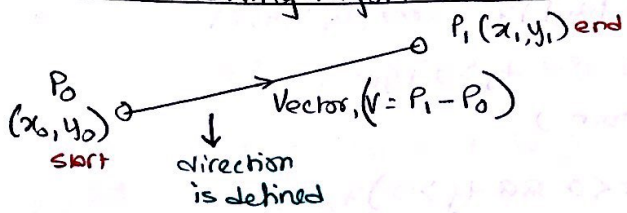
{LINE DRAWING}

①

line drawing Algorithm



$dx = x_1 - x_0$
$dy = y_1 - y_0$
} carries direction and magnitude.

$dx \Rightarrow i|dx|$
$dy \Rightarrow j|dy|$
+ve & -ve defines the direction.

Code {D,D,A}

line drawing algo;

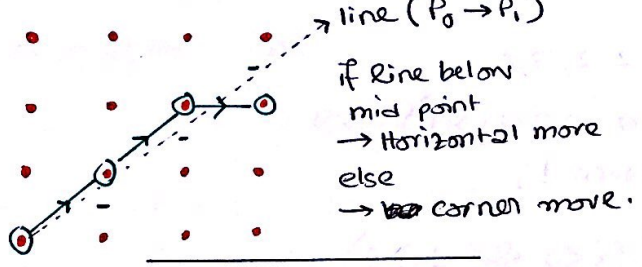
```
y = y0
for (x = x0 to x <= x1) {
    draw Pixel (x,y)
    y += m;
}
```

We know,

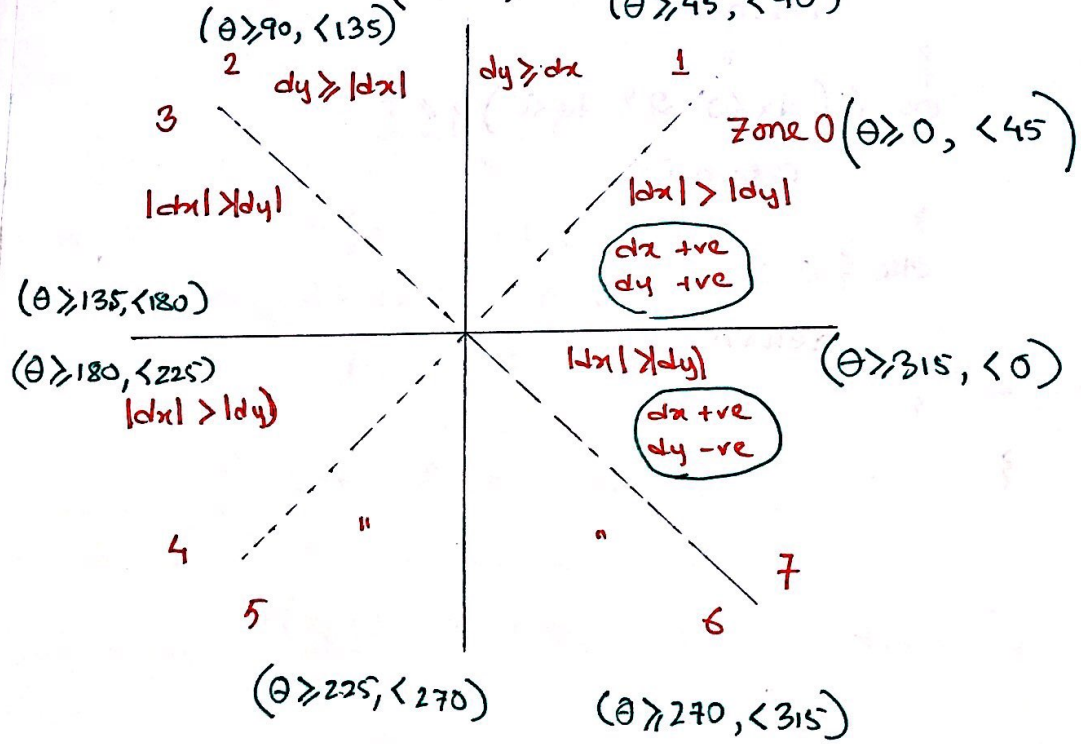
$$m = \frac{y_1 - y_0}{(x_1 - x_0)}$$

- when step size of x becomes 1
- $m = y_1 - y_0$
- Hence increment of $y = m$

IDEA (Bresenham's Algorithm)



(Zones)



- absolute values check then determine the sets of zones
- then check dy, and dx's solid ± values to determine which zone.

Code : (zone)

```
int find_slope (int x0, int y0, int x1, int y1) {
int dx = x1 - x0, dy = y1 - y0;
if (abs|dx| > abs|dy|) { // zone {0,3,4,7}
    if (dx > 0 && dy > 0) { // 0
        return 0;
    }
    else if (dx < 0 && dy > 0) { // 3
        return 3;
    }
    else if (dx < 0 , && dy < 0) { // 4
        return 4;
    }
    else { // 7
        return 7;
    }
}
else { // zone 1,2,5,6
    if (dx > 0 && dy > 0) { // 1
        return 1;
    }
    else if (dx < 0 && dy > 0) { // 2
        return 2;
    }
    else if (dx < 0 && dy < 0) { // 5
        return 5;
    }
    else { // 6
        return 6;
    }
}
}
```

drawing the line:

Eqn of line:

$$y = mx + c; \qquad m = \frac{dy}{dx}$$

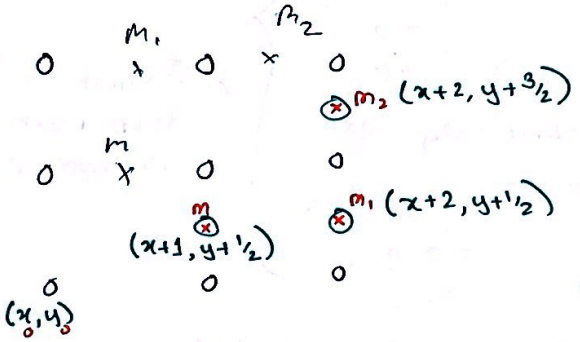
Multiply with dx (both side):

$$dy\,x - dx\,y + dx\,c = 0$$

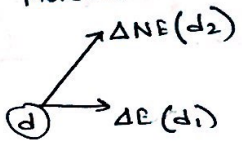
$$A = dy, \quad B = -dx$$

$$Ax + By + C = 0$$

Zone 0



movements



We need the deviation to form the whole line.

(deviation at m_1) $\rightarrow \Delta E$

at $M_1 \longrightarrow A(x+2) + B(y+1/2) + C = d_1$

at $M \longrightarrow -A(x+1) + B(y+1/2) + C = d$

$$A + 0 + 0 = d_1 - d \longrightarrow \Delta E$$ } Horizontal movement

$$\therefore \Delta E = A = dy$$

(deviation at m_2)

at m_2 $\quad A(x+2) + B(y+3/2) + C = d_2$

at m_1 $\quad -A(x+1) + B(y+1/2) + C = d$

$$A + B = d_2 - d \longrightarrow \Delta NE$$ } NE movement

$$\therefore \Delta NE = A + B = dy - dx$$

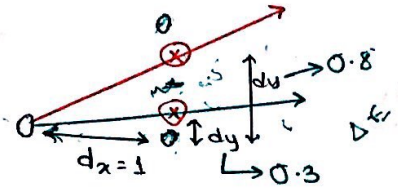
Dinit

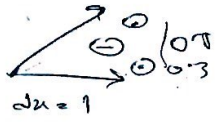
$$A(x_0+1) + B(y_0+1/2) + C = dinit$$

Since, $Ax_0 + By_0 + C = 0$ → line er upore pore.

$$\therefore dinit = A + B/2$$

$$dinit = dy - dx/2$$





(For Green line)

dinit = $dy - \frac{dx}{2}$

$= 0.3 - 0.5$

$= -0.2$

ΔE

(For Red line).

dinit = $dy - \frac{dx}{2}$

$= 0.8 - 0.5$

$= 0.3$

So,

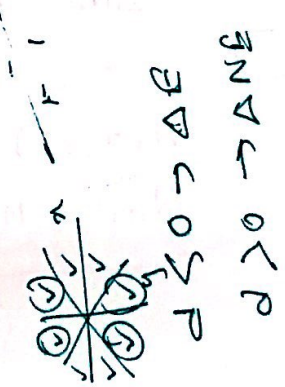
- if dinit is +ve then next pixel is upper pixel.
- if dinit is −ve then next pixel is lower pixel.

