1. DDA LINE GENERATION ALGORITHM

```
#include <graphics.h>
#include <conio.h>
#include <math.h>
#include <stdlib.h>

void drawDDA(int x1, int y1, int x2, int y2) {
   int dx = x2 - x1;
   int dy = y2 - y1;

   // Calculate the number of steps
   int steps = abs(dx) > abs(dy) ? abs(dx) : abs(dy);

   // Calculate the increment for each step
   float xIncrement = dx / (float)steps;
   float yIncrement = dy / (float)steps;

   // Start point
   float x = x1;
   float y = y1;

   // Declare loop variable outside the for loop
   int i;

   // Draw the line
   for (i = 0; i <= steps; i++) {
      putpixel((int)(x + 0.5), (int)(y + 0.5), WHITE); // Draw pixel
      x += xIncrement;
      y += yIncrement;
      delay(100);
   }
}

int main() {
   int gd = DETECT, gm;
   int x1, y1, x2, y2; // Declare all variables at the start of the block

   // Initialize the graphics mode
   initgraph(&gd, &gm, "C:\\TURBOC3\\BGI");

   printf("Enter the coordinates of the first point (x1, y1): ");
   scanf("%d %d", &x1, &y1);

   // Print the first point
```

```
    printf("First point entered: x1 = %d, y1 = %d\n", x1, y1);

    printf("Enter the coordinates of the second point (x2, y2): ");
    scanf("%d %d", &x2, &y2);

    // Call the DDA line-drawing function
    drawDDA(x1, y1, x2, y2);

    getch(); // Wait for user input
    closegraph(); // Close the graphics mode
    return 0;
}
```

2. BRESENHEM 'S LINE GENERATION ALGORITHM

```
 #include<stdio.h>

#include<graphics.h>
 #include<conio.h>

 void  main()  {
    int gd = DETECT, gm;
    int x1, y1, x2, y2, dx, dy, x, y, pk;
    // Initialize graphics mode
    initgraph(&gd,  &gm,  "C:\\TURBOC3\\BGI");
    // Input the start and end points

    printf("Enter the 1st point coordinates (x1, y1): ");

    scanf("%d%d", &x1, &y1);

    printf("Enter the 2nd point coordinates (x2, y2): ");

    scanf("%d%d", &x2, &y2);

    // Calculate differences

    dx = x2 - x1;

    dy = y2 - y1;
    // Determine the initial decision parameter

    pk = (2 * dy) - dx;

    // Set starting point

    x = x1;

    y = y1;
    // Draw the initial pixel

    putpixel(x,  y,  WHITE);
    // Bresenham's Line Algorithm for positive slope
    while (x < x2) {

    x=x+1;

    if (pk >= 0)
       { y=y+1;

       pk += (2 * dy) - (2 * dx);
    } else {

       pk += 2 * dy;
    }
    putpixel(x,  y,  WHITE);

    delay(10); // Add delay for visualization
    }
    getch();

}
```

3. MIDPOINT-LINE GENERATION ALORITHM

```
#include <graphics.h>
#include <conio.h>
#include <stdlib.h>

void drawMidpointLine(int x1, int y1, int x2, int y2) {
    int dx, dy, d, x, y, xIncrement, yIncrement;

    // Calculate dx and dy
    dx = abs(x2 - x1);
    dy = abs(y2 - y1);

    // Initialize starting point
    x = x1;
    y = y1;

    // Determine increments for x and y
    xIncrement = (x2 > x1) ? 1 : -1;
    yIncrement = (y2 > y1) ? 1 : -1;

    // Initial decision parameter
    d = 2 * dy - dx;

    // Draw initial pixel
    putpixel(x, y, WHITE);

    // For a line with a shallow slope (dx > dy)
    if (dx > dy) {
        while (x != x2) {
            x += xIncrement;
            if (d < 0) {
                d += 2 * dy;
            } else {
                y += yIncrement;
                d += 2 * (dy - dx);
            }
            putpixel(x, y, WHITE);
        }
    }
    // For a line with a steep slope (dy >= dx)
    else {
        while (y != y2) {
            y += yIncrement;
            if (d < 0) {
                d += 2 * dx;
            } else {
                x += xIncrement;
            d += 2 * (dx - dy);
        }
        putpixel(x, y, WHITE);
    }
  }
}

int main() {
  int gd = DETECT, gm;
  int x1, y1, x2, y2;

  // Initialize graphics mode
  initgraph(&gd, &gm, "C:\\TURBOC3\\BGI");

  // Input points
  printf("Enter the coordinates of the first point (x1, y1): ");
  scanf("%d %d", &x1, &y1);

  printf("Enter the coordinates of the second point (x2, y2): ");
  scanf("%d %d", &x2, &y2);

  // Draw the line using Midpoint Line Drawing Algorithm
  drawMidpointLine(x1, y1, x2, y2);

  getch(); // Wait for user input
  closegraph(); // Close graphics mode
  return 0;
}
```

