Design And Analysis Algorithms Course

# Instructor : Dr. Ahmed Salah

**Image Quantization Project**

**14th December 2019**

Team Members :-

* Amina Ahmed Abounawara (4) (Team Leader)
* Dina Abdalla Said (6)
* Habiba Khaled Mohammed (6)
* Abdelrahman Hegazy Atiya (9)
* Abdelrahman Mohammed Mohamed Abdullah (10)

# Introduction :

The goal of this project is to reduce the number of colors in a full resolution digital color image (24 bits per pixel) to a smaller set of representative colors called ***color palette*.** Reduction should be performed so that the quantized image differs as little as possible from the original image. Algorithmic optimization task is to find such a color palette that the overall distortion is minimized.

# Uses:

1. It can be used in image compression by reducing the colors number without distorting the image.
2. It can be used in displaying images on devices with limited number of colors.
3. It processes extracting useful objects from an image.

# Technical Details :

Quantization used in reducing colors by replacing the nearest colors with an average color which reduces the different color representation in the image and this may distort the image in some cases. This project is divided into classes, The **Main** class is **“ImageOperations”** class which contains all the operations done on the image starting from opening the image, compressing it and finally showing it on the screen. The image colors (RGB) are imported in a 2d array called ImageMatrix.

# This project is implemented in C#.

# Going Through Project:

# “ImageQuantization.cs”.

* RGBPixel struct.
* RGBPixelD struct.
* ImageOperations class.
* public static RGBPixel[,] OpenImage(string ImagePath)
* public static int GetHeight(RGBPixel[,] ImageMatrix)
* public static int GetWidth(RGBPixel[,] ImageMatrix)
* public static void DisplayImage(RGBPixel[,] ImageMatrix, PictureBox PicBox)
* public static RGBPixel[,] GaussianFilter1D(RGBPixel[,] ImageMatrix, int filterSize, double sigma)

# “ImageAnalytics.cs”

* ImageAnalytics class.
* public static long Find\_Distinct\_Color(RGBPixel[,] ImageMatrix)
* public static double MinimumSpanningTree()

# “FibbonacciHeap.cs”

* FibbonacciHeap class.
* public FibbonacciHeap(int capacity)
* public void Insert(HeapNode NewNode)
* public void decrease\_key(int indx, double value)
* public void Cut(HeapNode node, HeapNode parent)
* public void Cascase\_cut(HeapNode node)
* public void mergeTrees(HeapNode max, HeapNode min)
* public void Consolidate()
* public HeapNode Extract\_min()
* public void swap(ref HeapNode X, ref HeapNode Y)

# “HeapNode.cs”

* HeapNode class.
* public HeapNode(int key,double value)

# “MergeSort.cs”

* MergeSort class.
* public static List<int> Sort(List<int> unsorted)
* private static List<int> Merge(List<int> left, List<int> right)

# “AtkinsonDither.cs”

* AtkinsonDither class.
* public static RGBPixel[,] Atkinson\_Dithering(RGBPixel[,] ImageMatrix, int factor)
* static void gray\_scale(ref byte R, ref byte B, ref byte G)

# “FloyedDither.cs”

* FloyedDither class.
* public static RGBPixel[,] Floyed\_Dithering(int factor, RGBPixel[,] Filltered)
* public static void Scale()

# “GaussianBlur.cs”

* GaussianBlur class.
  + public GaussianBlur(Bitmap image)
  + public Bitmap Process(int radial)
  + private void gaussBlur\_4(int[] source, int[] dest, int r)
  + private int[] boxesForGauss(int sigma, int n)
  + private void boxBlur\_4(int[] source, int[] dest, int w, int h, int r)
  + private void boxBlurH\_4(int[] source, int[] dest, int w, int h, int r)
  + private void boxBlurT\_4(int[] source, int[] dest, int w, int h, int r)

# “MedianFilter.cs”

* MedianFilter class.
  + public static void Median\_Filter(Bitmap image, int Factor, ref Bitmap RezImage)

# “DetectNumOfClusters.cs”

* DetectNumOfClusters class.
* public DetectNumOfClusters(List<Edge> edges)
* double calculateMean()
* double calculateStandardDeviation()

# “QuantizeImage.cs”

* QuantizeImage class.
* public static void Extract\_K\_Cluster(int K)
* private static void BFS(int Color)
* public static void Find\_K\_Cluster()
* public static void Quantize\_Image()