Code drawline

```
void drawline (int x0, int y0, int x1, int y1) {
    int dx = x1 - x0 , dy = y1 - y0;
    int d = 2dy - dx ; // mult by 2 to remove fraction.
    int dE = 2dy ; dNE = 2(dy - dx); // also mult by 2

    int y = y0 , x = x0;
    drawPixel (x, y); // loop er baire first pixel draw hobe.
    while (x < x1) {
        if (d < 0) { // Horizontal movement
                     so x++, nothing for y
                     and d += dE
            x++ ;
            d += dE ;
        }
        else { // dNE movement
            x++ ;
            y++ ;
            d += dNE ;
        }
        drawPixel (x, y);
    } // end while

} // end void
```



Question 1

(30, 50) to (40, 54) } So 10×10 pixel size.

```
void drawline_0 { int x0, y0, x1, y1 }

dx = 10 ;
dy = 4 ;
d = -2 ; // 2dy - dx
         = 8 - 10
         = -2
dE = +8 ; // 2dy = 8
dNE = -12 ; // 2(dy - dx) = -12
```

x	y	d	ΔE/ΔNE
30	50	-2	ΔE
31	50	6	ΔNE
32	51	-6	ΔE
33	51	2	ΔNE
34	52	-10	ΔE
35	52	-2	ΔE
36	52	6	ΔNE
37	53	-6	ΔE
38	53	2	ΔNE
39	54	-10	ΔE
40	54	→ end	

no redundant calculation.

Question No 2

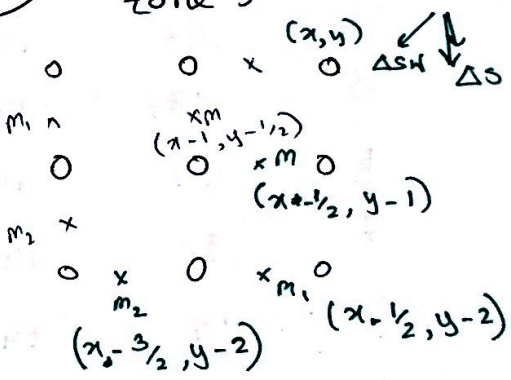
$(x_0, y_0) = (100, 150)$ to $(x_1, y_1) = (0, 0)$

dy = ~~+~~100 150 } Since abs|dx| < abs|dy| and (dx && dy) < 0

dx = -100

So zone 5,

Zone 5



ΔS movement

M_1 $A(x-1/2) + B(y-2) + C = d_1$

M $A(x-1/2) + B(y-1) + C = d$

$0 - B + 0 = d_1 - d \rightarrow \Delta S$

$\Delta S = -B = +dx$

ΔSH movement

M_2 $A(x-3/2) + B(y-2) + C = d_2$

M $A(x-1/2) + B(y-1) + C = d$

$-A - B + 0 = d_2 - d_1 \rightarrow \Delta SH$

$\Delta SH = (-B - A) = +dx - dy \rightarrow \Delta SH$

$d_{init} = A(x_0 + 1/2) + B(y_0 - 1) + C$ $\quad x(100, 150)$

$= -B - A/2$

$= dx - dy/2$

for, such Question

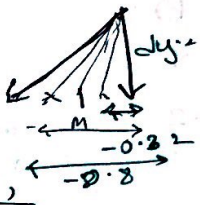
✓ $\Delta S = dx = -100$

✓ $\Delta SW = dx - dy = -100 - (-150)$

$= 50$

✓ $d_{init} = -100 - \left(\frac{-150}{2}\right)$

$= -25$



dy = -1

dx = -0.2 / -0.8
ΔS ΔSW

ΔS,

$= -0.2 - \left(\frac{-1}{2}\right)$ $\quad dx - \frac{dy}{2}$

$= 0.3$ (+ve) ✓ → ΔS

ΔSW →

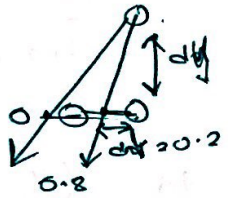
$= -0.8 + 1/2$

$= -0.3$ (-ve) → ΔSW

∴ $d > 0 \rightarrow \Delta S$ $\quad y--$

$d < 0 \rightarrow \Delta SW$ $\quad x--, y--$

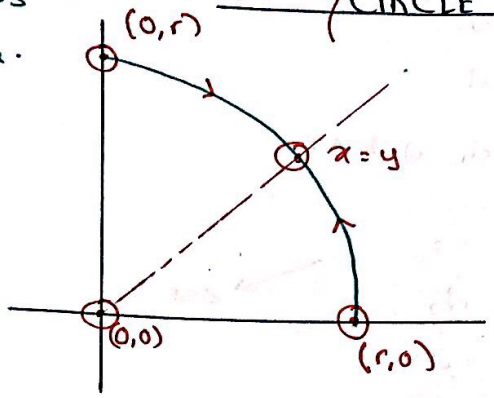
For 12 pixels



Pixel	x	y	d	ΔS / ΔSW
1	100	150	-25	ΔSW
2	99	149	25	ΔS
3	99	148	-75	ΔSW
4	98	147	-25	ΔSW
5	98	146	25	ΔS
6	97	145	-75	ΔSW
7	96	144	-25	ΔSW
8	95	143	25	ΔS
9	95	142	-75	ΔSW
10	94	141	-25	ΔSW
11	93	140	25	ΔS
12	93	139	-75	ΔSW.

CIRCLE DRAWING

r = radius of circle.



2 operations

① if starting from (0,r)

→ ΔE
→ ΔSE
} movements.

② if starting from (r,0)

ΔNW
ΔN
} movements.

(parametric equation of a circle)

$f(x,y) = x^2 + y^2 - r^2 = 0$ ---- (i)

① for (0,r) → start

(x,y)

m (x+1, y-1/2)

m₁ (x+2, y-1/2)

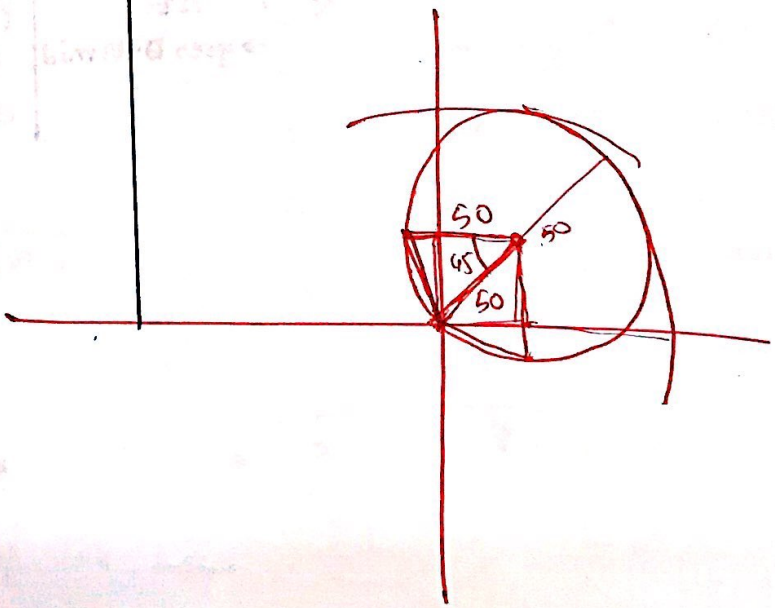
m₂ (x+2, y-3/2)

m (x+1, y-1/2)
m₁ (x+2, y-1/2)
m₂ (x+2, y-3/2)