4. BRESENHAM’S CIRCLE DRAWING ALGORITHM

```
#include <graphics.h>
#include <conio.h>
#include <stdio.h>

// Function to draw the eight symmetric points of the circle
void drawCirclePoints(int xc, int yc, int x, int y) {
   putpixel(xc + x, yc + y, WHITE); // Octant 1
   putpixel(xc - x, yc + y, WHITE); // Octant 2
   putpixel(xc + x, yc - y, WHITE); // Octant 3
   putpixel(xc - x, yc - y, WHITE); // Octant 4
   putpixel(xc + y, yc + x, WHITE); // Octant 5
   putpixel(xc - y, yc + x, WHITE); // Octant 6
   putpixel(xc + y, yc - x, WHITE); // Octant 7
   putpixel(xc - y, yc - x, WHITE); // Octant 8
}

// Bresenham's Circle Drawing Algorithm
void drawCircle(int xc, int yc, int r) {
   int x = 0, y = r;
   int d = 3 - 2 * r; // Initial decision parameter

   // Draw the initial points
   drawCirclePoints(xc, yc, x, y);

   // Iterate until x >= y
   while (x <= y) {
      x++;

      // Update decision parameter
      if (d < 0) {
         d = d + 4 * x + 6;
      } else {
         y--;
         d = d + 4 * (x - y) + 10;
      }

      // Draw symmetric points
      drawCirclePoints(xc, yc, x, y);
   }
}

int main() {
   int gd = DETECT, gm;
   int xc, yc, r;

   // Initialize graphics mode
   initgraph(&gd, &gm, "C:\\TURBOC3\\BGI");

   // Input the center and radius of the circle
   printf("Enter the coordinates of the center (xc, yc): ");
   scanf("%d %d", &xc, &yc);

   printf("Enter the radius of the circle (r): ");
   scanf("%d", &r);

   // Draw the circle using Bresenham's algorithm
   drawCircle(xc, yc, r);

   getch(); // Wait for user input
   closegraph(); // Close graphics mode
   return 0;
}
```

