Lecture 01:

1) Assignment - 01

$$T(n)=\begin{cases}\Theta(1), & \text{if } n=1,\\ aT(n/b)+f(n), & \text{if } n>1.\end{cases}$$

where $a \geq 1$, $b > 1$ are constants and f is nonnegative. Then

1. If $f(n)=O(n^{\log_b a-\varepsilon})$ for some constant $\varepsilon>0$, then $T(n)=\Theta(n^{\log_b a})$.
2. If $f(n)=\Theta(n^{\log_b a})$, then $T(n)=\Theta(n^{\log_b a}\lg n)$, where $\lg n$ stands for $\log_2 n$.
3. If $f(n)=\Omega(n^{\log_b a+\varepsilon})$ for some constant $\varepsilon>0$, and if $af(n/b)\leq cf(n)$ for some constant $c<1$ and all sufficiently large n, then $T(n)=\Theta(f(n))$.

直观上看就是比较 $f(n)$ 与 $n^{\log_b a}$ 的阶数, 其中大的决定了 $T(n)$ 的阶数; 如果阶数相同则乘 $\lg n$.

证明: 先对 n 为 b^k, $k\in\mathbb{N}$ 的情形证明, 下面的渐进符号都是对 n 在 b^k 上的点而言的. 写出递归树, 高度为 $\log_b n$, 故有 $a^{\log_b n}=n^{\log_b a}$ 个叶, 从而

$$T(n)=\Theta(n^{\log_b a})+g(n),$$

其中

$$g(n)=\sum_{j=0}^{\log_b n-1} a^j f(n/b^j).$$

Case 1. $f(n)=O(n^{\log_b a-\varepsilon})$, 易得 $g(n)=O(n^{\log_b a})$.

Case 2. $f(n)=\Theta(n^{\log_b a})$, 易得 $g(n)=\Theta(n^{\log_b a}\log_b n)=\Theta(n^{\log_b a}\lg n)$.

Case 3. 首先 $g(n)=\Omega(f(n))$. 又 $af(n/b)\leq cf(n)$ for some constant $c<1$ and sufficiently large n. 即 $a^j f(n/b^j)\leq c^j f(n)$. 得

$$\begin{aligned}g(n)&=\sum_{j=0}^{\log_b n-1} a^j f(n/b^j)\\ &\leq \sum_{j=0}^{\log_b n-1} c^j f(n)+O(1)\\ &\leq f(n)\sum_{j=0}^{\infty} c^j+O(1)\\ &=O(f(n)).\end{aligned}$$

故 $g(n)=\Theta(f(n))$.

2） Assignment – 02

该题直观的解决方法便是 Brute Force(暴力求解)。时间复杂度为 O（n^2）

Brute Force Algorithm:

```
    minDist = infinity
    for i = 1 to length(P) - 1
        for j = i + 1 to length(P)
            let p = P[i], q = P[j]
            if dist(p, q) < minDist:
                minDist = dist(p, q)
                closestPair = (p, q)
    return closestPair
```

利用分治思想进行求解。首先分析题目，符合分治法的适用条件，规模越小容易求解，同时具有最优子结构。

- 分解
  - 对所有的点按照x坐标（或者y）从小到大排序（排序方法时间复杂度$O(nlogn)$）。
  - 根据下标进行分割，使得点集分为两个集合。
- 解决
  - 递归的寻找两个集合中的最近点对。
  - 取两个集合最近点对中的最小值$min(dis_{left}, \ dis_{right})$。
- 合并
  - 最近距离不一定存在于两个集合中，可能一个点在集合A，一个点在集合B，而这两点间距离小于dis。

在分解和合并时，可能存在按照x轴、y轴进行排序的预处理$O(nlogn)$，该问题在解决阶段只做提取的操作为$\Theta(n)$，递推式为：

$$T(n) = \begin{cases} 1 & n <= 3 \\ 2T(\frac{n}{2}) + O(n) & n > 3 \end{cases}$$

计算后得到整体时间复杂度为：$O(nlogn)$

这其中如何合并是关键。根据递归的方法可以计算出划分的两个子集中所有点对的最小距离$dis_{left}, \ dis_{right}$，再比较两者取最小值，即$dis = min(dis_{left}, \ dis_{right})$。那么一个点在集合A，一个在集合B中的情况，可以针对此情况，用之前分解的标准值，即按照x坐标（或者y）从小到大排序后的中间点的x坐标作为mid，划分一个$[mid - dis, mid + dis]$区域，如果存在最小距离点对，必定存在这个区域中。