② for (r,0) → start.

m₂ (x-3/2, y+2)
m₁ (x-1/2, y+2)
m (x-1/2, y+1)

m (x-1/2, y+1)
m₁ (x-1/2, y+2)
m₂ (x-3/2, y+2)



(For Operation 1) → (0, r)

$f(m_1) = (x+2)^2 + (y-1/2)^2 - r^2 = d_1$ → ΔE

$f(m) = (x+1)^2 + (y-1/2)^2 - r^2 = d$

$\Rightarrow 2x + 3 = d_1 - d = \Delta E$

$\therefore \Delta E = 2x+3$

$f(m_2) = (x+2)^2 + (y-3/2)^2 - r^2 = d_2$ → ΔSE

$f(m) = (x+1)^2 + (y-1/2)^2 - r^2 = d$

$(2x+3) + (-2y+2) = d_2 - d = \Delta SE$

$\Delta SE = 2x - 2y + 5$

$x_0^2 + y_0^2 - r^2 = 0$ on circle.

$dinit = (x_0+1)^2 + (y_0 - 1/2)^2 - r^2 = dinit$

$= (x_0)^2 + 2x_0 + 1 + (y_0)^2 - y_0 + 1/4 - r^2 = dinit$

$= 2x_0 - y_0 + 5/4$

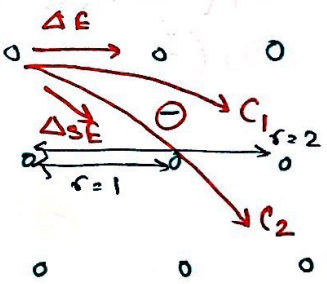
Since $x_0 = 0$, $y_0 = r$ } start (0, r)

$dinit = 5/4 - r$

Since, $dinit = 5/4 - r$

$= 1.25 - r$

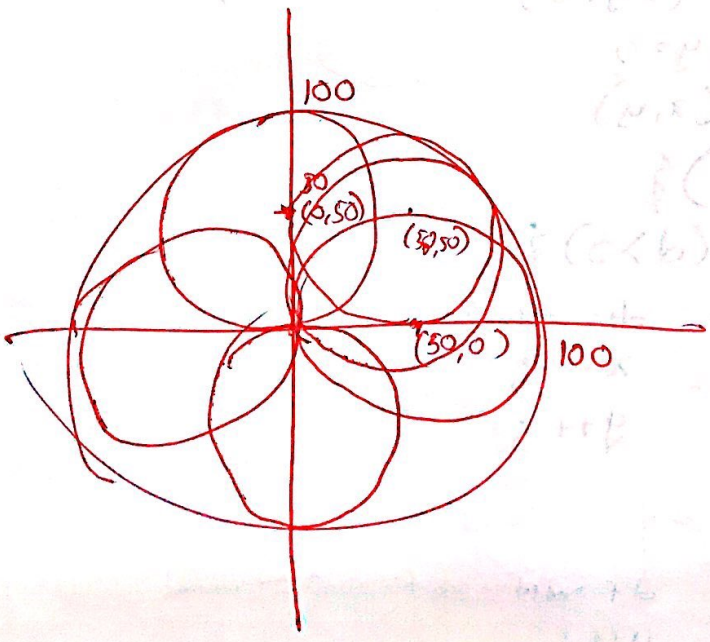
+ve: So, ΔSE → goes below mid	For r = 1 we get dinit +ve
-ve: ΔE → goes over mid	For r = 2 or greater dinit = -ve



Code: {zone 1 (start)}

```
Void drawCircle_1 (int r){
    int d = 5 - (4*r);  // mult 4
    int x = 0, y = r;   // start (0,r)
    int dE = 4*(2x + 3);  // mult 4
    int dSE = 4*(2x - 2y + 5);  // mult 4
    draw8way(x,y);
    while (x < y) {
        if (d<0){ // ΔE
            d+ = dE;   } → we cale d first
            x++;
        }
        else{  // ΔSE
            d+ = dSE;
            x++;
            y--;
        }
        draw8way(x,y)
    } //endwhile
} //end
```

Example : (0,10) {r = 10}



x	y	d	ΔE/ΔSE
0	10	-35	ΔE
1	10	-23	ΔE
2	10	-3	ΔE
3	10	25	ΔSE
4	9	-11	ΔE
5	9	33	ΔSE
6	8	21	ΔSE
→ 7	7		

FOR CIRCLE zone 0 (r, 0)

NW N ①

$f(m_1) = (x - 1/2)^2 + (y+1)^2 - r^2 = d_1$

$f(m) = (x - 1/2)^2 + (y+1)^2 - r^2 = d$

$0 + (2y + 3) + 0 = d_1 - d \rightarrow$ AN

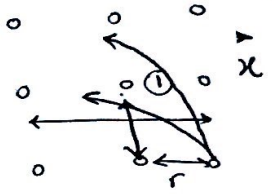
$\Delta N = 2y + 3$

$f(m_2) = (x - 3/2)^2 + (y+2)^2 - r^2 = d_2$

$f(m) = (x - 1/2)^2 + (y+1)^2 - r^2 = d$

$(-2x + 2) + (2y + 3) + 0 = d_2 - d \rightarrow \Delta NW$

$\Delta NW = 2y - 2x + 5$



$d_{init} = (x_0 - 1/2)^2 + (y_0 + 1)^2 - r^2 = d_2 - d$

$d_{init} = 5/4 - r$

Δ & NW when $r \leq 1$

So $d > 0$ {~~ANW~~}

So $d < 0$ {ΔN}

Code

```
void drawCircle_0 (int r) {
    int d = 5 - (4*r)
    int dNW = 4*(2y - 2x + 5)
    int dN = 4*(2y + 3)
    int x = r, y = 0
    draw8way (x, y)
    while (x > y) {
        if (d > 0) {
            d += dNW
            x--;
            y++;
        }
        else {
            d += dN
            y++;
        }
        draw8way (x, y)
    }
}
```

(20,0) r=20

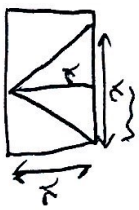
Question

$\Delta N = 2y+3$

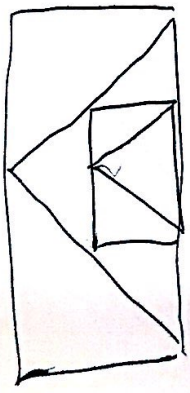
$\Delta NW = 2y+2x+5$

dinit = -18.75

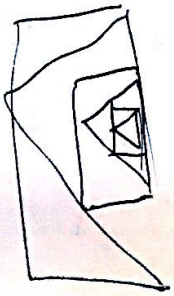
Pixel	x	y	d	ΔN / ΔNW
1	20	0	-18.75	ΔN
2	20	1	-15.75	ΔN
3	20	2	-10.75	ΔN
4	20	3	-3.75	ΔN
5	20	4	5.25	ΔNW
6	19	5	-21.75	ΔN
7	19	6	-8.75	ΔN
8	19	7	6.25	ΔNW
9	18	8	-12.75	ΔN
10	18	9	6.25	ΔNW
11	17	10	-6.75	ΔN
12	17	11	16.25	ΔNW



