Homework #1
Solution Manual

1 Manhaha Problem (programming assignment)

```
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>
#define N 1000010
struct Point { int x,y,id; } p[N],q[N];
int dis[N],who[N];
int cmp_x( const void *a, const void *b ) {
        return ((struct Point*)a)->x - ((struct Point*)b)->x;
}
void calculate( int cx, int cy, int s1, int e1, int s2, int e2 ) {
        int d1=s1<e1?1:-1,d2=s2<e2?1:-1,big=INT_MIN,who_big=-1;
        while ( s1!=e1 && cy*p[s1].y<cy*p[s2].y ) s1+=d1;
        while ( s1!=e1 ) {
                while ( s2!=e2 && cy*p[s2].y<=cy*p[s1].y ) {
                        int now=cx*p[s2].x+cy*p[s2].y;
                        if ( now>big || now==big && p[s2].id<who_big ) {
                                big=now;
                                who_big=p[s2].id;
                        }
                        s2+=d2;
                }
                int me=p[s1].id,now=cx*p[s1].x+cy*p[s1].y-big;
                if ( now<dis[me] || now==dis[me] && who_big<who[me] ) {
                        dis[me]=now;
                        who[me]=who_big;
                }
                s1+=d1;
        }
}
void merge( int l, int m, int r ) {
        int i=l,j=m,k=l;
        while ( i<m || j<r ) q[k++]=p[j==r||i<m&&p[i].y<p[j].y?i++:j++];
        for ( i=l; i<r; i++ ) p[i]=q[i];
}
void solve( int l, int r ) {
        if ( r-l==1 ) return;
        int m=(l+r)/2;
        solve(l,m);
        solve(m,r);
        calculate(-1,+1,l,m,m,r);
        calculate(+1,+1,m,r,l,m);
        calculate(-1,-1,m-1,l-1,r-1,m-1);
        calculate(+1,-1,r-1,m-1,m-1,l-1);
        merge(l,m,r);
}
int main()
{
```

```
48          int n,i;
49          scanf("%d",&n);
50          for ( i=0; i<n; i++ ) p[i].id=i;
51          for ( i=0; i<n; i++ ) dis[i]=INT_MAX;
52          for ( i=0; i<n; i++ ) scanf("%d%d",&p[i].x,&p[i].y);
53          qsort(p,n,sizeof(struct Point),cmp_x);
54          solve(0,n);
55          for ( i=0; i<n; i++ ) printf("%d␣%d\n",who[i],dis[i]);
56          return 0;
57  }
```

The key point is the function calculate. The Manhattan distance of two points $(x_1, y_1), (x_2, y_2)$ is defined as $|x_1-x_2|+|y_1-y_2|$. But if we constraint that $x_1 \geq x_2$ and $y_1 \geq y_2$, the distance will becomes to $x_1 - x_2 + y_1 - y_2 = (x_1 + y_1) - (x_2 + y_2)$. This equation shows that the distance to (x_1, y_1) is smaller as (x_2+y_2) larger, so we can simply sort points in both sides by y-coordinate, and then solve this problem in linear time under the constraint $x_1 \geq x_2$ and $y_1 \geq y_2$. However, we can still solve this problem without this constraint since the remaining is just the symmetry cases of this! By including merge sort on y-coordinate when doing divide and conquer instead of re-sorting every time, the total computation time is $T(n) = 2T(n/2) + O(n) = O(n \lg n)$.

2 Super Safe

Note that we can undo a move by press/release same button again, since the constraint that determine whether we can press/release button k or not is only related to button $1, 2, \ldots, k-1$.

1) Let s_i be the strategy to open type-i safe.
   i) If $n > 2$, do s_{n-2} to release button $1, 2, \ldots, n-2$.
   ii) Release the button n.
   iii) If $n > 2$, undo s_{n-2} to press button $1, 2, \ldots, n-2$.
   iv) If $n > 1$, do s_{n-1} to release button $1, 2, \ldots, n-1$.

2) According to our strategy, we have

$$\begin{aligned} a_1 &= 1 \\ a_2 &= 2 \\ a_n &= a_{n-2} + 1 + a_{n-2} + a_{n-1} = a_{n-1} + 2a_{n-2} + 1 \qquad \forall n > 2 \end{aligned}$$

3) We claim that the general form of a_n is $\frac{2^{n+2}-3-(-1)^n}{6}$.