之后只需要根据$[mid - dis, mid]$左边区域的点来遍历右边区域$[mid, mid + dis]$的点，即可找到是否存在小于dis距离的点对。

```
struct point {
    double x;
    double y;
    point(double x, double y) :x(x), y(y) {}
    point() { return; }
};

bool cmp_x(const point &A, const point &B)  // 比较 x 坐标
{
    return A.x < B.x;
}

bool cmp_y(const point &A, const point &B)  // 比较 y 坐标
{
    return A.y < B.y;
}

double distance(const point & A, const point & B)
{
    return sqrt(pow(A.x - B.x, 2) + pow(A.y - B.y, 2));
}
/*
* function: 合并，同第三区域最近点距离比较
* param: points 点的集合
*         dis 左右两边集合的最近点距离
*         mid x 坐标排序后，点集合中中间点的索引值
*/
double merge(vector<point> & points, double dis, int mid)
{
    vector<point> left, right;
    for (int i = 0; i < points.size(); ++i)  // 搜集左右两边符合条件的点
    {
        if (points[i].x - points[mid].x <= 0 && points[i].x - points[mid].x > -dis)
            left.push_back(points[i]);
        else if (points[i].x - points[mid].x > 0 && points[i].x - points[mid].x < dis)
            right.push_back(points[i]);
    }
    sort(right.begin(), right.end(), cmp_y);
    for (int i = 0, index; i < left.size(); ++i)  // 遍历左边点集，与右边符合条件的计算距离
    {
        for (index = 0; index < right.size() && left[i].y < right[index].y; ++index);
        for (int j = 0; j < 7 && index + j < right.size(); ++j)  // 遍历右边 6 个点
        {
            if (distance(left[i], right[j + index]) < dis)
```

```
                dis = distance(left[i], right[j + index]);
            }
        }
        return dis;
    }


    double closest(vector<point> & points)
    {
        if (points.size() == 2) return distance(points[0], points[1]);   // 两个点
        if (points.size() == 3)
            return min(distance(points[0], points[1]), min(distance(points[0], points[2]),
            distance(points[1], points[2])));   // 三个点
        int mid = (points.size() >> 1) - 1;
        double d1, d2, d;
        vector<point> left(mid + 1), right(points.size() - mid - 1);
        copy(points.begin(), points.begin() + mid + 1, left.begin());   // 左边区域点集合
        copy(points.begin() + mid + 1, points.end(), right.begin());   // 右边区域点集合
        d1 = closest(left);
        d2 = closest(right);
        d = min(d1, d2);
        return merge(points, d, mid);
    }

    int main()
    {
        int count;
        printf("点个数：");
        scanf("%d", &count);
        vector<point> points;
        double x, y;
        for (int i = 0; i < count; ++i)
        {
            printf("第%d 个点", i);
            scanf("%lf%lf", &x, &y);
            point p(x, y);
            points.push_back(p);
        }
        sort(points.begin(), points.end(), cmp_x);
        printf("最近点对值：%lf", closest(points));
        return 0;
    }
```

Lecture 02:

Assignment – 01

令 $y = \lceil \frac{n}{k} \rceil$, $x = \lfloor \frac{n-1}{k} \rfloor$

y*k >= n, (y - 1)*k < n,   (x+1)*k > n – 1, x*k <= n – 1

因(y-1)*k 是整数，故(y – 1)*k <= n-1

y*k > x*k,   (y-1)*k < (x+1)*k

y > x,   y < x + 2

因为 y 是整数

故 y = x + 1

Assignment – 02

```python
def binary_cmp(a, b):
    a = str(a)
    b = str(b)
    if len(a) > len(b):
        print("a is bigger than b")
        return 1
    elif len(a) < len(b):
        print("a is smaller than b")
        return -1
    else:
        while True:
            if len(a) < 1:
                print("a is equal to b")
                return 0
            if a[0] == b[0]:
                a = a[1:]
                b = b[1:]
            elif a[0] > b[0]:
                print("a is bigger than b")
                return 1
            else:
                print("a is smaller than b")
                return -1
```

Time Complexity: O(n)   Space Complexity: O(len(a)+len(b))

