Universidad de Guadalajara.
Centro Universitario de Ciencias Exactas e Ingenierías.

División de Electrónica y Computación.
Sem. de solución de problemas de algoritmia.

Sandoval Márquez Anthony Esteven
215660767

Planteamiento del problema.

Crear un programa que tenga la capacidad de seleccionar y analizar imágenes que contienen círculos negros. También será capaz de identificar círculos mediante la posición cartesiana en la que se encuentran. Los círculos que se identifiquen tendrán un id único para relacionar los círculos de la imagen y la lista con la información de los círculos. La lista de los círculos podrá clasificarse de mayor a menor área, de menor a mayor posición respecto al eje x y de menor a mayor posición respecto al eje y.

Se deben de analizar cualquier cantidad de imágenes en una sola ejecución, el programa puede repetir su análisis sin necesidad de tener que volver a abrir el programa.

Objetivos.

Detectar círculos negros en una imagen con fondo blanco. (Mostrar la imagen junto a una interfaz gráfica intuitiva)

Clasificar círculos: (Almacenar y mostrar las características de estos)

- Por tamaño. (Ordenamiento por Bubble Sort).
- Por orden de aparición en el eje x. (Ordenamiento por Selection Sort).
- Por orden de aparición en el eje y. (Ordenamiento por Insertion Sort).

Marco teórico.

Representación de imágenes en un sistema computacional.

Para mostrar las imágenes en una interfaz gráfica con el lenguaje c# son importantes las siguientes clases:

- **OpenFileDialog:** Muestra un cuadro de diálogo estándar que solicita al usuario que abra un archivo. Esta clase no puede heredarse.
- **PictureBox:** Representa un control de cuadro de imagen de Windows para mostrar una imagen.
- **Bitmap:** Encapsula un mapa de bits GDI +, que consta de los datos de píxeles de una imagen gráfica y sus atributos. Un mapa de bits es un objeto que se utiliza para trabajar con imágenes definidas por datos de píxeles.

Algoritmos de ordenamiento:

- **Selection sort.** Se comienza el ordenamiento por selección recorriendo la lista dada para encontrar el elemento más pequeño e intercambiarlo con el primer elemento, colocando así el elemento más pequeño en su posición final en la lista ordenada. Luego se recorre la lista, comenzando con el segundo elemento para encontrar el menor entre los últimos n−1n−1 elementos e intercambiarlo con el segundo, colocando el segundo elemento más pequeño en su posición final. Después de n–1n–1 pasadas, la lista está ordenada.

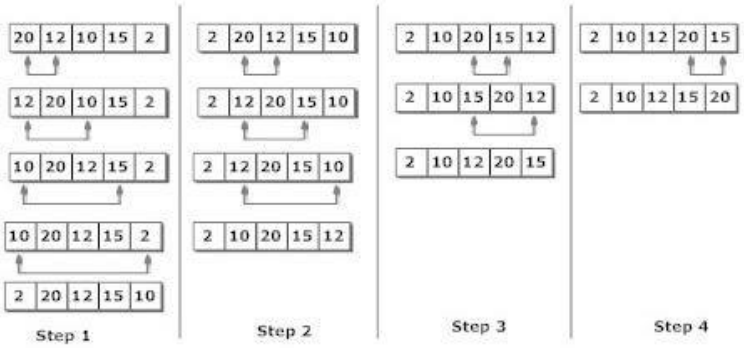
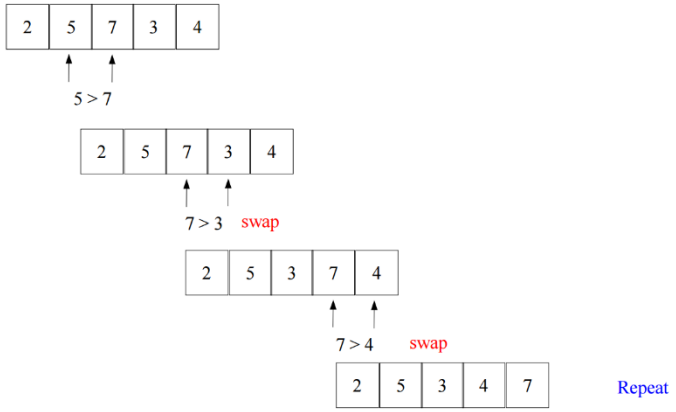
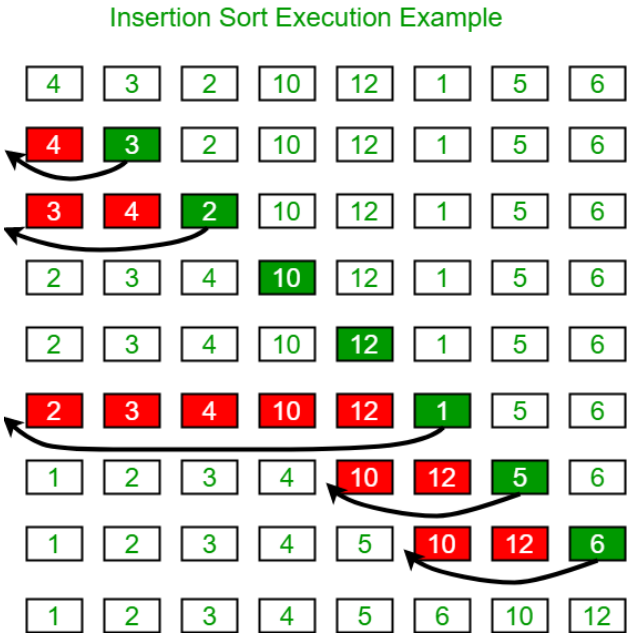


Figure: Selection Sort

- **Bubble sort.** El algoritmo de ordenamiento burbuja (bubble sort) hace una serie de pasadas a través de la lista de elementos. En cada pasada compara los valores de elementos adyacentes. Si están desordenados, los intercambia.