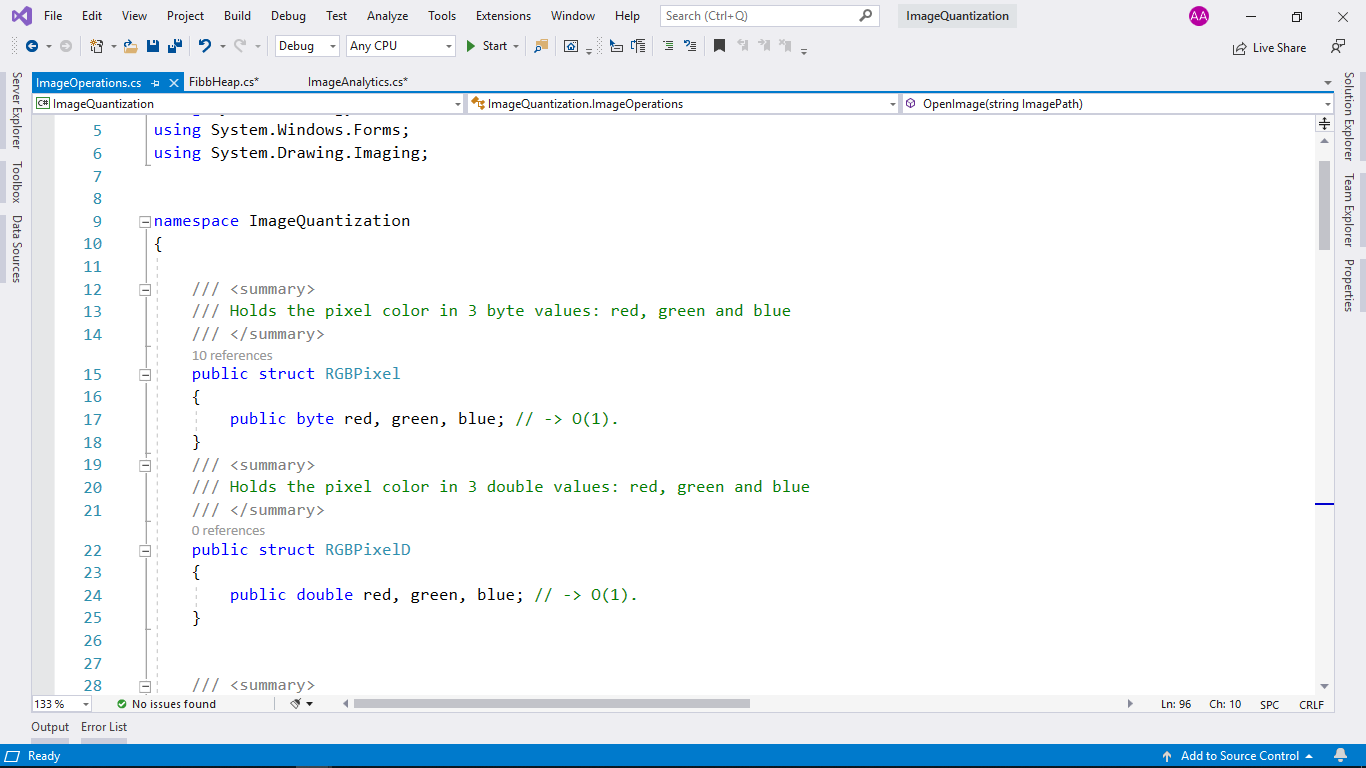
# “Edge.cs”