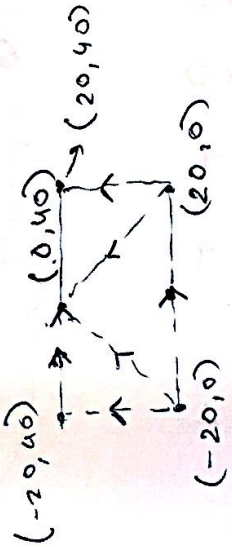
i=1



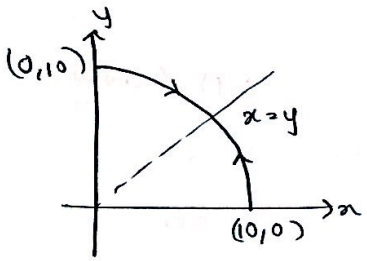
i=2

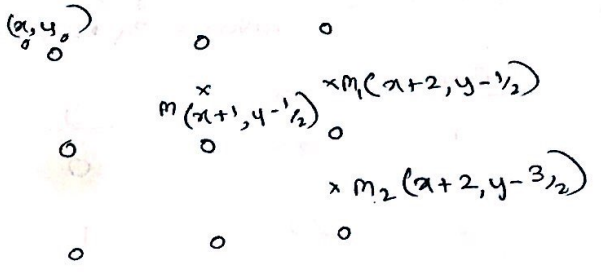


i=3



start → (0, 10)





$f(x,y) = x^2 + y^2 - r^2 = 0$

$f(m_1) = (x+2)^2 + (y-1/2)^2 - r^2 = d_1$

$- f(m) = -(x+1)^2 - (y-1/2)^2 + r^2 = -d$

$(2x+3) + 0 \quad = d_1 - d = \Delta E$

(✓) $\Delta E = (2x+3)$

$f(m_2) = (x+2)^2 + (y-3/2)^2 - r^2 = d_2$

$f(m) = -(x+1)^2 - (y-1/2)^2 + r^2 = -d$

$(2x+3) + (-2y+2) \quad = d_2 - d = \Delta SE$

(✓) $\Delta SE = 2x - 2y + 5$

dinit $= (x_0+1)^2 + (y_0 - 1/2)^2 - r^2$

$= 5/4 - r$

For $r \leq 1 \rightarrow \Delta SE$

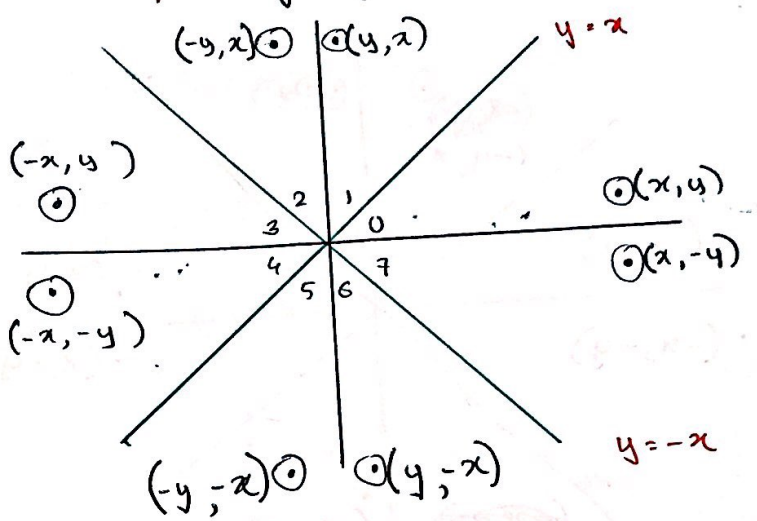
dinit = +ve

• dinit = -ve $\rightarrow \Delta E$

x	y	d	ΔE/ΔSE
0	10	-8.75 -35 ✓	ΔE
1	10	-5.75 23 ✓	ΔE
2	10	-0.75 -3 ✓	ΔE
3	10	6.25 25 ✓	ΔSE
4	9	-2.75 -4 ✓	ΔE
5	9	8.25 33 ✓	ΔSE
6	8	7.25 29 ✓	ΔSE
end → 7	7		

5 + 5 + 21

8-way Symmetry



Defn:

Zone division

The line drawing code for each individual zone is not optimized, since, we need to write a lot of redundant lines of code.

Hence, we use 8-way symmetry.

Explanation

If a ~~line~~ point is selected at zone 0, as (x, y) then we can derive its corresponding coordinates in the other zones by using reflection in the lines $(y = x)$ & $(y = -x)$

From the above figure it can be determined that any point in another zone can be traced back to zone 0 (x,y) pattern, by going through the corresponding rearrangement of the x & y coordinates in the other zones.

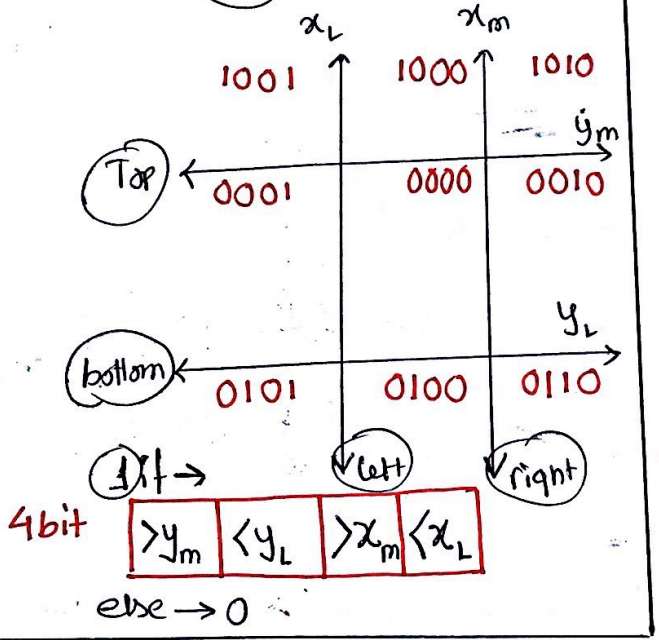
How algo works.

So, if we can derive the mid point line drawing algorithm for zone 0. We can use it draw a line in any other zone, just by rearrangement of the coordinates in any other zone as shown in figure above.

So we transform the points to zone 0. find the next pixel and by rearrangement, draw the pixel in its actual zone.