- **Insertion sort.** El ordenamiento por inserción trabaja de la forma en que muchas personas ordenan una mano de cartas de baraja. Se comienza con la mano izquierda vacía y las cartas hacia abajo. Después se remueve una carta de la mesa y se inserta en la posición correcta en la mano izquierda. Utiliza el ordenamiento por inserción cuando se tenga un número pequeño de elementos a ordenar o si los elementos en la colección inicial ya están “casi ordenados”.



Otros temas para el desarrollo.

- **ComboBox.** Representa un control de cuadro combinado de Windows.
- **System.Drawing.Drawing2D.** Proporciona funcionalidad avanzada de gráficos vectoriales y bidimensionales.

Bibliografia:

D. (s. f.). *OpenFileDialog Class (System.Windows.Forms)*. Microsoft Docs. Recuperado 26 de marzo de 2021, de https://docs.microsoft.com/en-us/dotnet/api/system.windows.forms.openfiledialog?view=net-5.0

D. (s. f.-b). *PictureBox Class (System.Windows.Forms)*. Microsoft Docs. Recuperado 26 de marzo de 2021, de https://docs.microsoft.com/en-us/dotnet/api/system.windows.forms.picturebox?view=net-5.0

D. (s. f.-a). *Bitmap Class (System.Drawing)*. Microsoft Docs. Recuperado 26 de marzo de 2021, de https://docs.microsoft.com/en-us/dotnet/api/system.drawing.bitmap?view=net-5.0

Sencion Echauri, F. (s. f.). *JupyterHub*. Ordenamientos. Recuperado 26 de marzo de 2021, de https://notebooks.gesis.org/binder/jupyter/user/sencionechauri-i5886-x2difhsg/notebooks/ordenamientos.ipynb

D. (s. f.-d). *System.Drawing.Drawing2D Espacio de nombres*. Microsoft Docs. https://docs.microsoft.com/es-es/dotnet/api/system.drawing.drawing2d?view=net-5.0

Desarrollo.

Primeramente, lo que tenía que hacer era una forma de visualizar imágenes en una interfaz gráfica creada en C#, para esto tome en cuenta la clase PictureBox, que me permitía crear un componente capaz de mostrar una imagen, para esto también hice uso de la clase OpenFileDialog que me proporciono una forma de obtener la ruta de la imagen que abriría. Con el uso de estas clases establecí los métodos que sería accionados al dar doble clic al PictureBox o clic al botón que dice “Seleccionar Imagen”.

Al tener una imagen cargada en el PictureBox entonces tenía que poder analizarla para encontrar círculos negros y para esto fue necesario el uso de un Bitmap, el cual me proporcionaba una manera de poder acceder y establecer los pixeles de una imagen, refiriéndome a su color y localización. Hice uso de la clase Color, para guardar los colores que obtenía del Bitmap. Teniendo ya todo lo anterior solo realice una serie de comparaciones que me permitían saber de qué color eran los pixeles

que comprobaba en una serie de repeticiones que se definían respecto al ancho y alto de la imagen, es decir, revisaba toda la imagen en busca de pixeles que fueran distintos al color blanco (255) y que también lo fueran del color Rojo, ya que este sería el que utilizaría para colorear todos los círculos posteriormente. Esto se acciona por el botón de “Análisis”.

Cuando encontrara un pixel negro, o, mejor dicho, un pixel distinto a blanco y rojo, era una señal de que había encontrado un circulo, por lo tanto, ahora lo que tocaba hacer era encontrar el centro de dicho circulo. Para encontrar el circulo realice una función que encontraría el centro del círculo, esta función se ejecutaría cuando se encontrara un pixel distinto a blanco y rojo, además de que recibiría por parámetros la posición del pixel (x, y) y el Bitmap para seguir obteniendo los pixeles. Para buscar el centro primero me base en los puntos de Y, utilice dos variables, una para determinar los pixeles que están al norte de la imagen y otra para los que están al sur, estas variables empezarían a contar desde el punto Y que recibí por parámetros. Para contar los pixeles solo realice un bucle con while que se ejecute hasta que ya no encuentre pixeles distintos de blanco, tanto en norte y sur. Cuando ya terminaba de contar los pixeles solo le reste al Y del sur lo que está el en norte, lo dividí entre 2 y se lo sume al norte de Y (Se podría denominar como el inicio de Y), así logre obtener el centro de Y. Para obtener el centro de X realice lo mismo pero con otras variables y utilizando lo que obtuve del centro de Y, al obtener completamente el centro procedí a guardar estos valores en una clase Circulo, además de también guardar un identificador, el oeste y este de X (inicio y fin) ya que los utilizaría más adelante. Los círculos que iba obteniendo los guardaba en una Lista Dinámica. Después de esto también se mandaba a llamar otra función que me colorearía el circulo que encontré.

En la clase Circulo almacene datos útiles para diversas funcionalidades, estos atributos son: el centro de x, el de y, el inicio de x, el fin de x, área del círculo, identificador y el radio. El fin e inicio de x los use para obtener el radio, y para obtener el área del circulo necesite del radio.

En el método para colorear el circulo recibí como parámetros el inicio de x, el centro de y, y el Bitmap, pues para pintar el circulo seleccione la siguiente manera: Avanzaría un pixel en x, cuando esto pasara entonces ya solo colorearía los pixeles en y que están al sur y al norte hasta que ya no se encuentren pixeles diferentes de blanco, todo esto repetirlo hasta que en x ya no halla tampoco pixeles distintos de blanco.