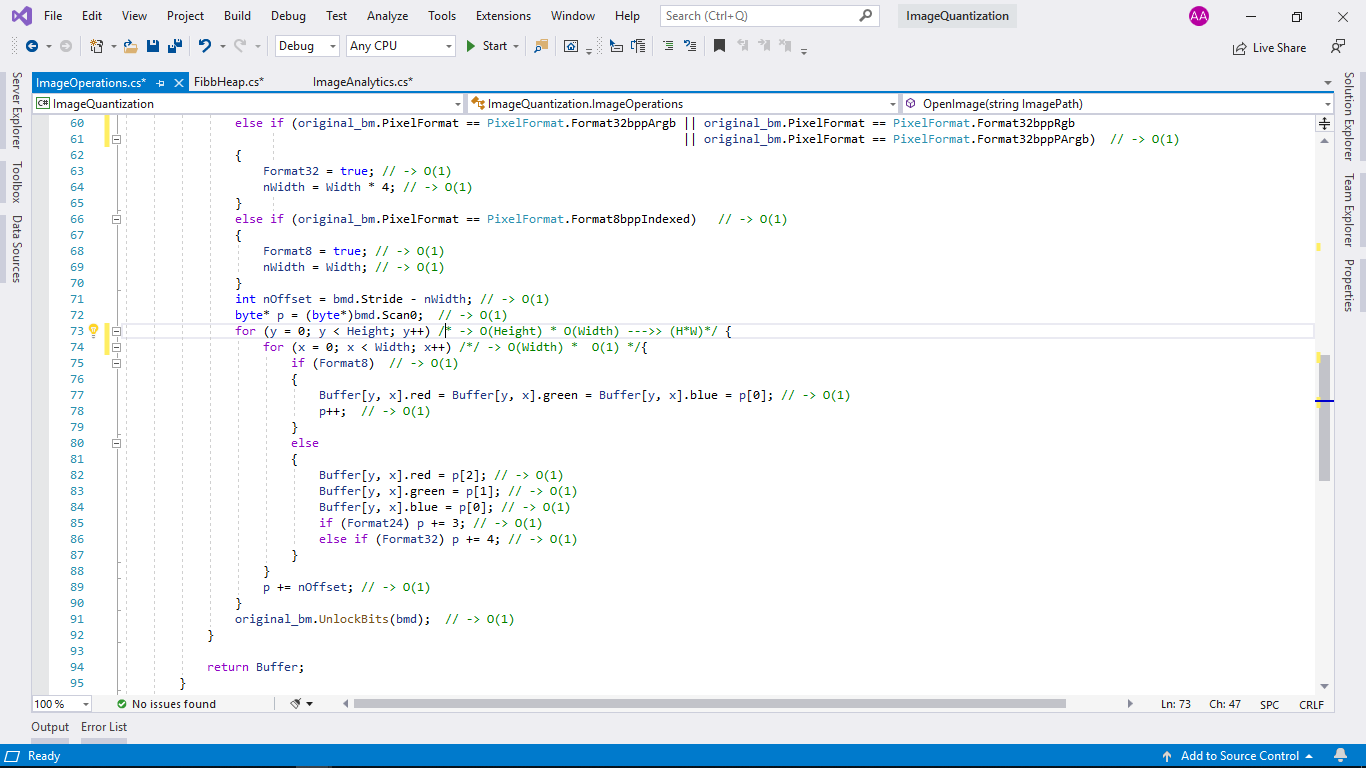
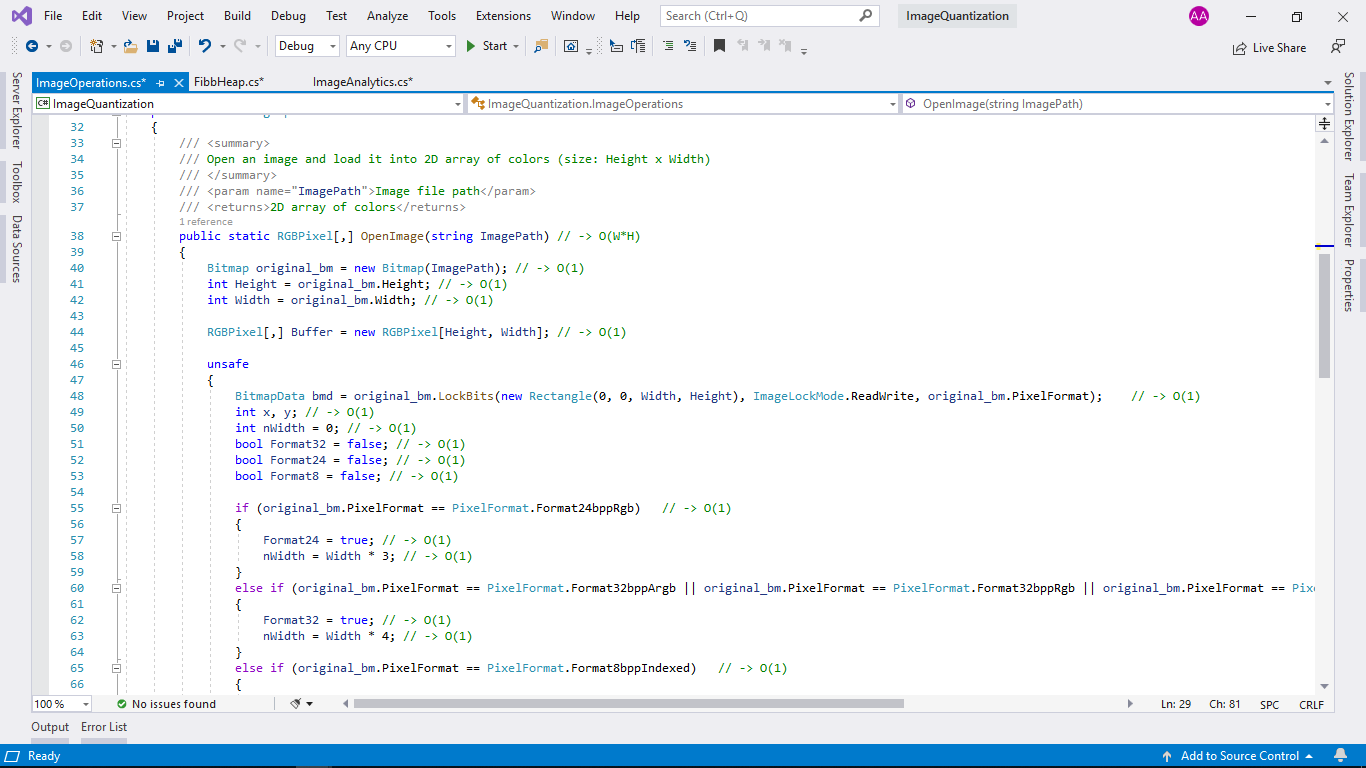
* Edge class.
* public Edge()
* public Edge(int source, int destionation, double weight)