Cohen – Sutherland LCA

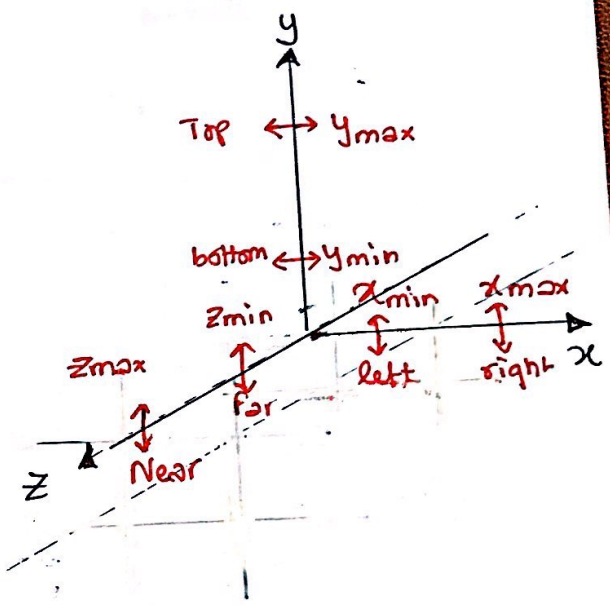
2D



4bit

$>y_m$	$<y_L$	$>x_m$	$<x_L$

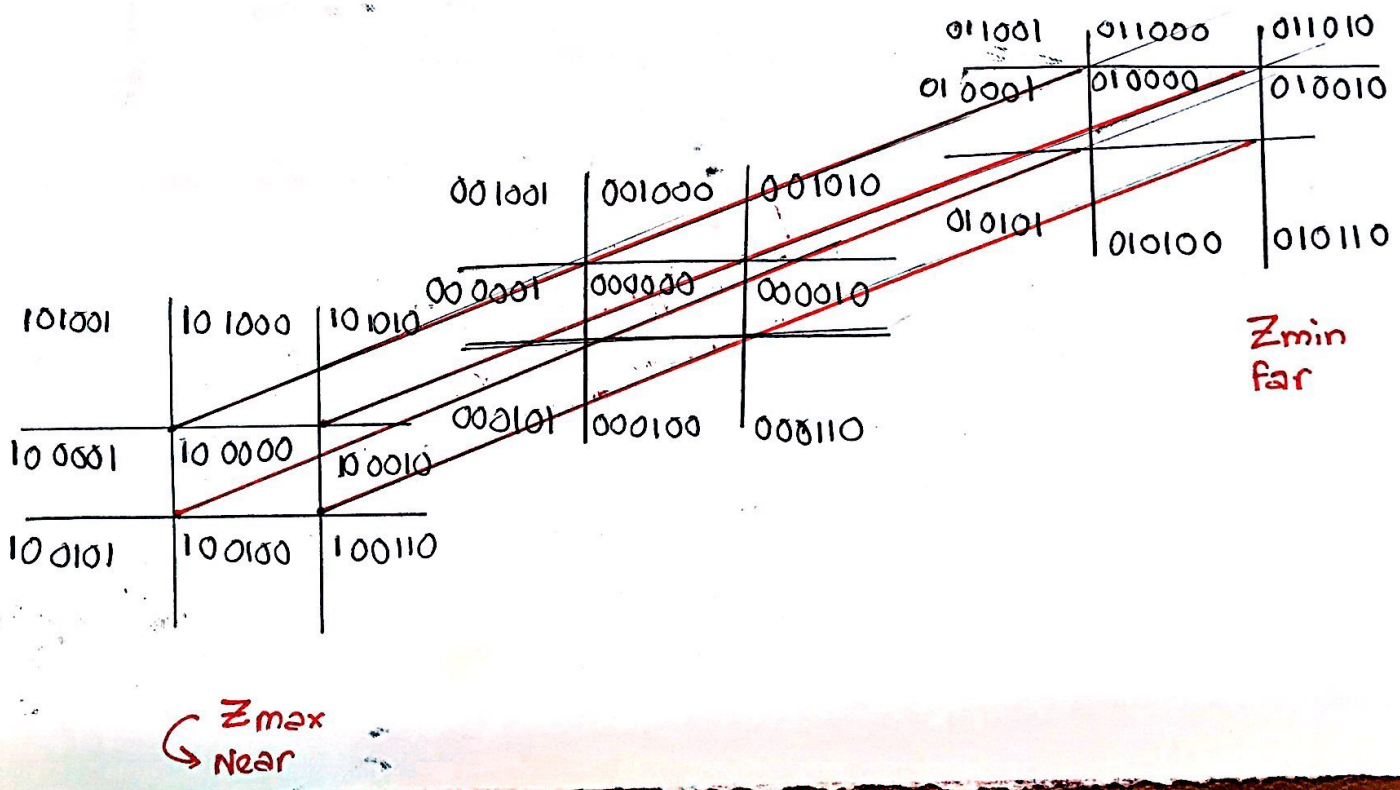
else → 0



3D

6bit

Near	far	top	bottom	right	left



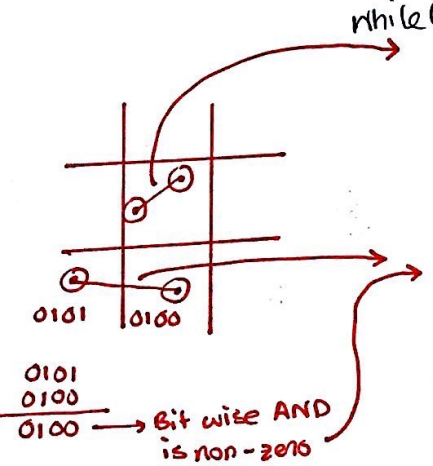
3D Cohen Sutherland Code

```
int  ymax = 100, ymin = -100, xmax = 120, xmin = -120, zmax = 100, zmin = -100
int  Near = 32, far = 16, Top = 8, Bottom = 4, right = 2, left = 1
int  makeCode (int x, int y, int z)
     int Code = 0;   // 0 0 0 0 0 0
     if (z > zmax) { // means adds 32 → so 32nd bit → 1
          Code += Near; → 32
     }
     else if (z < zmin) {
          Code += far;  → 16
     }
     if (y > ymax) {
          Code += top;  → 8
     }
     else if (y < ymin) {
          Code += bottom; → 4
     }
     if (x > xmax) {
          Code += right; → 2
     }
     else if (x < xmin) {
          Code += left; → 1
     }
     return Code;
}
```

(2) void Cohen-Sutherland (int x_0, int y_0, int z_0, int x_1, int y_1, int z_1)

```
int Code, Code0, Code1;
int x, y, z;
Code0 = makeCode(x0, y0, z0) // start point
Code1 = makeCode(x1, y1, z1) // end point

while(1){
    if(Code0 | Code1 == 0){ // complete accept
        drawline(x0, y0, z0, x1, y1, z1);
        break;
    }
    else if(Code0 & Code1){ // complete reject
        break;
    }
    else { // Partial Accept OR Reject
        if(Code0){
            Code = Code0;
        }
        else {
            Code = Code1;
        }
        if(Code & Near){ // z > zmax
            z = zmax;
            x = x0 + (z - z0)/(z1 - z0) * (x1 - x0);
            y = y0 + (z - z0)/(z1 - z0) * (y1 - y0);
        }
        else if(Code & far){ // z < zmin
            z = zmin
            x = x0 + (z - z0)/(z1 - z0) * (x1 - x0);
            y = y0 + (z - z0)/(z1 - z0) * (y1 - y0);
        }
    }
```



$x = x_0 + t(x_1 - x_0)$

$\rightarrow t = \dfrac{x - x_0}{x_1 - x_0}$

$\therefore t = \dfrac{z - z_0}{z_1 - z_0}$

$\therefore t = \dfrac{y - y_0}{y_1 - y_0}$

```
if (Code & Top) {   // y > ymax
    ------ same
}
else if (Code & Bottom) {   y < ymin
    ------ same
}
if (Code & right) {   x > xmax
    ------ same
}
else if (Code & left) {   x < xmin
    ------ same
}
if (Code = Code0) {
        x0 = x , y0 = y ; z0 = z ;
        Code0 = MakeCode (x0, y0, z0) ;
}
else {
        x1 = x , y1 = y , z1 = z ;
        Code 1 = MakeCode (x1, y1, z1) ;
}
} // partial else
} // while
} // end
```

Summary

→ will clip and make Code0 & Code1 again if out side, to fit it inside frame

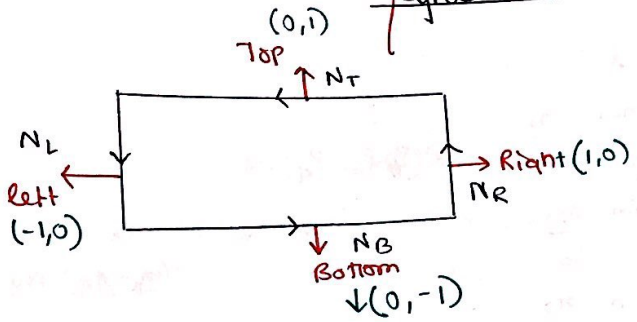
→ once inside from both points

point 0 → start → Code0 → 000000

point 1 → end → Code 1 → 000000

becomes fully accepted at first if cond and breaks.

