No:
Date:

1-3: $2,\ \log n,\ n^{\frac{2}{3}},\ 20n,\ 4n^2,\ 3^n,\ n!$

1-4: (1) 设第一台、第二台计算机 t 秒内可解决输入规模为 n_1、n_2 的问题

$$\frac{3\times 2^{n_1}}{3\times 2^{n_2}} = \frac{1}{64}$$

$$n_2 = n_1 + 6$$

(2) $$\frac{n_1^2}{n_2^2} = \frac{1}{64}$$

$$n_2 = 8n_1$$

(3) 由于 $T(n)=8$，8为常数，所以计算机可处理任意规模的问题。

1-7: 由 Stirling's approximation

$$n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n\left(1+\Theta\left(\frac{1}{n}\right)\right)$$

$$n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n \cdot e^{\alpha_n} \quad \left(\frac{1}{12n+1} < \alpha_n < \frac{1}{12n}\right)$$

$$\lim_{n\to\infty}\frac{n!}{n^n} = \lim_{n\to\infty}\frac{\sqrt{2\pi n}\left(\frac{n}{e}\right)^n\cdot e^{\alpha_n}}{n^n} = \lim_{n\to\infty}\sqrt{2\pi n}\left(\frac{1}{e}\right)^n\cdot e^{\alpha_n} = 0$$

$\therefore n! = o(n^n)$.

No:
Date:

1-8: $$T_{avg}(n) = \sum_{size(I)=n} p(I)\,T(I) = \Theta(f(n))$$

即 $\exists c_1, c_2 > 0, n_0$ 使 $\forall n > n_0$ 有：

$$c_1 f(n) \le \sum_{size(I)=n} p(I)\,T(I) \le c_2 f(n)$$

于是有 $\exists c > 0, n_0 > 0$，使 $\forall n > n_0$，有

$$0 \le c f(n) \le \sum_{size(I)=n} p(I)\,T(I)$$

又 $\because \sum_{size(I)=n} p(I)\,T(I) \le T_{max}(n)$

即 $0 \le c f(n) \le T_{max}(n)$

故 $T_{max}(n) = \Omega(f(n))$

1-11: $3n^2+10n = O(n^2)$

$\frac{n^2}{2} + 2^n = O(2^n)$

$21 + \frac{1}{n} = O(1)$

$\log n^3 = O(\log n)$

$10\log 3^n = O(n)$

1-12: 由 O 定义知 $O(1) = O(2)$

二者表达同一函数时，常数不同.

No:
Date:

2-3
```
template <class Type>
int binarySearch(Type a[], const Type& x, int left,
    int right, int &i, int &j){
 int middle;
 while(left <= right){
     middle = (left + right)/2;
     if(x == a[middle];
     return i;
     i=j=middle;
     return 1;
     }
     if(x > a[middle])
         left = middle+1;
     else
         right = middle-1;
     }
    i=right;
    j=left;
    return 0;
}
```

No:
Date:

2-4. 将 n 分成 $\frac{n}{m}$ 段，每段 m 位。
计算一次 m 位乘法，耗时 $O(m^{\log 3})$
(采用分治法)
计算结果为各结果拼接的结果
因此此，耗时为 $O(\frac{n}{m} \cdot m^{\log 3})$，即 $O(nm^{\log(\frac{3}{2})})$

2-6.
将 n 阶矩阵分块为 $m \times m$ 的矩阵。用传统方法求两个 m 阶矩阵的乘积需要计算 $O(m^3)$ 次 2 个 $2^k \times 2^k$ 矩阵的乘积。用 Strassen 矩阵乘法计算两个 $2^k \times 2^k$ 矩阵的乘积需要的计算时间为 $O(7^k)$，因此算法的计算时间为 $O(7^k m^3)$

2-7 $P(x) = \prod_{i=1}^{d}(x-n_i) = \prod_{i=1}^{d/2}(x-n_i)\prod_{i=d/2+1}^{d}(x-n_i)$

$= P_1(x)P_2(x)$

d 次多项式转化为两个 $\frac{d}{2}$ 次多项式的乘积.
计所需时间为 $T(d)$.

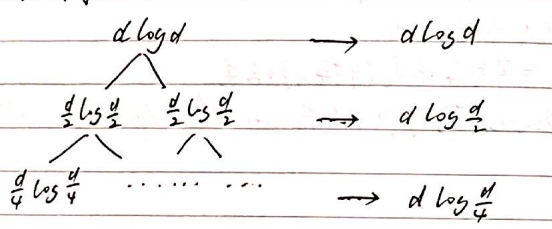
$$T(d) = \begin{cases} O(1) & d=1 \\ 2T(d/2) + O(d\log d) \end{cases}$$

$2T(d/2)$ 为两个 $\frac{d}{2}$ 次多项式乘积的计算时间

No:
Date:

$O(d\log d)$ 为两个 $\frac{d}{2}$ 次多项式的乘积所需时间

画出递归树：



树长为 $\log d$

$$T(d) = d\log d + d\log\frac{d}{2} + \cdots + d\log\frac{d}{2^{\log d-1}} + O(d)$$

$$= d\left(\log\frac{d^{\log d}}{2^{(\log d(\log d-1))/2}}\right) + O(d)$$

$$= d\cdot\log d\cdot\left(\log\frac{d}{2^{\frac{\log d-1}{2}}}\right) + O(d)$$

$$= d\cdot\log d\cdot\left(\log d - \frac{\log d-1}{2}(\log 2)\right) + O(d)$$

$$= O(d\log^2 d)$$

2-13 将算法 Partition 的不等号反向

```
template<class Type>
int Partition(Type a[], int p, int r){
    int i=p, j=r+1;
    Type x=a[p];
    while(true){
        while(a[++i] > x && i<r);
        while(a[--j] < x);
        if(i>=j)
            break;
        Swap(a[i], a[j]);
    Swap(a[j], a[p]);
    return j;
}
```