Cuando ya obtenía todos los círculos y los guardaba en la Lista dinámica ya solo lo que realizaba era volver a resetear el Bitmap para que los círculos ya no estuvieran coloreados. Después de lo anterior cree la funcionalidad para colocarle etiquetas a los círculos, en mi caso yo decidí ponerle letras Mayúsculas. Para esto utilice las clases RectangleF y Grafhics, además de otras que obtenía de

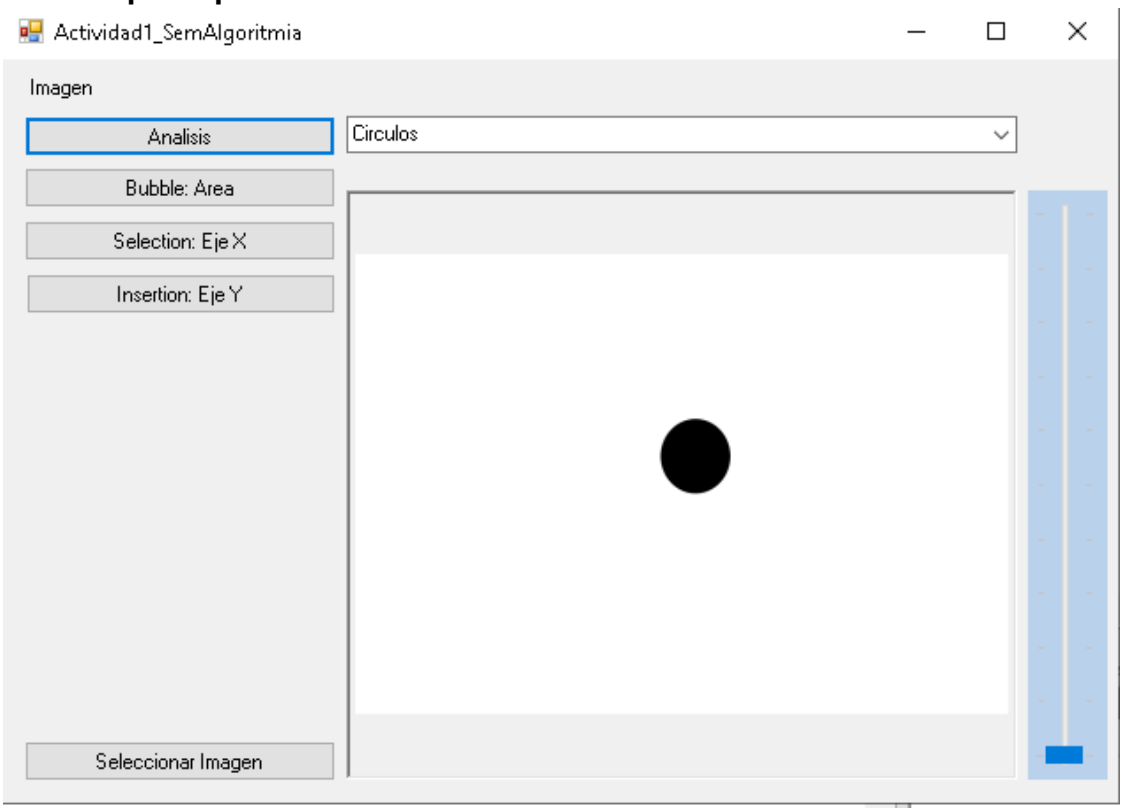
System.Drawing.Drawing2D, con es esas clases y un ciclo para recorrer la lista logre colocar las etiquetas en la imagen con la ayuda también del Bitmap.

Después de realizar un análisis de la imagen llamaba a una función que simplemente me cargaba un ComboBox a partir de la Lista dinámica que contenía a todos los círculos encontrados de la imagen, a partir de esto también se mostraron las características más importantes de cada circulo: Índice Centros, Área y Radio. Este ComboBox al ser clicando manda a llamar la función de colorear circulo con la ayuda del elemento que se selecciona del ComboBox, así es posible pasarle por parámetros el inicio de x y el centro de y. El circulo seleccionado se colorea de rojo.