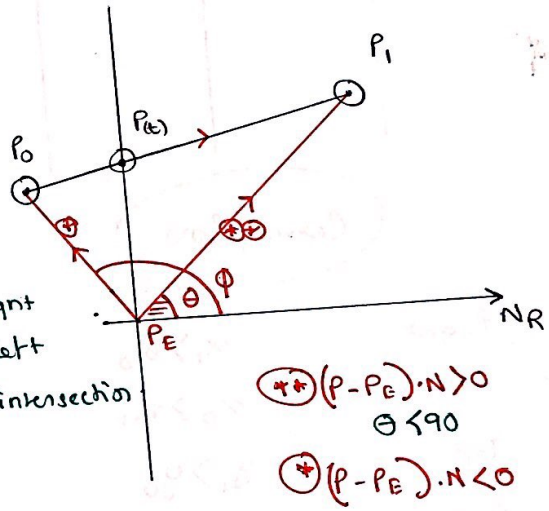
{Cyrus Back LCA}



∅ with respect to the direction of an edge anything left is inside and right outside.

	value
Top	(0,1)
bott	(0,-1)
left	(-1,0)
right	(1,0)

Value of $t \approx (0-1)$



(1) Parametric eqn of a line in view plane:

$$P_{(t)} = P_0 + t(P_1 - P_0)$$

→ at $t=0, P_t = P_0$ & at $t=1, P_t = P_1$

(2) dot product considering "cos"

$(P - P_E) \cdot N =$ <0, -ve, point on right
>0, +ve, point on left
0, point on edge, intersection

(**) $(P-P_E)\cdot N > 0$
$\theta < 90$

(*) $(P-P_E)\cdot N < 0$
$\phi > 90$

Conditions
(1) $\theta > 90$

So, if we consider, **DERIVATION**

$$P_{(t)} = P_0 + t(P_1 - P_0) \quad \text{---(i)}$$

and, for $P = P_{(t)}$ ---→ $(P - P_E)\cdot N = 0$ ---(ii)

Substituting, $P = P_{(t)}$

$$(P_t - P_E)\cdot N = 0$$ → substitute eqn 1

$$\Rightarrow (P_0 + t(P_1 - P_0) - P_E)\cdot N = 0$$

$$\Rightarrow \{(P_0 - P_E)\cdot N\} + \{t(P_1 - P_0)\cdot N\} = 0$$

$$\Rightarrow t = \frac{-(P_0 - P_E)\cdot N}{(P_1 - P_0)\cdot N} = \frac{(P_E - P_0)}{(P_1 - P_0)}$$

(*) List of t for all possible boundaries

Boundary	$P_{0E} - P_0$	$P_1 - P_0$	t
Top	$y_{max} - y_0$	$y_1 - y_0$	$\frac{y_{max} - y_0}{y_1 - y_0}$
Bottom	$y_{min} - y_0$	$y_1 - y_0$	$\frac{y_{min} - y_0}{y_1 - y_0}$
left	$x_{min} - x_0$	$x_1 - x_0$	$\frac{x_{min} - x_0}{x_1 - x_0}$
Right	$x_{max} - x_0$	$x_1 - x_0$	$\frac{x_{max} - x_0}{x_1 - x_0}$

(*) $(P_E - P_0) \cdot N$

$= (x_E - x_0) \cdot N_x + (y_E - y_0) \cdot N_y$

(*) $(P_1 - P_0) N$

$= (x_1 - x_0) \cdot N_x + (y_1 - y_0) \cdot N_y$.

Leaving Cond

Right	$x_1 > x_0$
left	$x_0 > x_1$
Top	$y_1 > y_0$
Bottom	$y_0 > y_1$

Entering Cond

Right	$x_0 > x_1$
left	$x_1 > x_0$
Top	$y_1 < y_0$
Bottom	$y_1 > y_0$

(Algorithm)

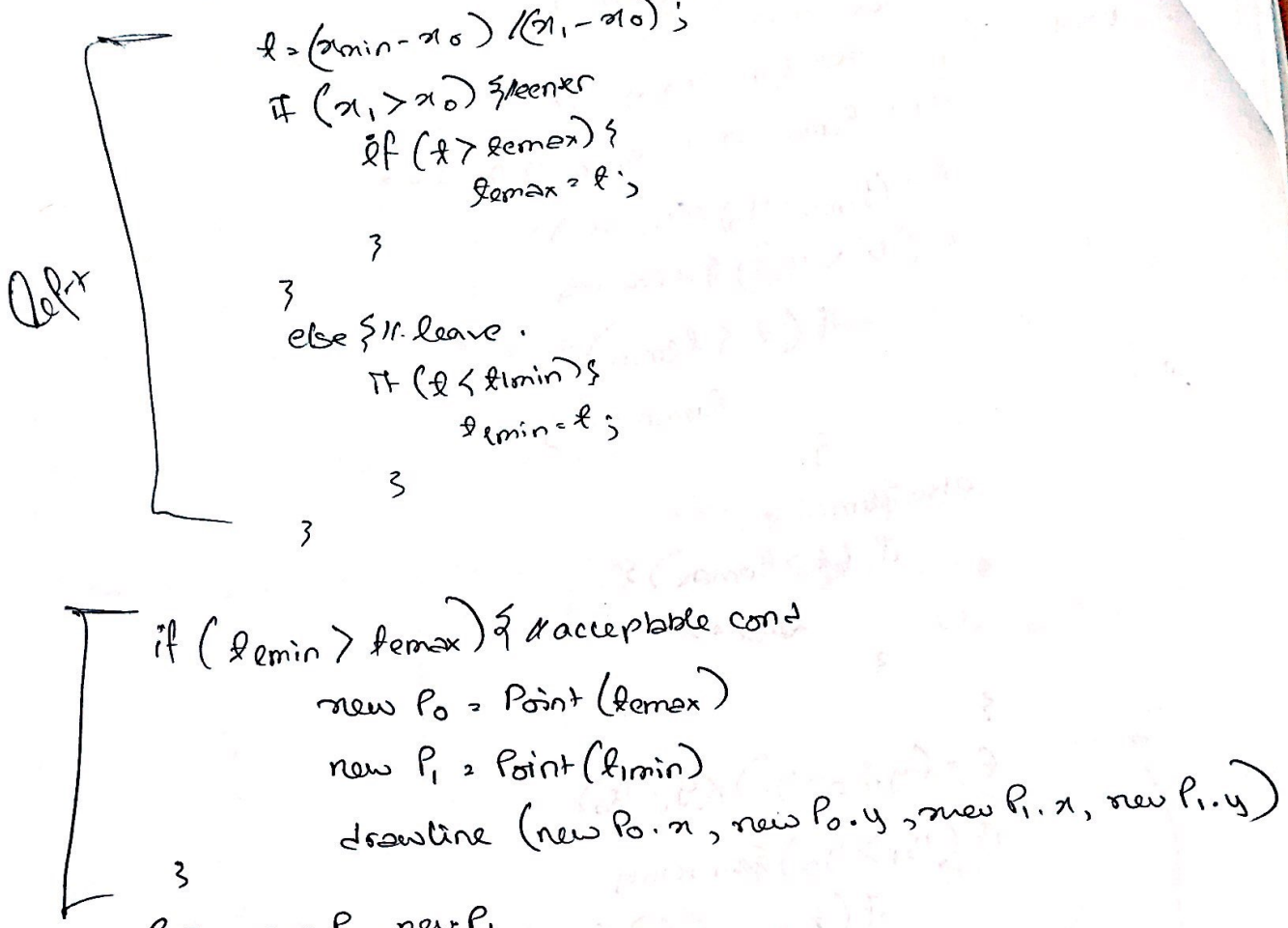
$y_{max}, y_{min}, x_{max}, x_{min}$.

```
Void Cyrus beck (int x0, y0, x1, y1){
    float tEmax = 0.0, tlmin = 0.0, t = 0;
    t = (ymax - y0)/(y1 - y0);
    if (y1 > y0) {//leaving
        if (t < tlmin){
            tlmin = t;
        }
    else{//entering
        if (t > temax){
            tmax = t
        }
    }
    t = (ymin - y0)/(y1 - y0)
    if (y1 > y0){//entering
        if(temax < t){
            temax = t;
        }
    else {//leaving
        if (tlmin > t){
            tlmin = t;
        }
    }
    t = (xmax - x0)/(x1 - x0)
    if (x1 > x0){//leaving
        if (tlmin > t){
            tlmin = t;
        }
    else{//entering
        if (t > temax){
            temax = t
        }
    }
}
```