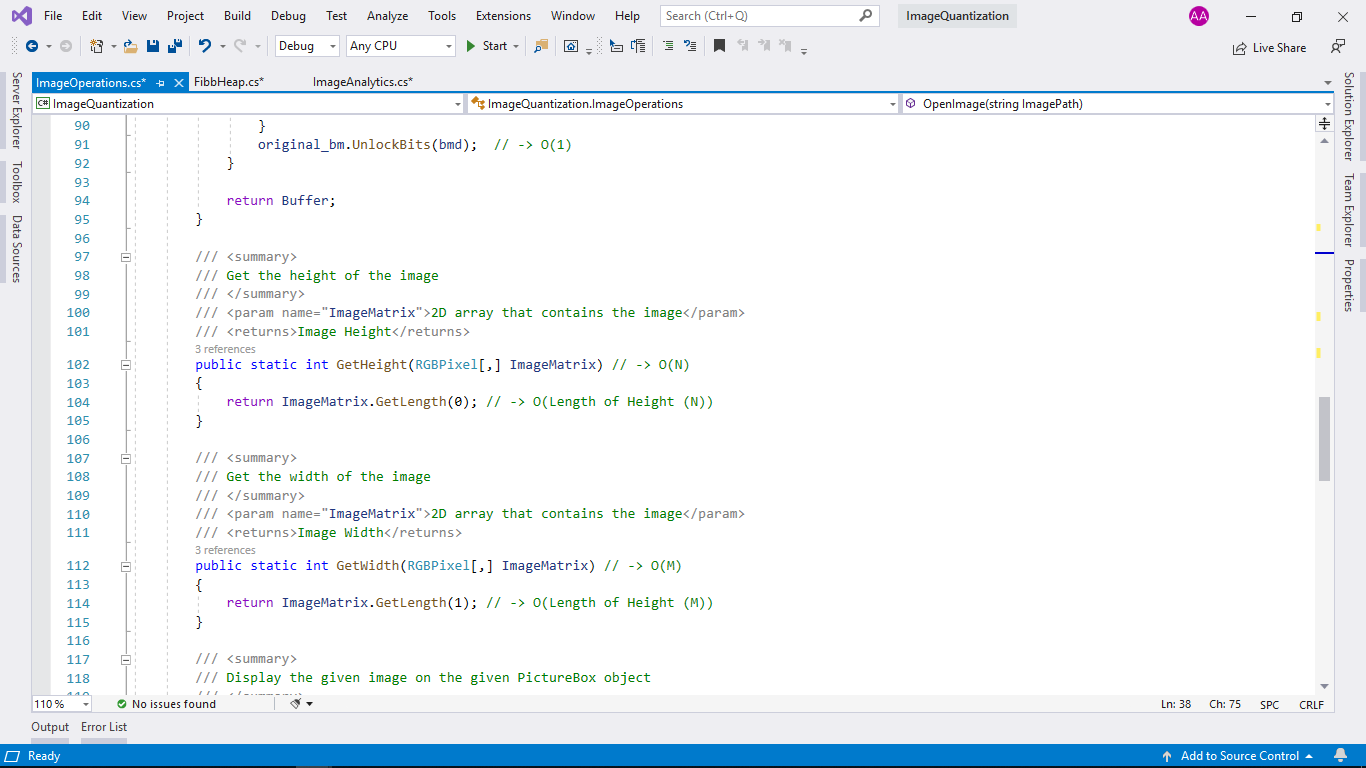
# Analysis Of Code:

# “ImageQuantization.cs”.

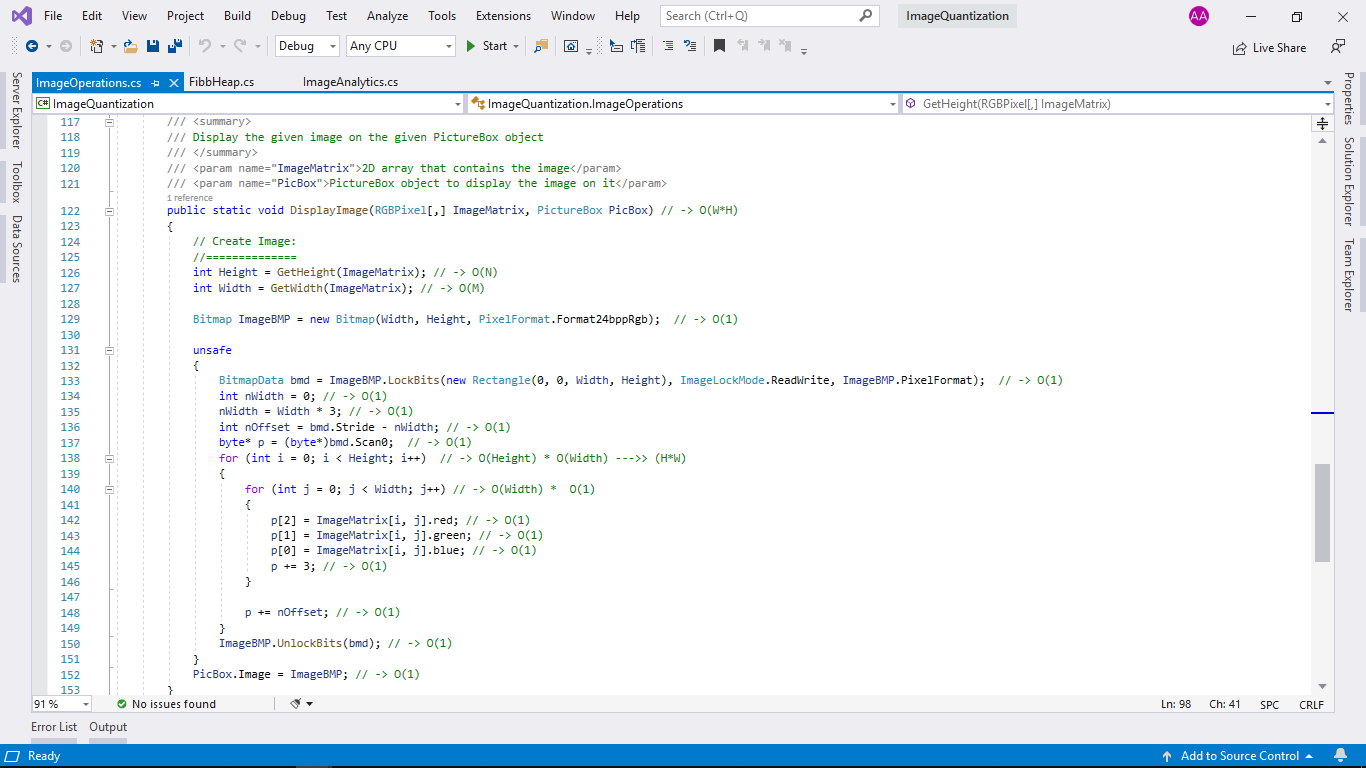
* RGBPixel struct.
* RGBPixelD struct.



* **Description : RGBPixel** which represents each pixel of the image in 3 values(RED, GREEN,BLUE) Once in byte value and again in double value.
* **Time Complexity : Θ(1).**
* ImageOperations class.
* **Description :** Library of static functions that deal with images (All Image Operations)
* public static RGBPixel[,] OpenImage(string ImagePath)