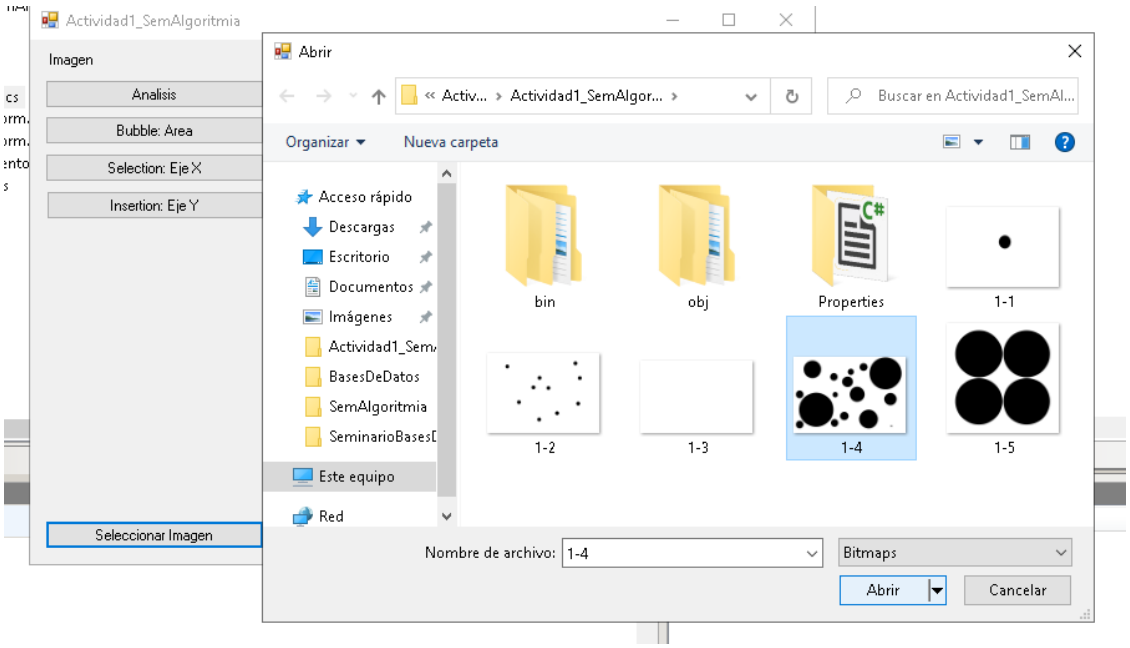
Por ultimo implemente los métodos de ordenamiento, para esto realice una clase Ordenamientos que contiene los tres métodos usados, Bubble sort, selection sort e insertion sort. Solamente coloque tres botones respectivos a cada método de ordenamiento y en su método de evento llame a la función determinada de la clase Ordenamientos. Les paso por parámetros la Lista dinámica de círculos y ya dentro de la clase la ordeno respecto al ordenamiento seleccionado.

Pruebas y resultados.

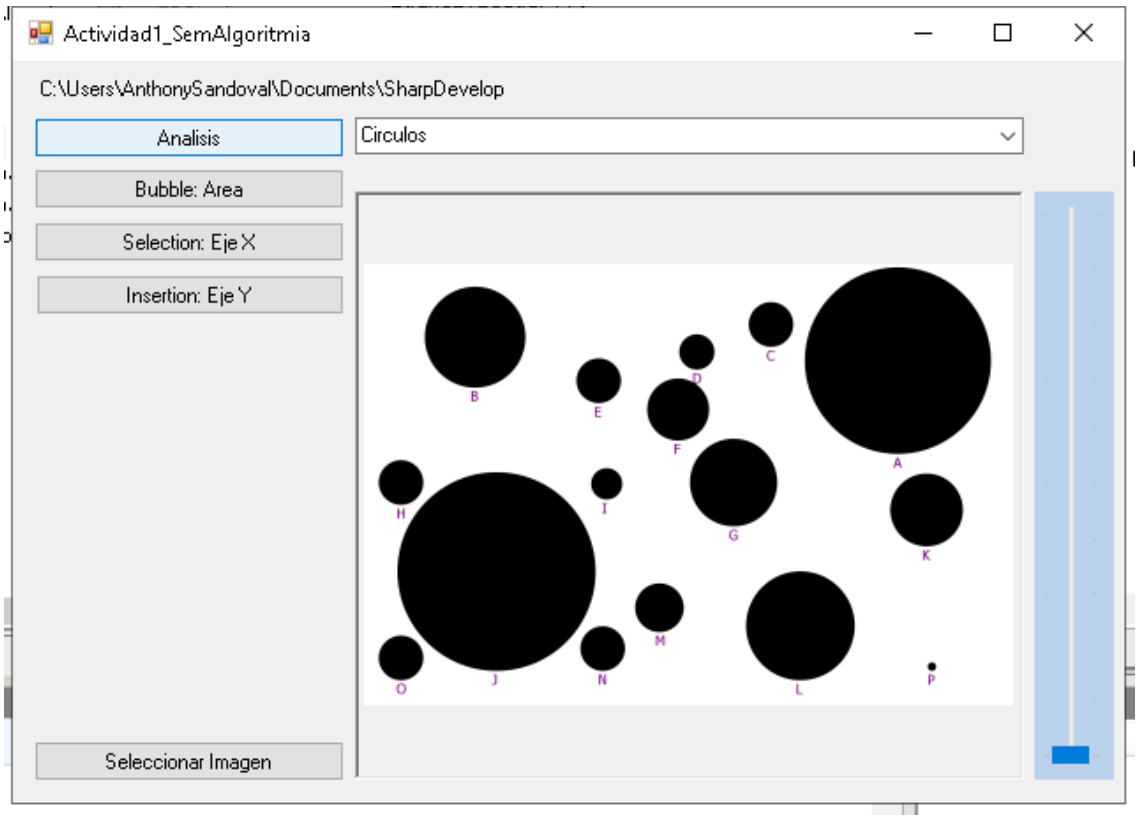
Interfaz principal.