Assignment – 03

100 的阶乘有 24 个结尾 0。

具体算法如下：

一、首先确定 5 因子有多少：

在 100 内，因子是 5 的数有 5, 10, 15, 20, 25… 总共有 20 个。但是 25, 50, 75, 100 都包含了 2 个 5 作为因子（25=5*5, 50=2*5*5），对于这些数，需要多数一次。所以总共有 24 个 5 因子。

从公式角度： 5 因子的数目 = 100/5 + 100/(5^2) + 100/(5^3) + … = 24

二、确定 2 的因子有多少：

2, 4, 6, 8, 10, … 总共有 100/2=50 个 2 因子，100/4=25 个 4 因子（要多计数一次），100/8=12 个 8 因子（要多计数一次）所以 2 因子的数目 = 100/2 + 100/(2^2) + 100/(2^3) + 100/(2^4) + 100/(2^5) + 100/(2^6) + 100/(2^7) + … = 97 综上所述，共有 24 个 5 因子 和 97 个 2 因子，所以能凑 24 个 (2,5) 对。

综上所述 100 的阶乘也就有 24 个结尾零。

Assignment – 04

证明如下,假如它是有理数,则可以写成 m/n 的形式,其中 m 和 n 均为整数且互素,那么以二为底三的对数=ln3/ln2=m/n,从而 2^m=3^n 而这是不可能的,因为 2^m 一定是偶数,而 3^n 一定是奇数,他们不可能相等,从而结论得证.

Assignment – 05

用反证法可以证明如果 2 的 n 次方减 1 是质数,则 n 必是质数.

假设 n 不是质数,则必存在大于 1 的数 a,b,有 n=ab,于是

2^n-1=2^(ab)-1=(2^a-1)(2^(a-1)+2^(a-2)b+...+2^(b-1))

这与 2^n-1 是质数矛盾

Assignment – 06

Ramsey Theorem：对于一个给定的两个整数 m,n≥2，则一定存在一个最小整数 r，使得用两种颜色（红色或蓝色）无论给K_r的每条边如何染色，总能找到一个红色的K_m或者蓝色的K_n。

一种更为人熟知的解释是友谊定理（Friendship Theorem）：在至少 6 人中，或者有 3 人，他们互相认识；或者有 3 人，他们两两互相不认识。友谊定理描述的就是问题中的情况。

证明$K_6 \to K_3, K_3$:

1） 从任意一点（此处以为示范）可以引 5 条线，由鸽巢原理(抽屉原理)可知，至少有 3 条边会被着同色。我们假定这三条边都着红色，而蓝色的情况类似于此。
2） 考虑此三边的终点的着色情况：我们任意将此三点中的两点的连线着红色，即可构成一个红色的K_3。
3） 如果不用红色给N_3, N_4, N_5组成的边着色，我们则会得到一个蓝色的K_3。

Assignment – 07

Q-1:

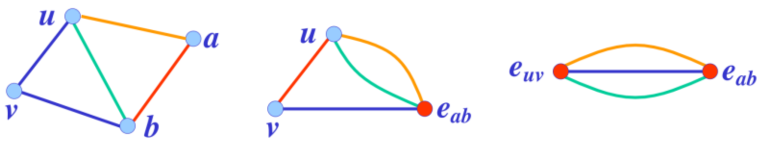
The size of the minimum cut of graph G is at most the mininal degree of vertices in G, and it is often referred to as the edge-connectivity of G since it is the minimal number of edges be removed from G to make it disconnected

Q-2:

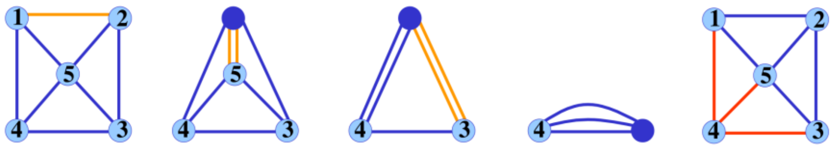
Suppose G has only one minimum cut – if it actually has more than one, just pick your favorite – and this cut has size k. Every vertex of G must lie on at least k edges; otherwise, we could separate that vertex from the rest of the graph with an even smaller cut. Thus, the number of incident vertex-edge pairs is at least kn. Since every edge is incident to exactly two vertices, G must have at least kn/2 edges.