Proof. We use mathematical induction to show that.

Basis:

$$\begin{aligned} \forall n = 1 \qquad & \frac{2^{n+2}-3-(-1)^n}{6} = \frac{2^3-3-(-1)}{6} = 1 = a_1 \\ \forall n = 2 \qquad & \frac{2^{n+2}-3-(-1)^n}{6} = \frac{2^4-3-(+1)}{6} = 2 = a_2 \end{aligned}$$

Induction:

$$\begin{aligned} a_n &= a_{n-1} + 2a_{n-2} + 1 && \forall n \geq 3 \\ &= \frac{2^{n+1} - 3 - (-1)^{n-1}}{6} + 2 \cdot \frac{2^n - 3 - (-1)^{n-2}}{6} + 1 \\ &= \frac{2^{n+1} - 3 - (-1)^{n-1}}{6} + \frac{2^{n+1} - 6 - 2 \cdot (-1)^{n-2}}{6} + \frac{6}{6} \\ &= \frac{2^{n+2} - 3 - (-1)^{n-1} - 2 \cdot (-1)^{n-2}}{6} \\ &= \frac{2^{n+2} - 3 - [(-1)^{n-1} + (-1)^{n-2}] - (-1)^{n-2}}{6} \\ &= \frac{2^{n+2} - 3 - (-1)^n}{6} \end{aligned}$$

□

4) *Proof.* We use mathematical induction for each stage to show that.

Basis:
When $n \leq 2$, $a_1 = 1, a_2 = 2$ is obviously optimal.

Induction:
$\forall n \geq 3$

i) `OOO...OOO` → `XXX...XOO`
We need to release the button n at some time, and the state which can release the button n is unique. Thus we use s_{n-2} to achieve this state, since s_{n-2} is optimal by induction hypothesis.

ii) `XXX...XOO` → `XXX...XOX`
Simply release the button n is obviously optimal.

iii) `XXX...XOX` → `OOO...OOX`
We have at exactly two choices for each state in this stage. These choices are press/release button 1 or $k+1$ where k is the first pressed button, and one of them will be the undo move, which lead to the previous state. But the optimal strategy should not enter same state more than once, so for each step the optimal way is uniquely determined. Undo s_{n-2} would not enter same state more than once by induction hypothesis, thus this stage is optimal.

iv) `OOO...OOX` → `XXX...XXX`
What should we do now is same as opening type-$(n-1)$ safe, thus simply use s_{n-2} is optimal. □

5) What we need is same as the stage iii and iv when opening type-$(n+1)$ safe, without the

last button. It takes

$$\begin{aligned} a_{n-1} + a_n &= \frac{2^{n+1} - 3 - (-1)^{n-1}}{6} + \frac{2^{n+2} - 3 - (-1)^n}{6} \\ &= \frac{2^{n+1} - 3 - (-1)^{n-1} + 2^{n+2} - 3 - (-1)^n}{6} \\ &= \frac{2^{n+1} + 2^{n+2} - 3 - 3 - (-1)^{n-1} - (-1)^n}{6} \\ &= \frac{2 \cdot 2^n + 4 \cdot 2^n - 6 - [(-1)^{n-1} + (-1)^n]}{6} \\ &= \frac{6 \cdot 2^n}{6} - 1 \\ &= 2^n - 1 \end{aligned}$$

steps to open the safe. Thus peter can always use this strategy to release the first pressed button until open the safe successfully regardless of the states of the buttons at the beginning. Another solution is using the fact that the total number of states in type-n safe is 2^n, so this strategy will visit **every** state before the safe be opened.

3 Read Your Textbook

1) Since $n \lg n = O\left(n^{\log_3 4 - \epsilon}\right), \forall 0 < \epsilon \leq log_3 4 - 1 \approx 0.26$, by case 1 of master theorem, we have $T(n) = \Theta\left(n^{\log_3 4}\right)$.