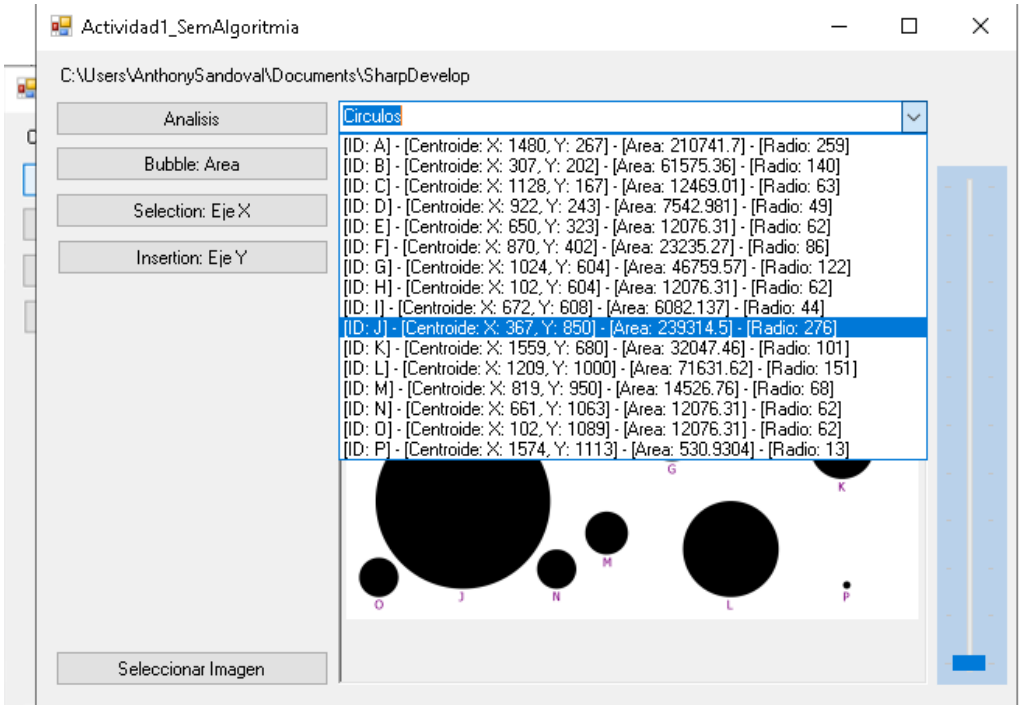
Clic en el botón de “Seleccionar Imagen”. Se abre la ventana para escoger la imagen



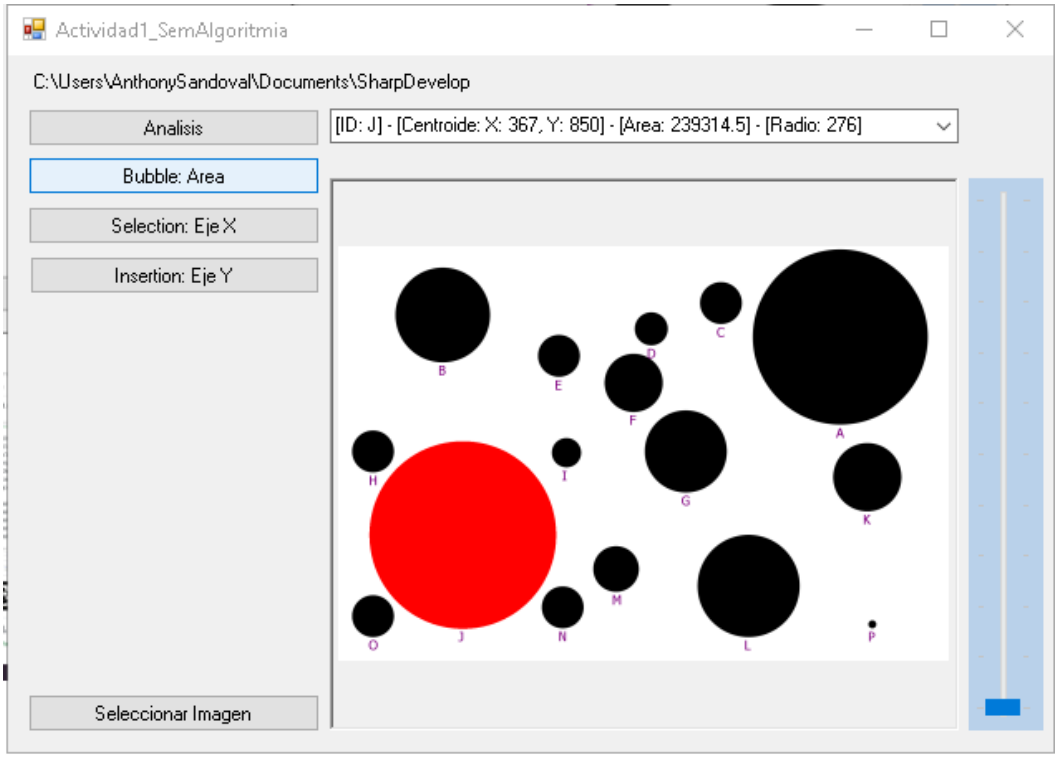
Clic en el boton de “Analisis”. Comienza la busqueda de Centros y etiqueta a los circulos.



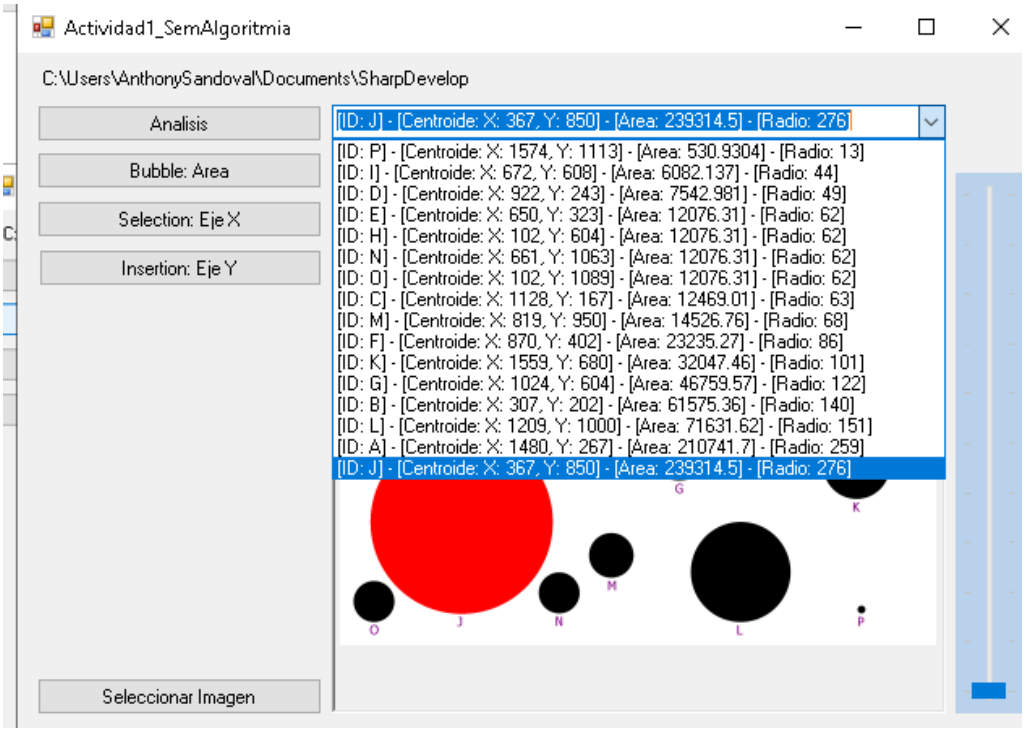
También se carga el combobox con los círculos encontrados. Se selecciona el circulo J.