TOP

BOTTOM

RIGHT



$y_{max} = 50$, $y_{min} = -50$, $x_{max} = 60$, $x_{min} = -60$

$(x_0, y_0) = (100, 60)$

$(x_1, y_1) = (110, -70)$

$t_{top} = \frac{50 - (60)}{-70 - (60)} = 0.0769$ → $y_1 < y_0$ → entering

$t_{bottom} = \frac{-50 - (60)}{-70 - (60)} = 0.846$ → $y_1 < y_0$ → leaving

$t_{left} = \frac{-60 - (-100)}{110 + 100} = 0.190$ → $x_1 > x_0$ → enter ✓ t_{xmax}

$t_{right} = \frac{60 - (-100)}{210} = 0.762$ → $x_1 > x_0$ → leave ✓ t_{xmin}

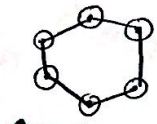
✓ ✓

$t_{left} \rightarrow t_{right}$

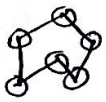
$t_{xmax} \rightarrow t_{xmin}$

line drawn.

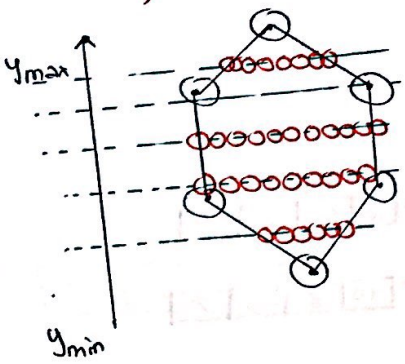
{POLYGON FILL & CLIP}



Convex polygon



Concave polygon.



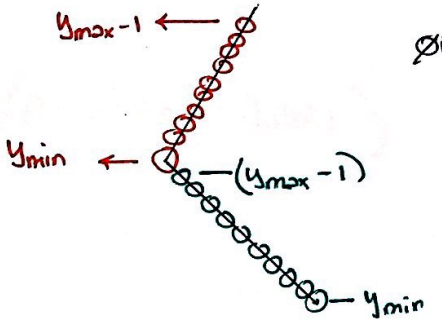
{Filling starts}

from y_{min} to y_{max} (iteration)

&

fills from (x_{min} to x_{max})

Ø Can't use midpoint algo, because it will have a lot of intersections and can create an ambiguous situation.



Ø we draw from ymin to y_{max-1}, so no double overlaps

(Edge Table)

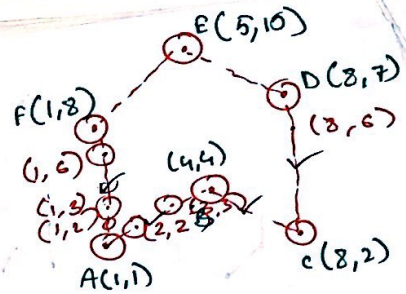
→ Ø only considers the corner points
Ø boundary set

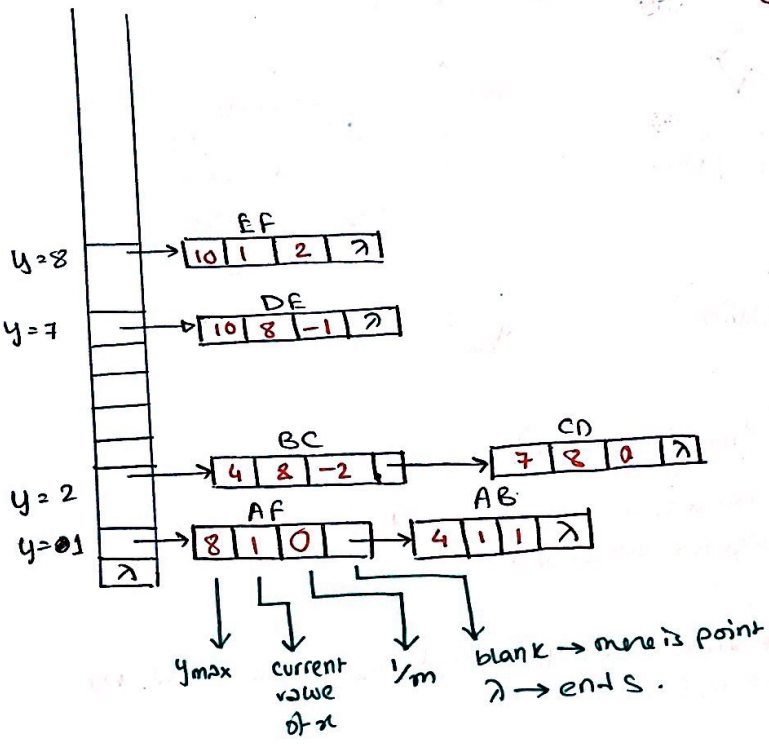
Ø we don't consider horizontal lines. they are omitted.

(Active Edge Table)

Ø has all the values of x, for all the values of y.