2) Note that the n-th harmonic number $H_n = \sum_{i=1}^{n} \frac{1}{i} = \Theta(\lg n)$. Let $f(n) = T(n)/n$, we have

$$\begin{aligned} f(n) &= T(n)/n \\ &= \left[3T(n/3) + n/\lg n\right]/n \\ &= 3T(n/3)/n + 1/\lg n \\ &= T(n/3)/(n/3) + 1/\lg n \\ &= f(n/3) + 1/\lg n \\ &= f(n/9) + 1/\lg(n/3) + 1/\lg n \\ &\vdots \\ &= \Theta\left(\sum_{i=0}^{\lfloor \log_3 n \rfloor - 1} \frac{1}{\lg(n/3^i)}\right) \\ &= \Theta\left(\sum_{i=0}^{\lfloor \log_3 n \rfloor - 1} \frac{1}{\log_3(n/3^i)}\right) \\ &= \Theta\left(\sum_{i=0}^{\lfloor \log_3 n \rfloor - 1} \frac{1}{\log_3 n - i}\right) \\ &= \Theta\left(\sum_{i=1}^{\lfloor \log_3 n \rfloor} \frac{1}{i}\right) \\ &= \Theta\left(\lg \lg n\right) \end{aligned}$$

Thus $T(n) = nf(n) = \Theta\left(n \lg \lg n\right)$.

3) We use the substitution method to show that $T(n) = O(n) = cn$ for some constant $c > 0$.

$$\begin{aligned} T(n) &= T(n/2) + T(n/4) + T(n/8) + n \\ &\leq cn/2 + cn/4 + cn/8 + n \\ &= 7cn/8 + n \\ &\leq cn \quad \forall c \geq 8 \end{aligned}$$

And $T(n) = T(n/2) + T(n/4) + T(n/8) + n \geq n = \Omega(n)$, so $T(n) = \Theta(n)$.

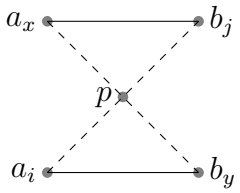
4) Let $f(n) = T(n)/n$, we have

$$\begin{aligned} f(n) &= T(n)/n \\ &= \left[\sqrt{n}T(\sqrt{n}) + n\right]/n \\ &= T\left(\sqrt{n}\right)/\sqrt{n} + 1 \\ &= f\left(\sqrt{n}\right) + 1 \\ &= f\left(\sqrt{2^{\lg n}}\right) + 1 \\ &= f\left(2^{\frac{\lg n}{2}}\right) + 1 \\ &= f\left(2^{\frac{\lg n}{4}}\right) + 1 + 1 \\ &\vdots \\ &= \Theta\left(\lg\lg n\right) \end{aligned}$$

Thus $T(n) = nf(n) = \Theta\left(n \lg\lg n\right)$.

4 Dividing and Matching by Segments

1) Let point p be the intersection of $\overline{a_i b_j}$ and $\overline{a_x b_y}$:



By triangle inequality, we have

$$\begin{aligned} & \overline{a_i b_y} < \overline{a_i p} + \overline{p b_y} \\ \text{and} \quad & \overline{a_x b_j} < \overline{a_x p} + \overline{p b_j} \\ \Longrightarrow \quad & \overline{a_i b_y} + \overline{a_x b_j} < \overline{a_i p} + \overline{p b_y} + \overline{a_x p} + \overline{p b_j} \\ \Longrightarrow \quad & \overline{a_i b_y} + \overline{a_x b_j} < \overline{a_i p} + \overline{p b_j} + \overline{a_x p} + \overline{p b_y} \\ \Longrightarrow \quad & \overline{a_i b_y} + \overline{a_x b_j} < \overline{a_i b_j} + \overline{a_x b_y} \end{aligned}$$

Thus the total length will be strict smaller after each iteration.

With the fact that the total number of methods to connect these points is finite, this algorithm will terminate eventually, so for any given $2n$ points, a solution always exists.

2) Take the lowest point(if more than one, take any) as the center, and sort the remaining points by degree respect to it. Because no three points are collinear, this order is unique.

Without loss of generality, assume that the lowest point is a b_n. We claim that there is a amber point a_m such that if we connect b_n to a_m, the remaining points will be divided into two half plane, and each half plane contains same number of amber/blue points. Thus we can recursively solve these two half plane by same algorithm, since any segment in one half plane would not intersect with another segment in other half plane.

Note that the amber point a_m is equivalent to the definition of lovely number in problem 5, if we define the relation $<$ on points by the degree respect to b_n. Thus we can do sorting points and finding a_m in $O(n \lg n)$ time with the same algorithm in problem 5. Since for each time we connect one pair of points, by repeating at most n times, the total time complexity is $O(n^2 \lg n) = O(n^3)$.