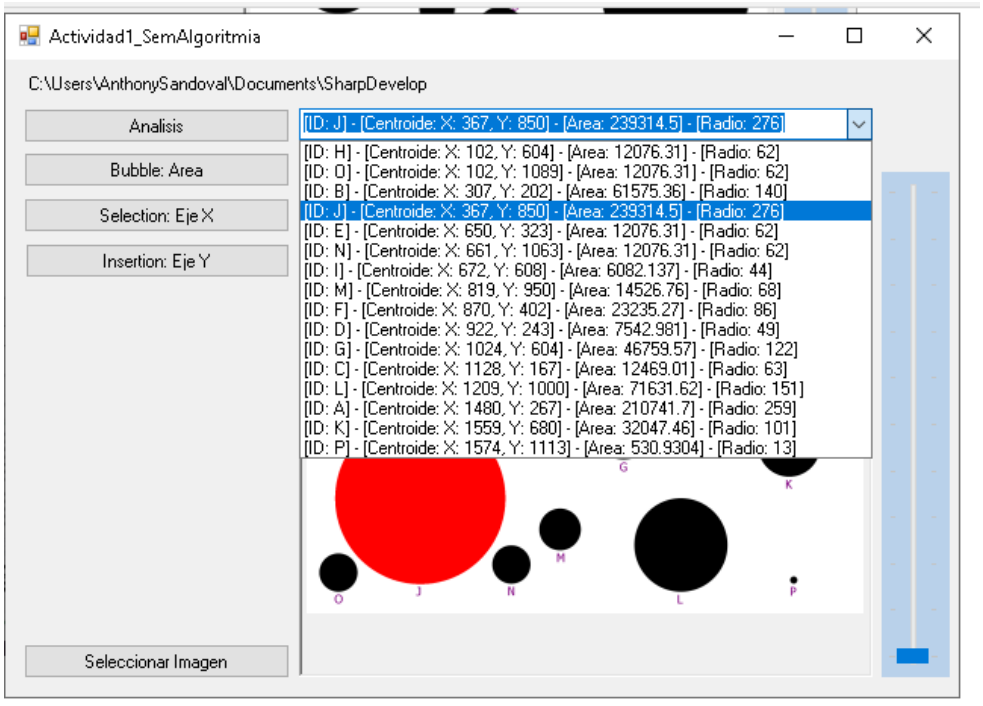
Se colorea el circulo seleccionado de rojo. Ahora clic en el boton Bubble



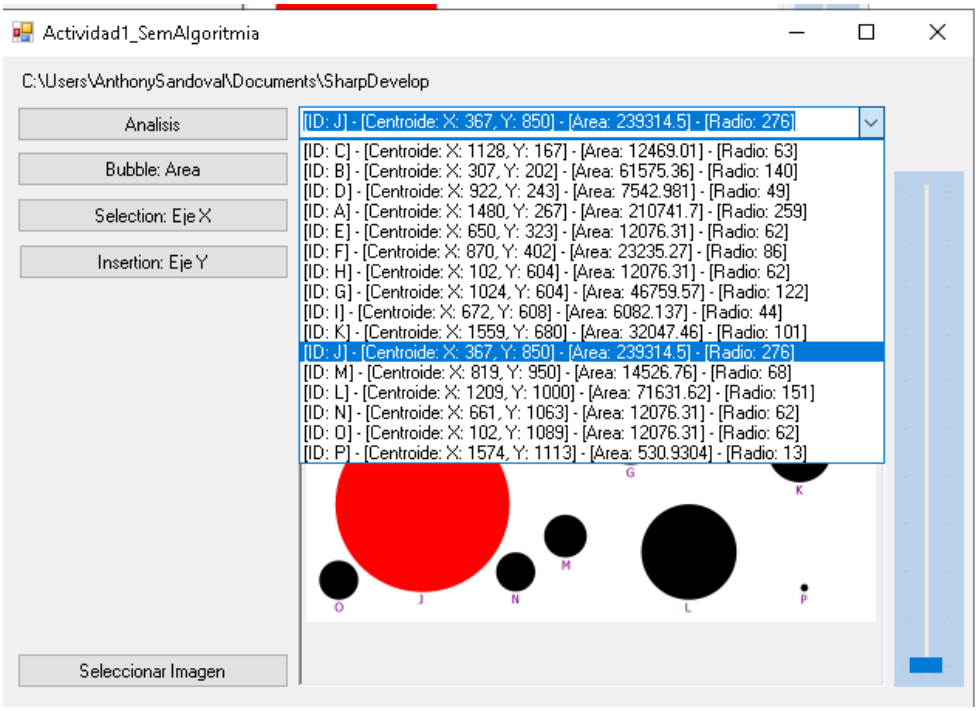
Los círculos en el combobox ahora están ordenados respecto al área. J cambio de posición, es el más grande de los círculos.



