搜索树, 其可能的形态有 (4)⁸ 种, 当输入关键码序列按顺序为 5, 7, 6, 8, 3, 2, 1, 4 时 (只需填一种输入), 所生成的二叉搜索树为完全二叉树。 (4 分)
2. 对序列 {10, 70, 40, 50, 80, 60, 20, 30, 90} 进行堆排序, 首先进行堆构建 (大顶堆), 采用堆合并法 (Floyd 算法), 得到的堆序列为 10, 20, 30, 80, 50, 70, 40, 60, 90 (3 分), 该建堆步骤的时间复杂度为 $O(n \log n)$ (2 分)。输出两个最大关键码后, 剩余的堆序列为 20, 50, 10, 70, 40, 60, 90 (3 分), 不需要包含两个最大的关键码, 算法的整体复杂度为 $O(\log n)$ (2 分)。
3. 求最短路径的 Bellman&Ford 算法, 相比于 Dijkstra 算法, 其主要的改进在于输入的有向带权图的约束条件更加宽松。Dijkstra 不能处理 权重为负 的情况, 而 Bellman&Ford 算法可处理该情况, 但不能处理 负值回路 的情况。 (4 分)
- 以下为 Bellman&Ford 算法的部分核心代码, 请在空格处填写缺失的代码语句 (3 分), 代码倒数第 3、4、5 行的作用是 提前结束并打印检查结果和回路 (2 分), 该算法的时间复杂度为 $O(ne)$ (设顶点规模为 n , 边的规模为 e) (3 分)。

```
#define N 1010 // N 足够大
int nodenum, edgenum, original; // 点数目, 边数目, 起点
typedef struct Edge {
    int u, v; // u 为起点, v 为终点
    int cost; // 边的权重
} Edge;
```

```
Edge edge[N];
int dis[N], pre[N];
```

```
bool Bellman_Ford(){
    for (int i = 1; i <= nodenum; ++i) // 初始化各非起点节点的距离为无穷, 顶点从 1 计数
        dist[i] = (i == original ? 0 : MAX);
    for (int k = 1; k <= nodenum - 1; ++k) // 节点数量-1 次松弛迭代更新各点的最短距离
        for (int j = 1; j <= nodenum edgenum; ++j) //  $n^2$  规模的松弛更新转化为边的松弛, 降低复杂度
            if (dist[edge[j].v] > dist[edge[j].u] + edge[j].cost) {
                dist[edge[j].v] = dist[edge[j].u] + edge[j].cost
                pre[edge[j].v] = edge[j].u; // pre 记录当前最短距离的前一顶点, 见函数 print_path
            }
    bool flag = 1;
    for (int j = 1; j <= edgenum; ++j) // 边计数从 1 开始
        if (dis[edge[j].v] > dis[edge[j].u] + edge[j].cost) {
            flag = 0;
            break;
        }
    return flag;
}
```

```
void print_path (int root) { // 打印最短路的路径 (反向)
    while (root != pre[root]) { // 前驱
        printf("%d-->", root);
        root = pre[root];
    }
    if (root == pre[root]) printf("%d\n", root);
}
```

三、代码填空题 (25 分)

1. 以下为将一个带头节点的单链表 L 进行递增排序(插入排序)的代码, 请完善。(9 分)

```
typedef struct{
    Elemtype data; //数据域
    struct LNode* next; // 指针域
} //LNode, *LinkList;

void Sort (LinkList* L) {
    LinkList *p = L->next, *pre;
    LinkList *r = p->next;          // r 保持*p 后继节点指针, 以保证不断链
    p->next = NULL;                 // 构造只含一个数据节点的有序表
    p = r;
    while((p != NULL)) {          // 保存*p 的后继节点指针
        r = p->next;
        pre = L;
        while (pre->next != NULL && pre->next->data < p->data) // 查找插入*p 的前驱节点*pre
            pre = pre->next; p = p->next
        pre->next = p;          p->next = pre->next
        pre->next = p;
        p = r;                      // 扫描原单链表中剩余节点
    }
}
```