5 A Lovely Problem (Uh, probably not)

1) *Proof.* Let $f(x) = |\{a_i \mid a_i < x\}| - |\{b_i \mid b_i < x\}|$, then $f(a_k) = 0$ iff a_k is a lovely number, and we have $f(x) \in \mathbb{Z}, f(a_1) \leq 0, f(a_{n+1}) \geq 0, f(a_{i+1}) \leq f(a_i) + 1$, We can show that $\exists a_k \colon f(a_k) = 0$ on above condition by mathematical induction.

Basis:
When $n = 0$, $f(a_1) \leq 0, f(a_{n+1}) = f(a_1) \geq 0 \implies f(a_1) = 0$.

Induction:
If $f(a_1) = 0$, just take $k = 1$ is enough.
If $f(a_1) < 0$, by $f(a_{i+1}) \leq f(a_i) + 1$, we have $a_2 \leq 0$, and just apply induction hypothesis on $a_2, a_3, \ldots, a_{n+1}$ is enough. □

2) **Require:** A and B are sorted

```
1: function LUCKY1(n, A, B)
2:     f ← 0
3:     j ← 1
4:     for i ← 1 to n + 1 do
5:         while j ≤ n and b_j < a_i do
6:             f ← f − 1
7:             j ← j + 1
8:         if f = 0 then
9:             return i
10:        f ← f + 1
```

This algorithm runs in $O(n)$ time since the variable i and j are strictly increasing.

3) **Require:** A is sorted

```
1: function LUCKY2(n, A, B)
2:     f ← 0
3:     l ← 1
4:     r ← n + 1
5:     loop
6:         m ← ⌈(l + r)/2⌉
7:         B′ ← {b_i | b_i ∈ B, b_i < a_m}
8:         f′ ← f + (m − l) − |B′|
9:         if f′ > 0 then
10:            r ← m − 1
```

```
11:             B ← B′
12:         else if f′ < 0 then
13:             f ← f′
14:             l ← m
15:             B ← B \ B′
16:         else
17:             return m
```

Note that we use $m \leftarrow \lceil (l+r)/2 \rceil$ to avoid infinite loop maybe caused by $m \leftarrow \lfloor (l+r)/2 \rfloor$, since if $l+1=r \implies \lfloor (l+r)/2 \rfloor = l$, $l \leftarrow m$ would change nothing. It is easy to see that $f \leq 0$, and we claim that $|B| \leq f+r-l$ always be true in above algorithm. thus the total computation cost T is

$$\begin{aligned} T &= O(\sum_{\text{loop}} |B|) \\ &= O(\sum_{\text{loop}} f+r-l) \\ &= O(\sum_{\text{loop}} r-l) \\ &= O(\sum_{i=1}^{\lg n} \frac{n}{2^i}) \\ &= O(n) \end{aligned}$$

proof of $|B| \leq f+r-l$. We use mathematical induction on loop to show that.

Basis:
The initial state is $|B| = n = 0 + (n+1) - 1 = f + r - l$

Induction:

$$\begin{aligned} &\forall f' > 0 \\ &|B'| = f - f' + (m-l) \\ &\quad < f + m - l \\ &\quad \leq f + (m-1) - l \end{aligned}$$

$$\begin{aligned} &\forall f' < 0 \\ &|B \setminus B'| = |B| - |B'| \\ &\quad \leq (f+r-l) - (f - f' + m - l) \\ &\quad = f + r - l - f + f' - m + l \\ &\quad = f' + r - m \end{aligned}$$

If $f = 0$, this algorithm will terminate immediately. □

4)
```
1: function LUCKY3(n, A, B)
2:     f ← 0
3:     l ← 1
4:     r ← n + 1
5:     loop
6:         m ← ⌈(l + r)/2⌉
7:         Run selection algorithm such that a_l, a_{l+1}, ..., a_{m-1} < a_m < a_{m+1}, a_{m+2}, ..., a_r
8:         B′ ← {b_i | b_i ∈ B, b_i < a_m}
9:         f′ ← f + (m − l) − |B′|
10:        if f′ > 0 then
11:            r ← m − 1
12:            B ← B′
13:        else if f′ < 0 then
```

14: $f \leftarrow f'$
15: $l \leftarrow m$
16: $B \leftarrow B \setminus B'$
17: **else**
18: **return** m

This algorithm just take another $O(r-l)$ for each iteration due to the linear time selection algorithm, thus the total computation time is same as the previous problem, $O(n)$.