Edge Table



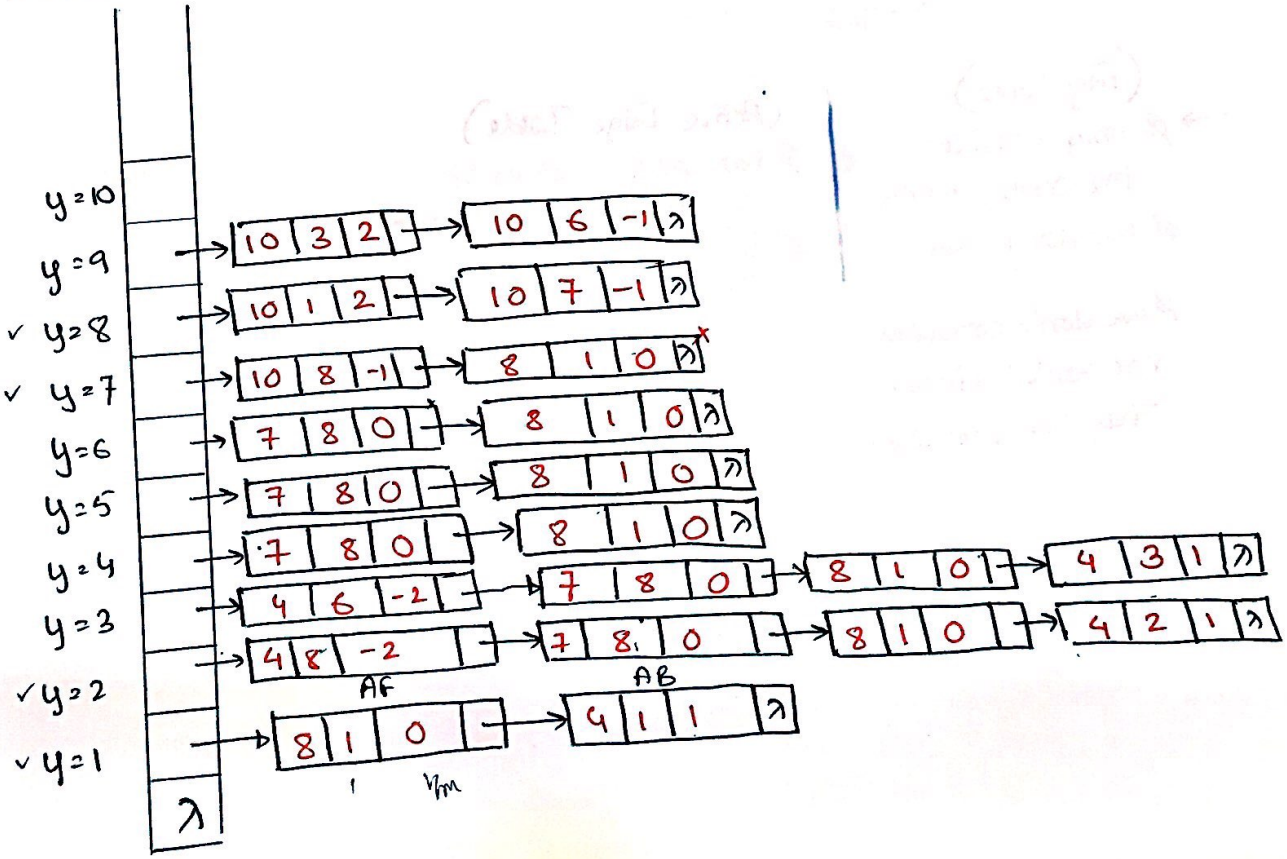


$y_{max} = 10$

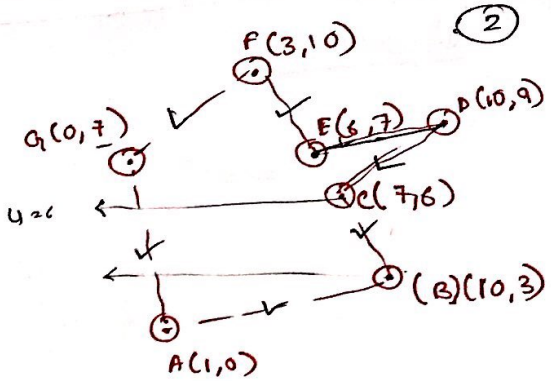
$y_{min} = 1$

So $\{1 \rightarrow 9\}$

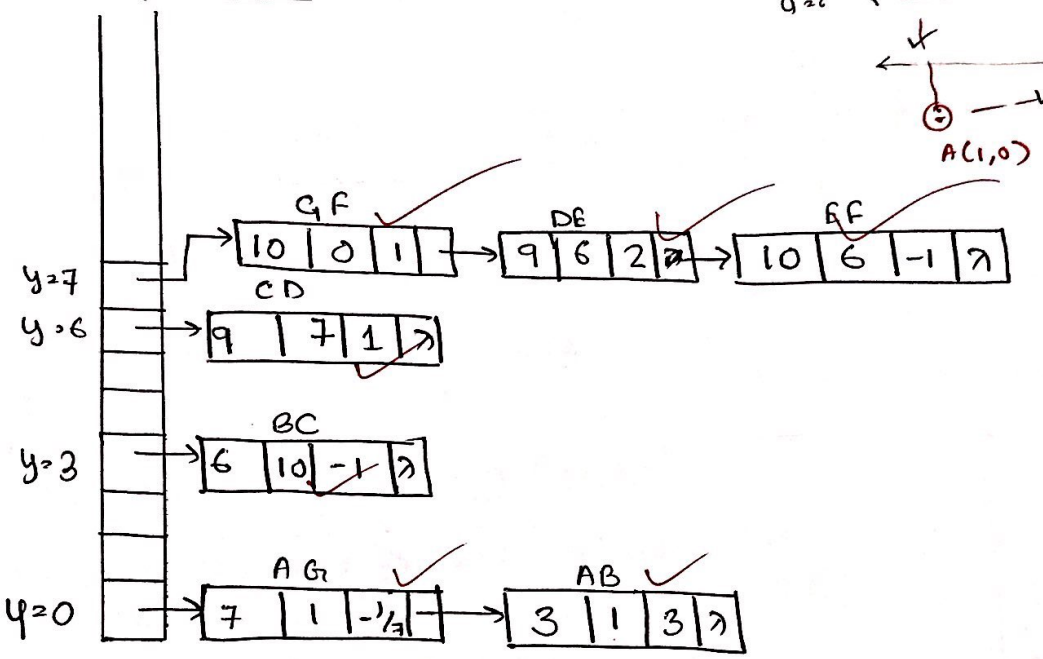
Active Edge table.



Question 2



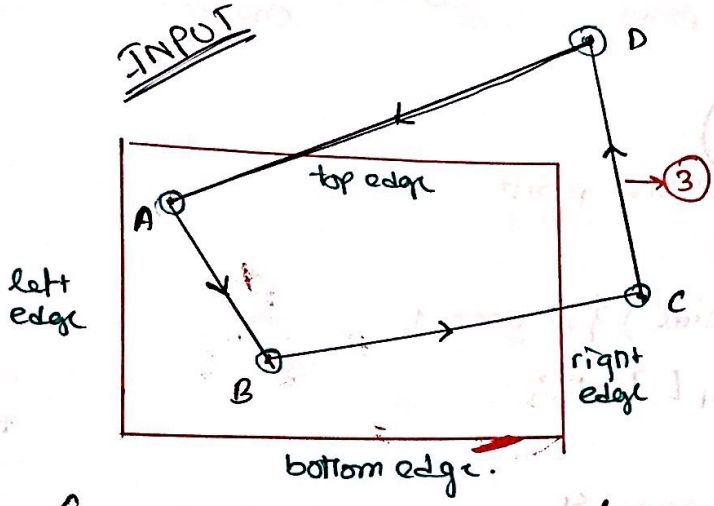
Edge table



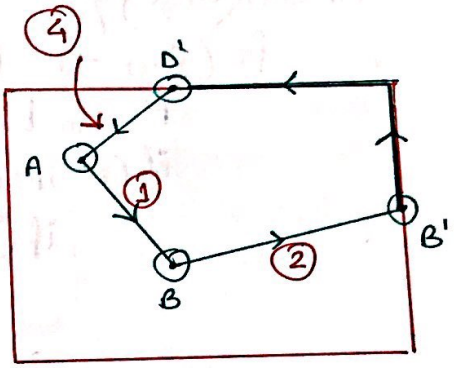
(Active Edge table)

Polygon Clipping

INPUT



OUTPUT



Sutherland-Hodgeman PCA (4 rules)

(1) Both the start and end point are inside the frame. (in to in movement) → (A B)
Output : Destination vertex
= B

(2) One point inside another outside the frame
(movement in to out) → (B C)
Ostp : intersection point
= B'

(3) Both points outside the frame, so we ignore that (in case of CD)
Output : nothing.

(4) One point outside, two other inside the frame.
(movement D A → out to in)
output : intersection point & destination point.
= D' A

Usually done in 3 parts :
1. left clip
2. right clip
3. bottom clip
4. Top clip.

(Algorithm)

N → no. of points
0 → (n-1) choice.

```
void PolygonClip (
    S = Pin[n-1]; // start point
    For each(edge) {
        For (j=0 to (n-1))
            P = Pin[j]; // end point
            if (P inside) {
                if (S inside) { // case 1
                    Pout[] = P;
                }
                else { // case 4
                    i = intersection (S,P);
                    Pout[] = i;
                    Pout[] = P;
                }
            }
            else { // P outside.
                if (S inside) { // case 2
                    i = intersection (S,P);
                    Pout[] = i;
                }
                else { // case 3
                    // do nothing.
                }
            }
            S = P;
        } // end for loop
    } // end of each edge.
```