Aquí se dio clic en el botón Selection, se ordenan los círculos del combobox respecto al Eje X. J volvió a cambiar de posición.



Aquí se dio clic en el botón Insetion, se ordenan los círculos del combobox respecto al Eje Y. J volvió a cambiar de posición.



Conclusiones.

Antes ya había usado bastante Java para crear interfaces gráficas, con el uso de C# me di cuenta que casi lo mismo al momento de programar, debido a ello me fue sencillo comprender la forma de programar esta actividad, además de que de por sí creo que es más sencillo la programación orientada a eventos que todas las demás. Lo que me pareció más confuso de todo fue encontrar el centro del círculo, ya que no me sentía familiarizado con el manejo de pixeles en imágenes, nunca había practicado algo parecido, sin embargo, cuando termine esa parte que se me dificulto ya lo demás fue más sencillo de comprender. Esta actividad me pareció entretenida, tarde bastante tiempo en hacerla, pero en ningún momento me desagrado, espero que las demás sean así.

Apendice(s).

Método para abrir imagen

```
void seleccionarImagen(){
        OpenFileDialog opF = new OpenFileDialog();
        opF.Filter = "Bitmaps|*.png|jpeps|*.jpg";

        cBox_Circulos.Items.Clear();
        if(opF.ShowDialog() == DialogResult.OK){
           imagen_PBox.ImageLocation = opF.FileName;
           //imagen_PBox.Image = Bitmap.FromFile(opF.FileName);
           c1 = opF.FileName;
           ruta.Text = opF.FileName;
        }
     }
```

Método para buscar círculos

```
void buscarCirculos(int comienzoX, int comienzoY){


        Bitmap bmp = new Bitmap(c1);
        imagen_PBox.Image = bmp;
        Color colorActual;
        int x, y;

        Debug.WriteLine("\nBuscando Circulos...");
        for(y = comienzoY; y < bmp.Height; y++){
           for(x = comienzoX; x < bmp.Width; x++){
              colorActual  = bmp.GetPixel(x, y);
              if(colorActual.R != 255 && colorActual.R != Color.Red.R){
                 if(colorActual.G != 255 && colorActual.G != Color.Red.G){
                    if(colorActual.B != 255 && colorActual.B != Color.Red.B){
                       Debug.WriteLine("Circulo Encontrado");
                       buscarCentro(x, y, bmp);
                    }
                 }
              }
           }
        }
        bmp = new Bitmap(c1);
        imagen_PBox.Image = bmp;
        etiquetasCirculos(35, 50, bmp, 30);
     }
```

Método para buscar el centro de un circulo

```
void buscarCentro(int X, int Y, Bitmap bmp){

        int norteY=0, surY=0;
```

```
int oesteX=0, esteX=0;
int centroX=0, centroY=0;

Color colorActual;

Debug.WriteLine("Buscando Centroide...");
/* Y */
//Sur (++)
surY = Y;
colorActual = bmp.GetPixel(X, surY);
while (colorActual.R != 255 && colorActual.G != 255 && colorActual.B != 255) {
   colorActual  = bmp.GetPixel(X, surY);
   surY++;
}
surY-=1;
//Norte (--)
norteY = Y;
norteY--;
colorActual  = bmp.GetPixel(X, norteY);
while (colorActual.R != 255 && colorActual.G != 255 && colorActual.B != 255) {
   colorActual  = bmp.GetPixel(X, norteY);
   norteY--;
}
norteY+=1;
//Centro en Y
centroY = norteY + ((surY - norteY)/2);

/* X */
//Este (++)
esteX = X;
colorActual  = bmp.GetPixel(esteX, centroY);
while(colorActual.R != 255 && colorActual.G != 255 && colorActual.B != 255){
   colorActual  = bmp.GetPixel(esteX, centroY);
   esteX++;
}
esteX-=1;
//Oeste (--)
oesteX = X;
oesteX--;
colorActual  = bmp.GetPixel(oesteX, centroY);
while(colorActual.R != 255 && colorActual.G != 255 && colorActual.B != 255){
   colorActual  = bmp.GetPixel(oesteX, centroY);
   oesteX--;
}
oesteX+=1;

centroX = oesteX + ((esteX - oesteX)/2);
```

```
        Debug.WriteLine("Centroide Encontrado [x, y]: " + centroX + ", " + centroY);
        circulo = new Circulo(centroX, centroY, oesteX, esteX, identificador);
        identificador++;
        circulos.Add(circulo);

        Debug.WriteLine("Area del circulo: " + circulo.getArea());

        Debug.WriteLine("Coloreando Circulo...");
        colorearRespecto_X(circulo.getIniX(), circulo.getCentroY(), bmp);

    }
```

Método para colorear un circulo

```
void colorearRespecto_X(int inicioX, int centroY, Bitmap bmp){
        Color c;
        int x, y;

        x =  inicioX;
        x++;
        c  = bmp.GetPixel(x, centroY);
        bmp.SetPixel(x, centroY, Color.Red);
        while(c.R != 255 && c.G != 255 && c.B != 255){

          y = centroY;
          y--;
          c  = bmp.GetPixel(x, y);
          while(c.R != 255 && c.G != 255 && c.B != 255){
            c  = bmp.GetPixel(x, y);
            bmp.SetPixel(x, y, Color.Red);
            y--;
          }

          y = centroY;
          y++;
          c  = bmp.GetPixel(x, y);
          while(c.R != 255 && c.G != 255 && c.B != 255){
            c  = bmp.GetPixel(x, y);
            bmp.SetPixel(x, y, Color.Red);
            y++;
          }
          bmp.SetPixel(x, centroY, Color.Red);
          x++;
          c  = bmp.GetPixel(x, centroY);

        }
        Debug.WriteLine("Circulo Coloreado");
```

```
    }
```