5. MIDPOINT CIRCLE DRAWING ALGORITHM

```
#include <graphics.h>
#include <conio.h>
#include <stdio.h>

// Function to draw the eight symmetric points of the circle
void drawCirclePoints(int xc, int yc, int x, int y) {
   putpixel(xc + x, yc + y, WHITE); // Octant 1
   putpixel(xc - x, yc + y, WHITE); // Octant 2
   putpixel(xc + x, yc - y, WHITE); // Octant 3
   putpixel(xc - x, yc - y, WHITE); // Octant 4
   putpixel(xc + y, yc + x, WHITE); // Octant 5
   putpixel(xc - y, yc + x, WHITE); // Octant 6
   putpixel(xc + y, yc - x, WHITE); // Octant 7
   putpixel(xc - y, yc - x, WHITE); // Octant 8
}

// Midpoint Circle Drawing Algorithm
void drawMidpointCircle(int xc, int yc, int r) {
   int x = 0, y = r;             // Starting point
   int d = 1 - r;               // Initial decision parameter

   drawCirclePoints(xc, yc, x, y); // Draw initial points

   // Iterate until x >= y
   while (x < y) {
      x++;

      // Update decision parameter
      if (d < 0) {
         d = d + 2 * x + 1;
      } else {
         y--;
         d = d + 2 * (x - y) + 1;
      }

      // Draw the symmetric points
      drawCirclePoints(xc, yc, x, y);
   }
}

int main() {
   int gd = DETECT, gm;
   int xc, yc, r;
```

```
    // Initialize graphics mode
    initgraph(&gd, &gm, "C:\\TURBOC3\\BGI");

    // Input the center and radius of the circle
    printf("Enter the coordinates of the center (xc, yc): ");
    scanf("%d %d", &xc, &yc);

    printf("Enter the radius of the circle (r): ");
    scanf("%d", &r);

    // Draw the circle using the Midpoint Circle Drawing Algorithm
    drawMidpointCircle(xc, yc, r);

    getch(); // Wait for user input
    closegraph(); // Close graphics mode
    return 0;
}
```

6. 2-D SCALING

```
#include <graphics.h>
#include <conio.h>
#include <stdio.h>

void drawTriangle(int x[], int y[], int color) {
   setcolor(color);
   line(x[0], y[0], x[1], y[1]); // Line from Point 1 to Point 2
   line(x[1], y[1], x[2], y[2]); // Line from Point 2 to Point 3
   line(x[2], y[2], x[0], y[0]); // Line from Point 3 to Point 1
}

void scaleTriangle(int x[], int y[], float sx, float sy) {
   for (int i = 0; i < 3; i++) {
      x[i] = x[0] + (int)((x[i] - x[0]) * sx); // Scale X-coordinate
      y[i] = y[0] + (int)((y[i] - y[0]) * sy); // Scale Y-coordinate
   }
}

int main() {
   int gd = DETECT, gm;
   int x[3], y[3]; // Arrays for triangle vertices
   float sx, sy;   // Scaling factors

   // Initialize graphics mode
   initgraph(&gd, &gm, "C:\\TURBOC3\\BGI");

   // Input the vertices of the triangle
   printf("Enter the coordinates of the triangle vertices:\n");
   for (int i = 0; i < 3; i++) {
        printf("Vertex %d (x, y): ", i + 1);
        scanf("%d %d", &x[i], &y[i]);
    }

    // Input the scaling factors
    printf("Enter scaling factor for X (sx): ");
    scanf("%f", &sx);
    printf("Enter scaling factor for Y (sy): ");
    scanf("%f", &sy);

    // Draw the original triangle
    drawTriangle(x, y, WHITE);
    printf("Original triangle drawn. Press any key to apply scaling.\n");
    getch();

    // Apply scaling transformation
    scaleTriangle(x, y, sx, sy);

    // Draw the scaled triangle
    drawTriangle(x, y, GREEN);
    printf("Scaled triangle drawn. Press any key to exit.\n");
    getch();

    // Close graphics mode
    closegraph();
    return 0;
}
```

7. <u>2D TRANSLATION</u>

```
#include <graphics.h>
#include <conio.h>
#include <stdio.h>

// Function to translate a rectangle
void translateRectangle(int *x1, int *y1, int *x2, int *y2, int tx, int ty) {
    *x1 += tx;
    *y1 += ty;
    *x2 += tx;
    *y2 += ty;
}

void main() {
    int gd = DETECT, gm;
    int x1, y1, x2, y2, tx, ty;

    // Initialize graphics mode
    initgraph(&gd, &gm, "C:\\TURBOC3\\BGI");

    // Get rectangle coordinates from user
    printf("Enter top-left corner of rectangle (x1, y1): ");
    scanf("%d%d", &x1, &y1);
    printf("Enter bottom-right corner of rectangle (x2, y2): ");
    scanf("%d%d", &x2, &y2);

    // Draw the original rectangle
    rectangle(x1, y1, x2, y2);
    outtextxy(10, 10, "Original Rectangle");
    // Get translation factors
    printf("Enter translation factors (tx, ty): ");
    scanf("%d%d", &tx, &ty);

    // Wait for user to view original rectangle
    getch();
    cleardevice();

    // Call the translation function
    translateRectangle(&x1, &y1, &x2, &y2, tx, ty);

    // Draw the translated rectangle
    rectangle(x1, y1, x2, y2);
    outtextxy(10, 10, "Translated Rectangle");

    // Wait for user to exit
    getch();

    // Close graphics mode
    closegraph();
}
```