* **Description :** Open an image and load it into 2D array of colors (size: Height x Width)
* **Time Complexity : Θ(Width\*Height).**
* public static int GetHeight(RGBPixel[,] ImageMatrix)
* public static int GetWidth(RGBPixel[,] ImageMatrix)



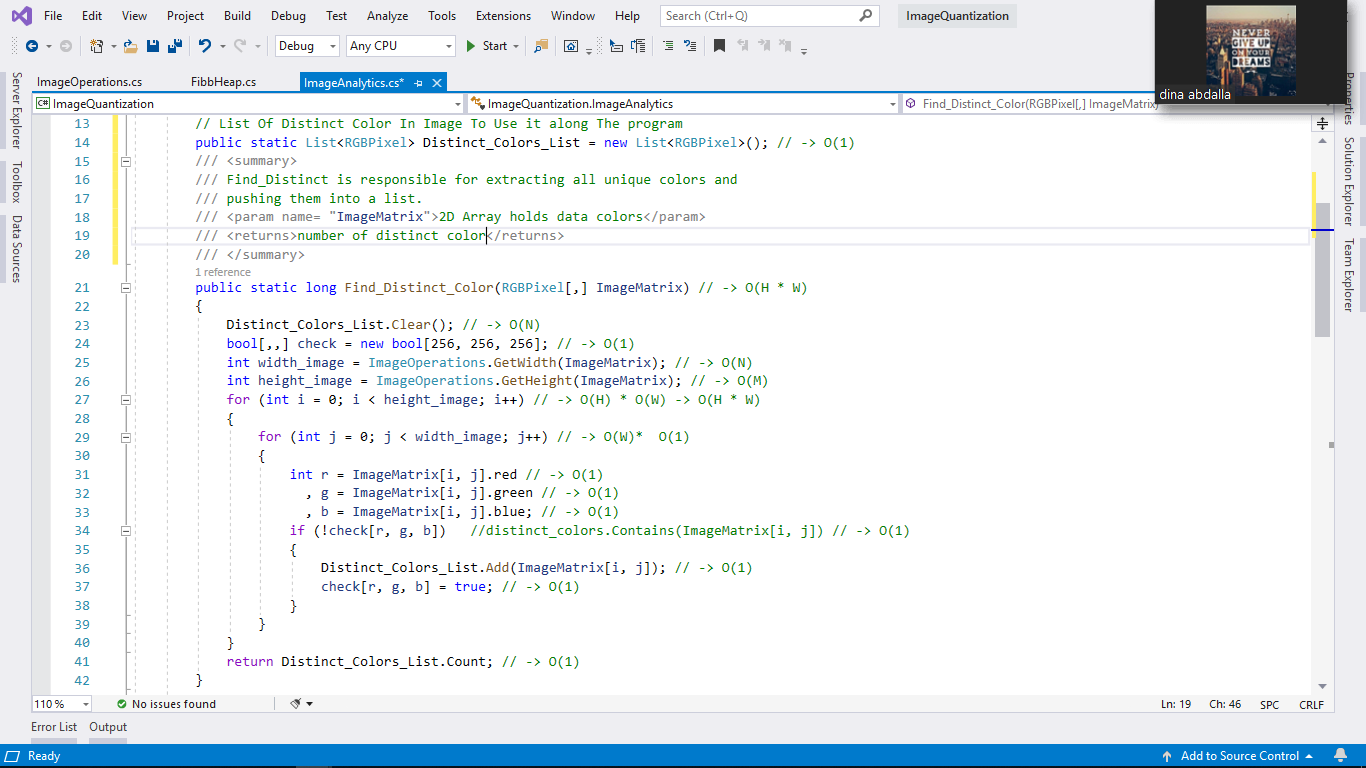
* **Description :** Get the height and weight of the image.
* **Time Complexity : GetHeight Θ(W) & GetWeight Θ(H) .**
* public static void DisplayImage(RGBPixel[,] ImageMatrix, PictureBox PicBox)