Métodos para poner etiquetas a los círculos

```
void etiquetasCirculos(int width, int heigth, Bitmap bmp, int tamLetra){
        for(int i = 0; i < circulos.Count; i++){
          ponerEtiqueta(circulos[i].getCentroX()-
18, (circulos[i].getCentroY() +  Convert.ToInt32(circulos[i].getRadio())), width, heigth, bmp, circulo
s[i].getId(), tamLetra);
        }
    }

void ponerEtiqueta(int x, int y, int width, int height, Bitmap bmp, char id, int tamLetra){
        RectangleF rectf = new RectangleF(x, y, width, height);

        Graphics g = Graphics.FromImage(bmp);

        g.SmoothingMode = SmoothingMode.AntiAlias;
        g.InterpolationMode = InterpolationMode.HighQualityBicubic;
        g.PixelOffsetMode = PixelOffsetMode.HighQuality;
        g.DrawString(Char.ToString(id), new Font("Tahoma", tamLetra), Brushes.Purple, rectf);

        g.Flush();
    }
```

Método para llenar el ComboBox

```
void cargarComboBox(){
        for(int i = 0; i < circulos.Count; i++){
          cBox_Circulos.Items.Add("[ID: " + circulos[i].getId() + "] - [Centroide: X:
" + circulos[i].getCentroX() + ", Y: " + circulos[i].getCentroY() +
                                "] - [Area: " + circulos[i].getArea() + "] - [Radio:
" + circulos[i].getRadio() + "]");
        }
    }
```

Método para el botón de Análisis

```
void Btn_AnalisisClick(object sender, EventArgs e)
    {
        identificador = 'A';
        circulos.Clear();
        cBox_Circulos.Items.Clear();
        buscarCirculos(0, 0);
        cargarComboBox();
    }
```

Métodos para los botones de ordenamientos.

```
//Ordenamientos
    void Btn_BurbujaClick(object sender, EventArgs e)
```

```
    {
        Ordenamientos burbuja = new Ordenamientos();
        cBox_Circulos.Items.Clear();

        circulos = burbuja.bubbleSort_Area(circulos);
        cargarComboBox();
    }

    void Btn_SelectionClick(object sender, EventArgs e)
    {
        Ordenamientos burbuja = new Ordenamientos();
        cBox_Circulos.Items.Clear();

        circulos = burbuja.seleccionSort_EjeX(circulos);
        cargarComboBox();
    }

    void Btn_InsertionClick(object sender, EventArgs e)
    {
        Ordenamientos burbuja = new Ordenamientos();
        cBox_Circulos.Items.Clear();

        circulos = burbuja.insetionSort_EjeY(circulos);
        cargarComboBox();
    }
```

Clase Circulo

```
public class Circulo
  {
    private int centroX;
    private int centroY;
    private int inicioX;
    private int finX;
    private float areaCirculo;
    private char identificador;
    private float radio;

    public Circulo(int cX, int cY, int iniX, int fX, char id)
    {
        this.centroX = cX;
        this.centroY = cY;
        this.inicioX = iniX;
        this.finX = fX;
        this.identificador = id;
        calcularArea();
    }

    private void calcularArea(){
```

```
        radio = (finX - inicioX) / 2;
        areaCirculo = 3.1416f * (radio*radio);
    }
    public float getArea(){ return areaCirculo;}
    public int getCentroX(){ return centroX;  }
    public int getCentroY(){  return centroY; }
    public int getIniX(){ return inicioX;}
    public int getFinX(){ return finX;}
    public char getId(){return identificador; }
    public float getRadio(){ return radio;}
  }
```

Clase Ordenamientos

```
public class Ordenamientos
  {
    public Ordenamientos()
    {
    }

    public List<Circulo> bubbleSort_Area(List<Circulo> circulos){

      Circulo aux;

      for(int i = 0; i < circulos.Count - 1; i++){
        for(int j = 0; j  < circulos.Count - 1; j++){
          if(circulos[j+1].getArea() < circulos[j].getArea()){
            aux = circulos[j+1];
            circulos[j+1] = circulos[j];
            circulos[j] = aux;
          }
        }
      }

      return circulos;
    }


    public List<Circulo> seleccionSort_EjeX(List<Circulo> circulos){
      int min;

      for (int i = 0; i < circulos.Count-1; ++i)
      {
        min = i;
        for (int j = (i + 1); j < circulos.Count; ++j)
        {
          if (circulos[min].getCentroX() > circulos[j].getCentroX())//lista[j] < lista[min]
          {
            min = j;
```

```
            }
        }
        Circulo aux;
        aux = circulos[i];
        circulos[i] = circulos[min];
        circulos[min] = aux;
    }

    return circulos;
}


public List<Circulo> insetionSort_EjeY(List<Circulo> circulos){
    Circulo v;
    int j;

    for(int i = 1; i < circulos.Count; i++){
        v = circulos[i];
        j = i - 1;
        while(j >= 0 && circulos[j].getCentroY() > v.getCentroY()){
            circulos[j+1] = circulos[j];
            j = j - 1;
        }
        circulos[j + 1] = v;
    }

    return circulos;
}

}
```