2023.9.9

作业：算法分析题1 中 1-3、1-4、1-6、1-7

1-3

① 排序结果如下：$2 < \log n < 20n < n^{3/2} < 4n^2 < 3^n$

② $n!$ 排在最高位

1-4

解：(1) 设第一台计算机时间为 t_1，规模为 n_1

有 $t_1 = 3 \times 2^{n_1}$

有 $n_1 = \log_2 \frac{t_1}{3}$

因为第二台计算机运行速度为64倍

即相同计算量，用时为 $\frac{t_1}{64}$

代入得，

$$n_2 = \log_2 \frac{t_2}{3} = \log_2 \frac{64t_1}{3} = n_1 + 6$$

t秒内，能解决 n_1+6 规模

(2) 解题步骤如上，

得，$8T = 8n$

(3) 因为 $T(n) = 8$，即无论规模多大，计算时间恒定，

即计算量恒定，

得：任意值。

1-b

(1) $f(n)=\log n^2=2\log n$ $g(n)=\log n+5$

定义可知：① $\exists C=2$，$\exists N_0=1$，当 $n\geq N_0$，$f(n)\leq Cg(n)$

② $\exists C=1$，$\exists N_0=10^5$，当 $n\geq N_0$，$f(n)\geq Cg(n)$

所以 $f(n)=\theta(g(n))$

(2) $f(n)=\log n^2=2\log n$ $g(n)=\sqrt{n}=n^{\frac{1}{2}}$

定义可知：$\exists C=1$，$\exists N_0=10^5$，当 $n\geq N_0$

$f(n)\leq Cg(n)$

所以，$f(n)=O(g(n))$

(3) $f(n)=n$ $g(n)=\log^2 n$

定义可知：$\exists C=1$，$\exists N_0=10^5$，当 $n\geq N_0$

有 $f(n)\geq Cg(n)$

证明如下：$n\geq\log^2 n$

即证：$\sqrt{n}-\log n\geq 0$ 当 $n>N_0$

设 $t(n)=\sqrt{n}-\log n$

$t'(n)=\frac{1}{2\sqrt{n}}-\frac{1}{n}=\frac{\sqrt{n}-2}{2n}$

当 $n>N_0$，$t'(n)>0$，$t(n)$ 单增

$t(N_0)>0$ 有 $t(n)>0$ 当 $n>N_0$

所以 $n\geq\log^2 n$

$f(n)=\Omega(g(n))$

(4) $f(n) = n\log n + n \quad g(n) = \log n$

可知：$n = \Omega(\log n)$

所以，$f(n) = \Omega(g(n))$

(5) $f(n) = 10 \quad g(n) = \log 10$

均为常数

$f(n) = \theta(g(n))$

(6) $f(n) = \log^2 n \quad g(n) = \log n$

由定义可知：$\exists C=1$，$\exists N_0 = 10^6$，$n \geq N_0$，有

$f(n) \geq Cg(n)$

$\therefore f(n) = \Omega(g(n))$

(7) $f(n) = 2^n \quad g(n) = 100n^2$

由定义可知：$\exists C=1$，$\exists N_0 = 10^3$，$n \geq N_0$，有

$f(n) \geq Cg(n)$

$\therefore f(n) = \Omega(g(n))$

(8) $f(n) = 2^n \quad g(n) = 3^n$

由定义可知：$\exists C=1$，$\exists N_0 = 10^2$，$n \geq N_0$，有

$f(n) \leq Cg(n)$

$\therefore f(n) = O(g(n))$

1-7

证明：$n! = o(n^n)$

定义：$\forall \varepsilon > 0$，$\exists N_0$，当 $n > N_0$，$\frac{n!}{n^n} < \varepsilon$，即证 $\lim_{n\to\infty} \frac{n!}{n^n} = 0$

$0 \le \frac{n!}{n^n} = \frac{1}{n} \cdot \frac{2}{n} \cdot \frac{3}{n} \cdots \frac{n}{n} \le \frac{1}{n}$　已知 $\lim_{n\to\infty} \frac{1}{n} = 0$

由夹点定理得：$\lim_{n\to\infty} \frac{n!}{n^n} = 0$

证毕.