8. FOR ROTATION :

```
#include <graphics.h>
#include <conio.h>
#include <stdio.h>
#include <math.h>

// Function to rotate a point (x, y) around a pivot (px, py) by angle theta
void rotatePoint(int *x, int *y, int px, int py, float theta) {
   int x_old = *x, y_old = *y;
   float rad = theta * M_PI / 180.0; // Convert angle to radians

   // Apply rotation formula
   *x = px + (int)((x_old - px) * cos(rad) - (y_old - py) * sin(rad));
   *y = py + (int)((x_old - px) * sin(rad) + (y_old - py) * cos(rad));
}

void main() {
   int gd = DETECT, gm;
   int x1, y1, x2, y2, px, py, x3, y3, x4, y4;
   float angle;

   // Initialize graphics mode
   initgraph(&gd, &gm, "C:\\TURBOC3\\BGI");

   // Input rectangle coordinates
   printf("Enter top-left corner of rectangle (x1, y1): ");
   scanf("%d%d", &x1, &y1);
   printf("Enter bottom-right corner of rectangle (x2, y2): ");
   scanf("%d%d", &x2, &y2);

   // Input pivot point for rotation
```

```
printf("Enter pivot point (px, py): ");
scanf("%d%d", &px, &py);

// Input rotation angle
printf("Enter rotation angle (in degrees): ");
scanf("%f", &angle);

// Draw the original rectangle
rectangle(x1, y1, x2, y2);
outtextxy(10, 10, "Original Rectangle");

// Wait for user to view the original rectangle
getch();
cleardevice();

// Rotate each corner of the rectangle
rotatePoint(&x1, &y1, px, py, angle);
rotatePoint(&x2, &y2, px, py, angle);

// Compute other two corners of the rectangle
x3 = x1, y3 = y2;
x4 = x2, y4 = y1;

// Rotate the computed corners
rotatePoint(&x3, &y3, px, py, angle);
rotatePoint(&x4, &y4, px, py, angle);

// Draw the rotated rectangle
line(x1, y1, x3, y3); // Top side
line(x3, y3, x2, y2); // Right side
line(x2, y2, x4, y4); // Bottom side
```

```
    line(x4, y4, x1, y1); // Left side


    outtextxy(10, 10, "Rotated Rectangle");


    // Wait for user to exit
    getch();
    closegraph();
}
```