2. 请填补以下快速排序代码 (9 分)

快速排序基本思想: 取待排序元素序列中的第一个元素作为基准, 将整个元素序列划分为左右两个子序列, 左侧子序列中所有元素都小于或等于基准元素, 右侧子序列中所有元素的排序码都大于基准元素的排序码, 基准元素置于两个子序列中间, 分别对这两个子序列重复施行上述方法, 直到所有的元素都排在相应位置上为止。

void quickSort(int data[], int left, int right)

```
{
    if (left < right)
    {
        int i = left, j = right, x = data[left];
        while (i < j)
        {
            while (i < j && data[j] >= x)
                j--;
            if (i < j)
                x = data[j]; data[j] = data[left]; data[left] = data[j];
            while (i < j && data[i] <= x)
                i++;
            if (i < j)
                data[j--] = data[i];
        }
        data[i] = x;
        quicksort(data, left, i-1);
        quicksort(data, i+1, right);
    }
}
```


3. 请补充以下实现二叉搜索树的插入代码。(7 分)

```
struct BSTNode{
    int data;
    BSTNode* left;
    BSTNode* right;
```

```
};
```

```
bool Insert(int x, BSTNode *&p) {
```

```
    if (p==NULL){ // 实际节点插入
        p = new BSTNode;
        p->data = x;
        p->next = NULL;
    }
```

(以上代码段请填写，语句数目不限，3 分)

```
    else if (x < p->data) // 递归左孩子插入
```

```
        insert(x, p->left); (此处填入单一代码语句，2 分)
```

```
    else if (x > p->data)
```

```
        insert(x, p->right); (此处填入单一代码语句，2 分)
```

```
    else return false;
```

```
        // 已存在雷同节点，返回错误
```

```
};
```

```
int main()
```

```
{
    int x;
    std::cin >> x; // 首个插入元素
```

```
    BSTNode* root = NULL;
```

```
    while (x > 0) { // 输入元素为正整数则插入 BST
```

```
        Insert(root, x);
```

```
        std::cin >> x; // 不断等待新插入元素
```

```
    }
```

```
    return 0;
```

```
}
```

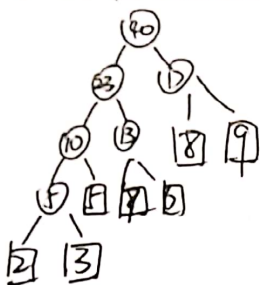
1. 一个算法所需的时间复杂度由下述递归方程表示, 试求出该算法的时间复杂度级别 (以下请给出推导过程)。(4 分) 式中, n 是问题的规模, 为简单起见, n 为 2 的整数幂。

$$T(n) = \begin{cases} 1, & \text{若 } n = 1 \\ 2T\left(\frac{n}{2}\right) + n, & \text{若 } n > 1 \end{cases}$$

$$T(n) = 2\left(2T\left(\frac{n}{4}\right) + \frac{n}{2}\right) + n = \dots = 2^{\log_2 n} T\left(\frac{n}{2^{\log_2 n}}\right) + n \log_2 n$$

$$= n + n \log_2 n = n \log_2 n.$$

2. 设给定权集 $w = \{5, 7, 2, 3, 6, 8, 9\}$, 画出构造 W 的哈夫曼编码树 (只需画出最终形态), 并求其加权路径长度 WPL。(4 分) (叶节点用方框画, 内部节点用圆圈画)



$$WPL = (2+3) \times 4 + (5+7+6) \times 3 + (8+9) \times 2 = 108$$

3. 以下第一排为文本串 T , 第二排为模式串 P , 请填写以下 NextValue 表, 以及填写整个 KMP 算法 (改进版本) 的匹配过程, 并完成相应的 KMP 代码填空。(8 分, 其中代码部分为 4 分)

b		a		b		a		b	
-1		0		-1		0		-1	

a	b	a	a	b	a	b	c	b	a	b	a	a	b	a	b	a	b	b	c	a
b _x	a	b	a	b																
	b	a	b _x	a	b															
			b	a	b	a _x	b													
						b _x	a	b	a	b										
							b	a	b	a	b _x									
												b	a	b	a	b				