2023.9.16

作业：算法分析题2中 2-2、2-3、2-4、2-6、2-7、2-10

2-2

(1) 错误，理由如下：

算法的边界判断为 "left <= right"

但转化下标时，即拆分子问题时，

"right = middle"，这是错误的，会导致死循环，

应改为 "right = middle-1"

(2) 错误，理由如下：

会陷入死循环，即 x = a[n-1]

(3) 错误，理由如下：

存在错误返回，当 x = a[n-1]，应返回 n-1，

该算法返回 -1

(4) 错误，理由如下：

边界为 "left < right"

而转化下标时：right = middle-1

其中 middle = (left+right)/2

会出现 right = left 陷入死循环

(5) 正确。

(6) 错误，理由如下：

有 "left = middle - 1" 导致错找正确结果

(7)错误，理由如下：

middle = (left + right + 1) / 2

right = middle

会导致死循环。

2-3

JAVA语言：

```
public static int[] binarySearch(int[] a, int x, int n){
    if( x > a[n-1]){
        return [n-1, -1];{
    else if( x < a[0]){
        return [-1, 0];{
    else{
        int left = 0;
        int right = n-1;
        while( left <= right){
            int middle = (left + right) /2;
            if( x == middle) return [middle, middle];
            else if (x < middle) right = middle-1;
            else left = middle+1; {
```

return [right, left];

return [-1, -1];

2-4

解：①　u有m位，定义为 $u = A$

v有n位，拆分为 $v = B \cdot 2^{\frac{n}{2}} + C$

$u \cdot v = AB \cdot 2^{\frac{n}{2}} + AC$

设 $T(m,n)$ 为 m位u与n位v相乘所需运算总数

$T(m,n) = 2T(m,\frac{n}{2}) + O(n)$

习一直递归至 m与n规模相近，利用本章介绍的分治法，需要 $O(m^{\log 3})$

递归树如下：

$T(m,n)$ ··· $O(n)$

$2T(m,\frac{n}{2})$ ··· $O(n)$

$4T(m,\frac{n}{4})$ ··· $O(n)$

共有 $\log n$ 层

$\log n\, T(m,m) = \log n\, O(m^{\log 3})$

解得：$T(n) = O(\log n \cdot m^{\log 3}) + O(n \log n) = O((n + m^{\log 3}) \log n)$

②　由①可知：一开始将n位拆分成 $\frac{n}{m}$ 份，一份m位

有　$T(m,n) = \frac{n}{m} T(m,m) + O(\frac{n}{m})$

$T(m,m)$ 耗时为 $O(m^{\log 3})$

一共耗时 $O(\frac{n}{m}\cdot m^{\log 3})+O(\frac{n}{m})=O(nm^{\log\frac{3}{2}})$

2-6

解：算法步骤如下：

① 将 n 阶矩阵分成 $m\times m$ 个子矩阵，转化为如下乘法：

$$\begin{bmatrix} A_{11}\cdots A_{1m} \\ \vdots \\ A_{m1}\cdots A_{mm} \end{bmatrix}\cdot\begin{bmatrix} B_{11}\cdots B_{1m} \\ \vdots \\ B_{m1}\cdots B_{mm} \end{bmatrix}=\begin{bmatrix} C_{11}\cdots C_{1m} \\ \vdots \\ C_{m1}\cdots C_{mm} \end{bmatrix}$$

其中 $C_{ij}=\sum_{p=1}^{m} A_{ip}\cdot B_{pj}$

② A、B 矩阵均为 $2^k\times 2^k$ 个元素的矩阵，

因此 $A\cdot B$ 可以用 strassen 算法。

时间复杂度分析：

① $n=m2^k$ 拆分：$O(\log n)$

② $A\cdot B$：$O(2^{k\log 7})=O(7^k)$

③ 共有 m^3 个 $A\cdot B$ 运算：$m^3O(7^k)=O(m^37^k)$

综上：$T(n)=O(m^3\cdot 2^{k\log 7})+O(\log n)=O(m^37^k)$

2-7

解：① 分析：$P(n_1)=a_0+a_1n_1+a_2n_1^2+\cdots+n_1^d=0$

$$\vdots$$

$$P(n_d)=a_0+a_1n_d+a_2n_d^2+\cdots+n_d^d=0$$

化为矩阵：

$$\begin{bmatrix} -n_1^d \\ \vdots \\ -n_d^d \end{bmatrix} = \begin{bmatrix} 1 & n_1 & \cdots & n_1^{d-1} \\ \vdots & & & \vdots \\ 1 & n_d & \cdots & n_d^{d-1} \end{bmatrix} \cdot \begin{bmatrix} a_0 \\ \vdots \\ a_{d-1} \end{bmatrix}$$

记作：$N = BA$

线性代数可知：

$$A = B^{-1} \cdot N$$

关键在于对 B 求逆，由线性代数可知：

一般而言对 $n \times n$ 矩阵求逆时间复杂度为 $O(n^3)$

所以，$T(n) = O(n^3)$

② 利用分治法：

$$P(x) = \prod_{i=1}^{d}(x - n_i) = \prod_{i=1}^{\frac{d}{2}}(x - n_i) \prod_{i=\frac{d}{2}+1}^{d}(x - n_i) = P_1(x) P_2(x)$$