9. Mirror image:

```
#include <graphics.h>
#include <conio.h>
#include <stdio.h>

// Function to reflect a point across the x-axis, y-axis, or origin
void reflectPoint(int *x, int *y, char axis) {
    switch (axis) {
        case 'x': // Reflect about x-axis
            *y = -*y;
            break;
        case 'y': // Reflect about y-axis
            *x = -*x;
            break;
        case 'o': // Reflect about origin
            *x = -*x;
            *y = -*y;
            break;
        default:
            printf("Invalid axis selected!\n");
            break;
    }
}

void main() {
    int gd = DETECT, gm;
    int x1, y1, x2, y2;
    char axis;
    int centerX, centerY;
```

```
// Initialize graphics mode
initgraph(&gd, &gm, "C:\\TURBOC3\\BGI");

// Get screen center
centerX = getmaxx() / 2;
centerY = getmaxy() / 2;

// Input rectangle coordinates (relative to the center)
printf("Enter top-left corner of rectangle (x1, y1): ");
scanf("%d%d", &x1, &y1);
printf("Enter bottom-right corner of rectangle (x2, y2): ");
scanf("%d%d", &x2, &y2);

// Input axis of reflection
printf("Enter axis for reflection (x for x-axis, y for y-axis, o for origin): ");
scanf(" %c", &axis);

// Translate coordinates to graphics coordinate system
x1 += centerX; y1 = centerY - y1;
x2 += centerX; y2 = centerY - y2;

// Draw the original rectangle
rectangle(x1, y1, x2, y2);
outtextxy(10, 10, "Original Rectangle");

// Wait for user to view the original rectangle
getch();
cleardevice();

// Translate back to original system for reflection
x1 -= centerX; y1 = centerY - y1;
```

```
    x2 -= centerX; y2 = centerY - y2;

    // Reflect each corner of the rectangle
    reflectPoint(&x1, &y1, axis);
    reflectPoint(&x2, &y2, axis);

    // Translate back to graphics coordinate system
    x1 += centerX; y1 = centerY - y1;
    x2 += centerX; y2 = centerY - y2;

    // Draw the mirrored rectangle
    rectangle(x1, y1, x2, y2);
    outtextxy(10, 10, "Mirrored Rectangle");

    // Wait for user to exit
    getch();
    closegraph();
}
```

Q10. flood_fill

```
#include <graphics.h>

#include <stdio.h>

#include<conio.h>

#include<dos.h>


void flood_Fill(int x, int y, int fill_Color, int old_Color)

   {

   if (getpixel(x, y) == old_Color)

     {


       putpixel(x, y, fill_Color); // Set the pixel to the fill color



       // To fill surrounding pixels

      flood_Fill(x + 1, y, fill_Color, old_Color); // Right side

      flood_Fill(x - 1, y, fill_Color, old_Color); // Left side

      flood_Fill(x, y + 1, fill_Color, old_Color); // Down side

      flood_Fill(x, y - 1, fill_Color, old_Color); // Up side

   }

}
```

```
void main()
 {
   int gd = DETECT, gm;


   int x, y, fill_Color, old_Color;
   initgraph(&gd,  &gm,  "C:\\Turboc3\\BGI");


   // A rectangle's point

   rectangle(100, 100, 200, 200);

   // Set the starting point for filling x

   = 150;

   y = 150;


   fill_Color = RED;

   old_Color = BLACK;

   flood_Fill(x, y, fill_Color, old_Color);


   getch();

}
```

Q11. Boundary Fill

```
#include <graphics.h>
#include <stdio.h>
#include<conio.h>

void boundaryFill(int x, int y, int fillColor, int boundaryColor)
 {


   if (getpixel(x,y)!= boundaryColor && getpixel(x,y)!= fillColor)
   { putpixel(x, y, fillColor); // Set the pixel to the fill color


      delay(30);
   boundaryFill(x + 1, y, fillColor, boundaryColor); // Right
   boundaryFill(x, y-1, fillColor, boundaryColor); // Left
   boundaryFill(x, y + 1, fillColor, boundaryColor); // Down
   boundaryFill(x-1, y , fillColor, boundaryColor); // Up


   }

}


void main()
{
```

```
int gd = DETECT, gm;

int x, y, fillColor, boundaryColor;


initgraph(&gd,  &gm,  "C:\\Turboc3\\BGI");


// Draw a closed boundary (e.g., a circle)
circle(200, 200, 50);


// Set_R the starting_A point_H inside the_U boundary_L x =
200;
y = 200;


fillColor = RED;
boundaryColor = WHITE;


boundaryFill(x, y, fillColor, boundaryColor);

getch();
```