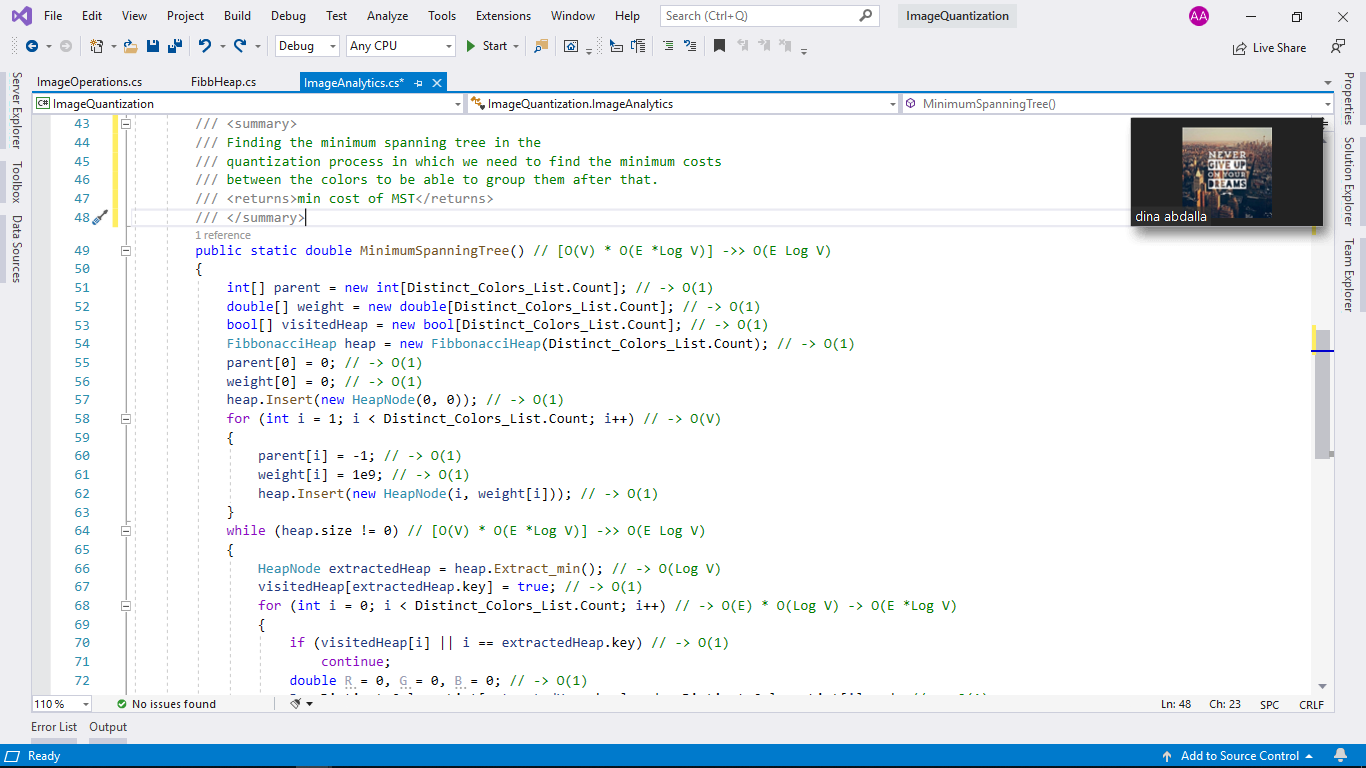
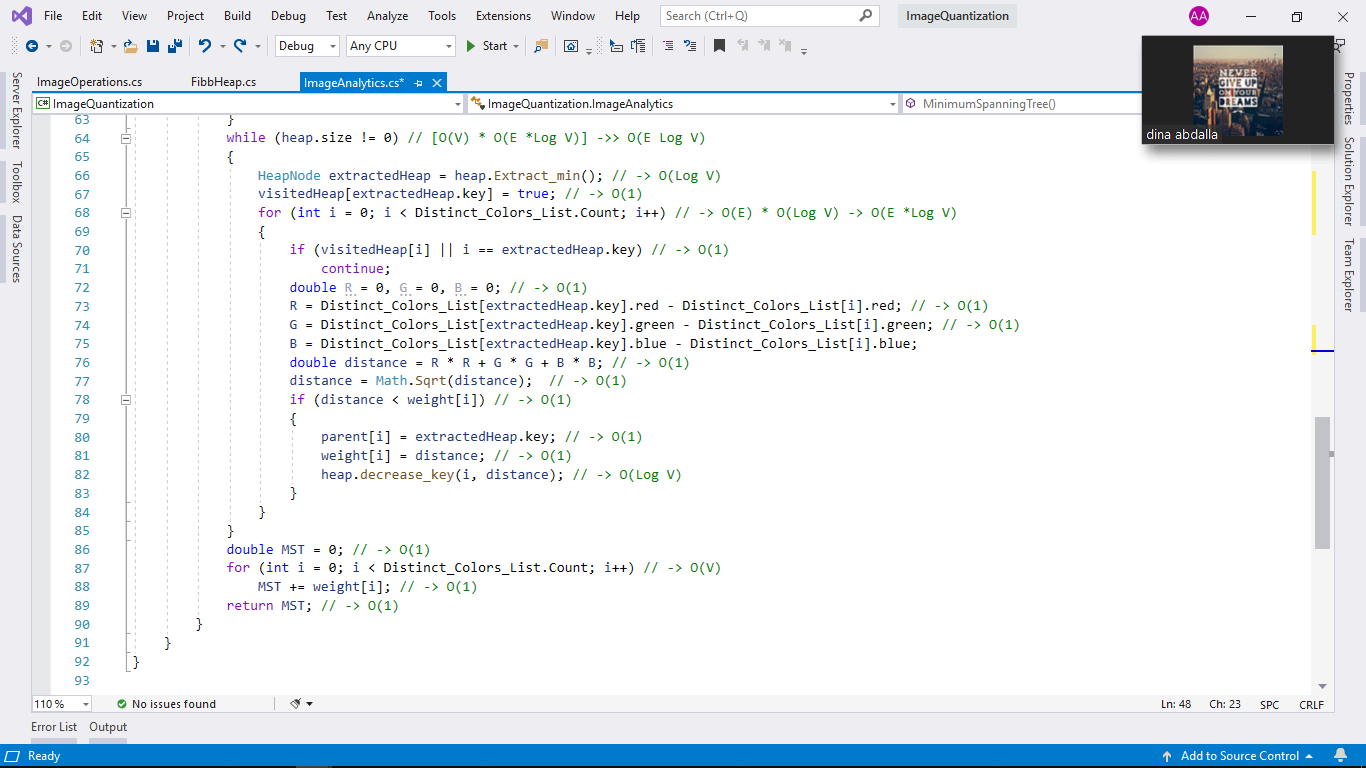