即 d 次多项式转化为两个 d/2 次多项式乘积

由题可知：

$$T(n) = \begin{cases} T(1) & n = 1 \\ 2T(\frac{n}{2}) + O(n\log n) & n \geq 2 \end{cases}$$

递归树：

$n\log n$

$2 \cdot \frac{n}{2}\log\frac{n}{2}$

$2 \cdot \frac{n}{4}\log\frac{n}{4}$

$\log n$ 层

$2^{\log n} O(1) = O(n)$

得：$T(n) = O(n\log^2 n) + O(n) = O(n\log^2 n)$

2-10

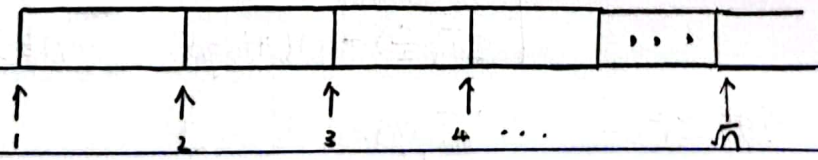
解：算法 JAVA 代码如下：

```
public void mergesort(int[] a, int[] b, int left, int right){
    if (left < right){
        int pivot = (int) sqrt(right - left + 1);
        if (pivot > 1){
            for(int i = 0; i < pivot, i++){
                mergesort(a, b, left + i*pivot, left + (i+1)*
pivot - 1);{
            mergesort(a, b, left + pivot*pivot, right){
        merge(a, b, left, right);{{
```

（其中 b 为辅助数组，merge 为合并算法）

merge 为合并 $\sqrt{n}$ 个已排好序的数组，在最坏情况下，merge 需要 $O(n\log n)$ 时间，下面进行证明。

证明：如图：



分为 $\sqrt{n}$ 块数组和 $\sqrt{n}$ 个指针

合并时，需选出 $\sqrt{n}$ 个指针所指数中最小的一个，

又因为 $\sqrt{n}$ 个数为无序，建立最小堆时间 $O(\sqrt{n}\log\sqrt{n})$

若最坏情况，会"均匀"找到最小数，但可以利用堆结构不断入堆出堆，入/出堆花费 $O(\log\sqrt{n})$

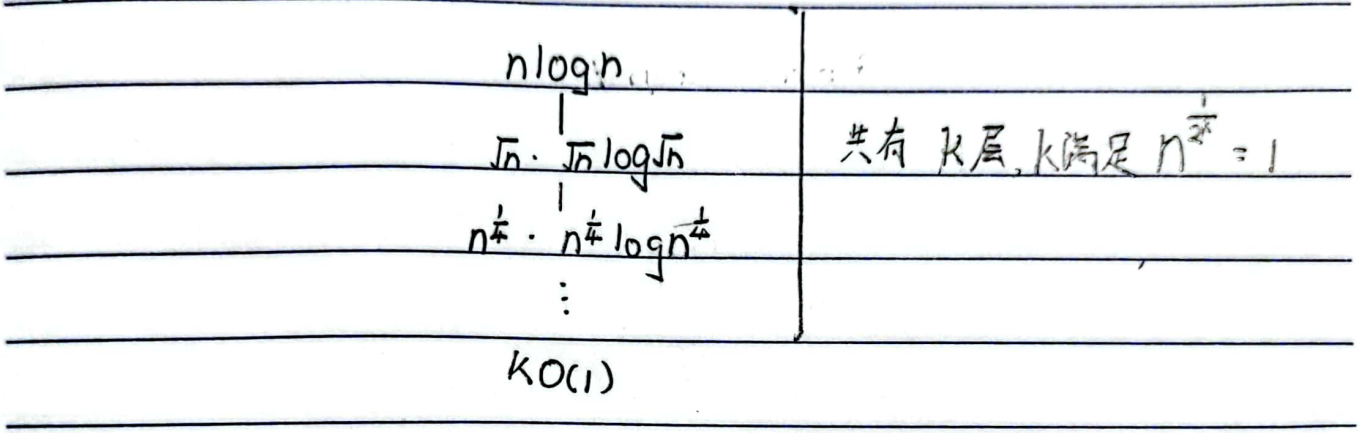
入队/出队操作为 n 次，至此时间复杂度为 $O(n\log\sqrt{n})$

所以最坏情况下，时间复杂度为 $O(n\log n)$

计算时间 $T(n)$ 满足：

$$T(n)=\begin{cases}O(1) & n\le 1\\ \sqrt{n}\,T(\sqrt{n})+O(n\log n) & n>1\end{cases}$$

递归树如下：



理论上有无数层才能减至 1，但是编程时只需保证小于一定值即可，又由于指数增长速度远大于 $n\log n$，可以将 K 视作常数

则，$T(n)=O(n\log n)$