Assignment – 08

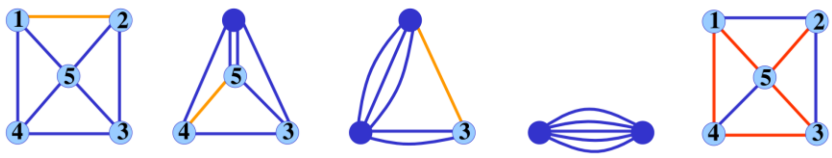
The algorithm is simply as follows: Pick an edge (a, b) from $E(G)$ uniformly at random, and then obtain a new graph $G/(a, b)$ by contracting edge (a, b). If there are several edges between some pairs of (newly formed) vertices, retain them all. Edges between vertices that are merged are removed, so that there will never be any self-loops. The contraction process is repeated with $G/(a, b)$ until only two vertices remain in $G/(a, b)$; at this point, the set of edges between these two vertices is a cut in G (here we may need to trace back the corresponding edges in the original graph).



Whether the returned cut is a minimum or not may depend on the choices of contraction of edges. The returned cut in the following figure is a minimum.



The returned cut in the following is not a minimum, it, however, is maximal (it is not a cut anymore if any edge is removed).



Lecture 03

Assignment – 01

Q – 1:

```
CONVERT(φ):    // returns a CNF formula equivalent to φ
// Any syntactically valid propositional formula φ must fall into exactly one of the
// following 7 cases (that is, it is an instance of one of the 7 subclasses of Formula)

If φ is a variable, then:
   return φ.
   // this is a CNF formula consisting of 1 clause that contains 1 literal

If φ has the form P ^ Q, then:
   CONVERT(P) must have the form P1 ^ P2 ^ ... ^ Pm, and
   CONVERT(Q) must have the form Q1 ^ Q2 ^ ... ^ Qn,
   where all the Pi and Qi are disjunctions of literals.
   So return P1 ^ P2 ^ ... ^ Pm ^ Q1 ^ Q2 ^ ... ^ Qn.

If φ has the form P v Q, then:
   CONVERT(P) must have the form P1 ^ P2 ^ ... ^ Pm, and
   CONVERT(Q) must have the form Q1 ^ Q2 ^ ... ^ Qn,
   where all the Pi and Qi are disjunctions of literals.
   So we need a CNF formula equivalent to
      (P1 ^ P2 ^ ... ^ Pm) v (Q1 ^ Q2 ^ ... ^ Qn).
   So return (P1 v Q1) ^ (P1 v Q2) ^ ... ^ (P1 v Qn)
           ^ (P2 v Q1) ^ (P2 v Q2) ^ ... ^ (P2 v Qn)
             ...
           ^ (Pm v Q1) ^ (Pm v Q2) ^ ... ^ (Pm v Qn)

If φ has the form ~(...), then:
  If φ has the form ~A for some variable A, then return φ.
  If φ has the form ~(~P), then return CONVERT(P).   // double negation
  If φ has the form ~(P ^ Q), then
     return CONVERT(~P v ~Q).   // de Morgan's Law
  If φ has the form ~(P v Q), then
     return CONVERT(~P ^ ~Q).   // de Morgan's Law

If φ has the form P -> Q, then:
  Return CONVERT(~P v Q).   // equivalent

If φ has the form P <-> Q, then:
  Return CONVERT((P ^ Q) v (~P ^ ~Q)).

If φ has the form P xor Q, then:
```

Return CONVERT((P ^ ~Q) v (~P ^ Q)).

This is a recursive algorithm, and it can be proved that it won’t recurse forever.

Take the CONVERT routine above, and replace each "and" operator with "or" and vice-versa, a routine for converting to DNF instead of to CNF is get.

Q – 2:

使用反证法：假设可满足的布尔表达式的否定是永真的。

因可满足的布尔表达式表示在所有变元的取值当中，总有一组取值使得该布尔表达式为真，取其否定后，总有一组取值使该布尔表达式为假，与其否定是永真的矛盾，得证。反之亦然，证毕。

Lecture 04

Assignment – 01

$P(divisible) = \frac{p+q-2+1}{n}$ => $P(undivisible) = 1 - \frac{p+q-1}{n}$

Assignment – 02

$GCD(m,n)$ means greatest common divisor of m and n

$P(divisible) = \frac{m+n-2+1-n(shared\ numbers)}{mn} = \frac{m+n-1-(GCD(mn)-1)}{mn}$ =>

$P(undivisible) = 1 - \frac{m+n-GCD(mn)}{mn}$