Q12. cohen-sutherland line clipping

```
 #include<stdio.h>

#include<conio.h>

#include<graphics.h>

int   xwmax=300,xwmin=200,ywmax=100,ywmin=200,ax,ay,bx,by;   void

input()

{

   printf("Enter TWO points (x1,y1) & (x2,y2) to Draw a line :");

   scanf("%d%d%d%d",&ax,&ay,&bx,&by);

}

void draw()

{

   rectangle(xwmin,ywmin,xwmax,ywmax);


}

void clip(int x,int y,int p[4])


{

   if(y<ywmax)

      p[0]=1;



   if(y>ywmin)

      p[1]=1;
   if(x>xwmax)

      p[2]=1;

   if(x<xwmin)

      p[3]=1;

   else


      p[3]=0;

}


void main()

{

   int gd=DETECT,gm,y,x,c,p1[4],p2[4],p3[4],i;

   float m;

   initgraph(&gd,&gm,"C:\\TURBOC3\\BGI");

   cleardevice();

   input();

   cleardevice();

   clip(ax,ay,p1);

   clip(bx,by,p2);

   for(i=0;i<4;i++)

      p3[3]=p1[i]&&p2[i];

   for(i=0;i<4;i++)

   if(p3[i]==1)

      break;
```

```
draw();

line(ax,ay,bx,by);

getch(); cleardevice();

if(i!=4)

    draw();

    else

    {

    m=(float)(by-ay)/(bx-ax);

    if(p1[0]==1)

       y=ywmax; if(p1[1]==1)

       y=ywmin;

    if(p1[0]==1||p1[1]==1)

    {

       ax=ax+(y-ay)/m;

       ay=y;

    }

    if(p2[0]==1)

       y=ywmax; if(p2[1]==1)

       y=ywmin;

    if(p2[0]==1||p2[1]==1)

    {
```

```
    bx=bx+(y-by)/m; by=y;

}

if(p1[2]==1)


    x=xwmax;

if(p1[3]==1)

    x=xwmin;

if(p1[2]==1||p1[3]==1)

{

    ay=ay+m*(x-ax);

    ax=x;

}


if(p2[2]==1)

    x=xwmax;

if(p2[3]==1)

    x=xwmin;

if(p2[2]==1||p2[3]==1)

{

    by=by+m*(x-bx); bx=x;

}


draw(); line(ax,ay,bx,by);

}
```

```
    getch();

    closegraph();


}
```

Q13. Bezier Curve

```
  #include <graphics.h>

  #include <conio.h>

  #include <math.h>

  #include <stdio.h>


  // Line drawing function using DDA
void drawLine(int x1, int y1, int x2, int y2) { int dx, dy, steps, i;

    float xIncrement, yIncrement, x = x1, y = y1;



    dx = x2 - x1; dy

    = y2 - y1;


    steps = (abs(dx) > abs(dy)) ? abs(dx) : abs(dy);


    xIncrement = dx / (float)steps;

    yIncrement = dy / (float)steps;


    for (i = 0; i <= steps; i++)

    { putpixel((int)x, (int)y, GREEN);

    x += xIncrement;
    y += yIncrement;

    delay(50);

    }

}



// bezeir curve drawing function

void drawBezierCurve(int x[], int y[])

    { double putx, puty, t;

    for (t = 0.0; t <= 1.0; t += 0.001) {


    putx = pow(1 - t, 3) * x[0] + 3 * t * pow(1 - t, 2) * x[1] + 3

        * t * t * (1 - t) * x[2] + pow(t, 3) * x[3];

puty = pow(1 - t, 3) * y[0] + 3 * t * pow(1 - t, 2) * y[1] + 3 * t * t * (1 - t) * y[2] + pow(t, 3) * y[3];

    putpixel((int)putx, (int)puty, WHITE);

    }

}



void main() {

    int x[4], y[4], i;

    int gd = DETECT, gm;



    initgraph(&gd, &gm, "C:\\TURBOC3\\BGI");
    // Input points
    for (i = 0; i < 4; i++) {

    printf("Enter x and y coordinates of point %d: ", i + 1); scanf("%d%d",
    &x[i], &y[i]);
    putpixel(x[i], y[i], GREEN); // Display the points

    }


    // Draw lines between consecutive points for clarity for
    (i = 0; i < 3; i++) {
    drawLine(x[i], y[i], x[i + 1], y[i + 1]);

    }


    // Draw the Bezier curve
    drawBezierCurve(x, y);


    getch();
    closegraph();


}
```