代码填空:

```
int match ( char* P, char* T ) { //KMP 算法
```

```
    int* next = buildNext ( P );
```

```
    int n = (int) strlen ( T ), i = 0;
```

```
    int m = (int) strlen ( P ), j = 0;
```

```
    while ( j < m && i < n )
```

```
        if ( 0 > j || T[i] == P[j] )
```

```
            { i++; j++; }
```

```
        else
```

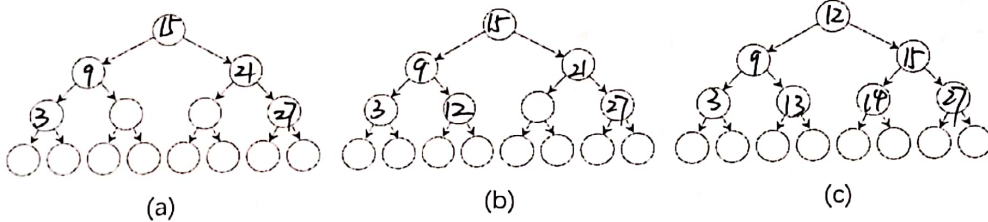
```
            j = next[j]
```

```
    delete [] next;
```

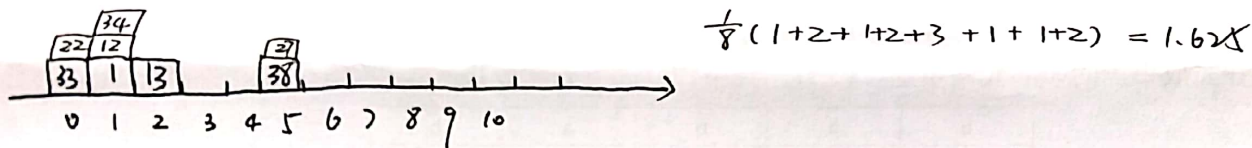
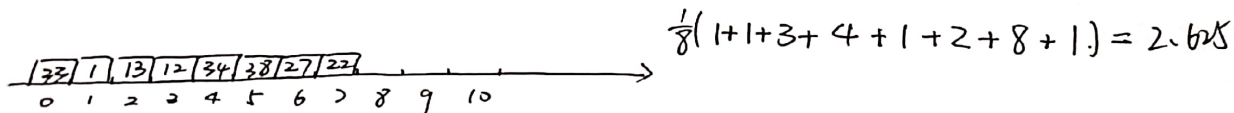
```
    return i - j;
```

```
}
```

4. 按顺序从空树插入序列{3,9,15,21,27}之后的 AVL 树的形式填入图中(a), 再插入元素{12}后的 AVL 树形式填入图中(b), 再插入元素{14,13}, 删除元素{21}后的 AVL 树填入图中(c)。(6分)



5. 使用散列函数 $H(\text{key}) = \text{key} \% 11$, 把一个整数值转化为散列表下标, 现要把数据{1,13,12,34,38,33,27,22}依次插入散列表中。请回答 1) 画出使用线性试探法来构造的散列表 (2分); 2) 画出使用独立链地址法构造的散列表 (2分); 3) 分别给出查找成功所需的平均查找长度 (3分)。



6. 如下图表示一个地区的通讯网, 边表示城市间的通讯线路, 边上的权重表示架设线路花费的代价, 如何选择能沟通每个城市且总代价最省的 $n-1$ 条线路, 画出所有可能的选择。(6分)