* **Description :** Display the given image[2D array of colors] on the given PictureBox object.
* **Time Complexity : Θ(W\*H).**

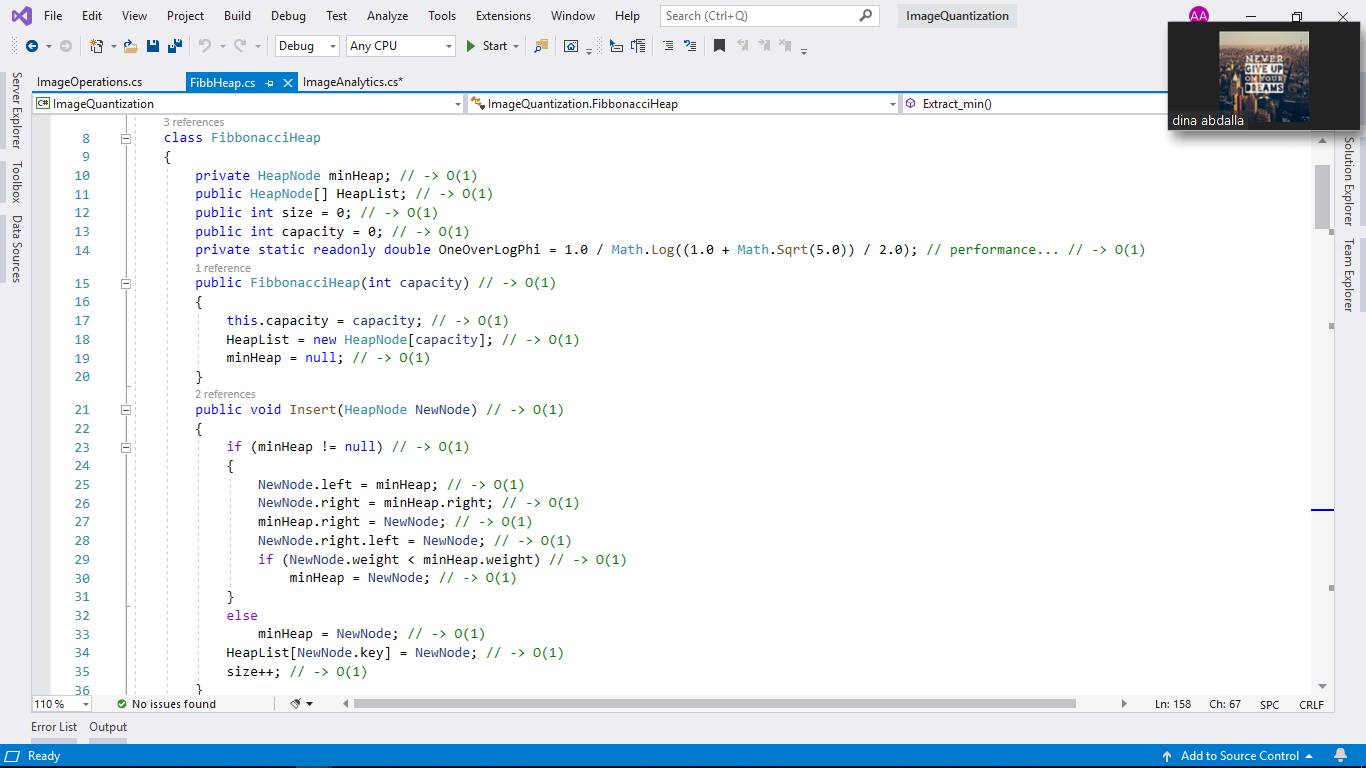
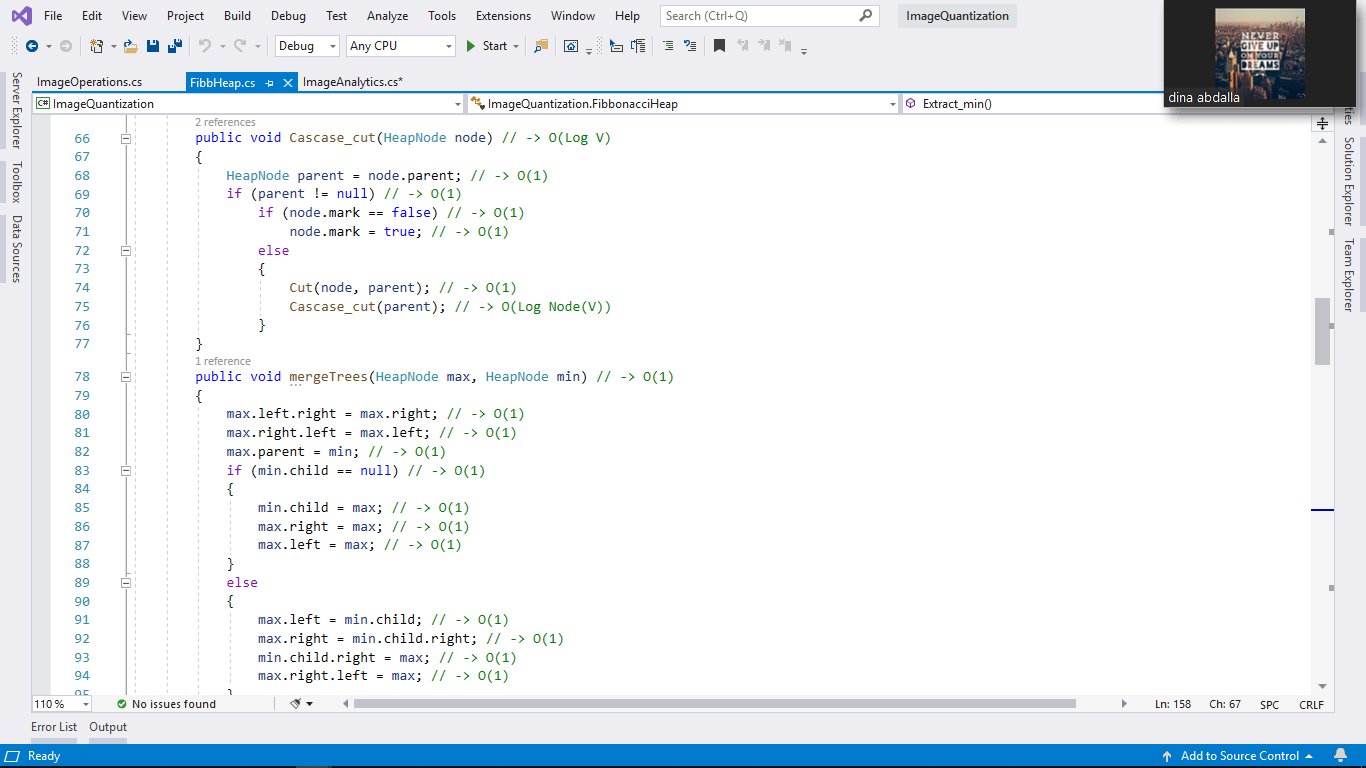
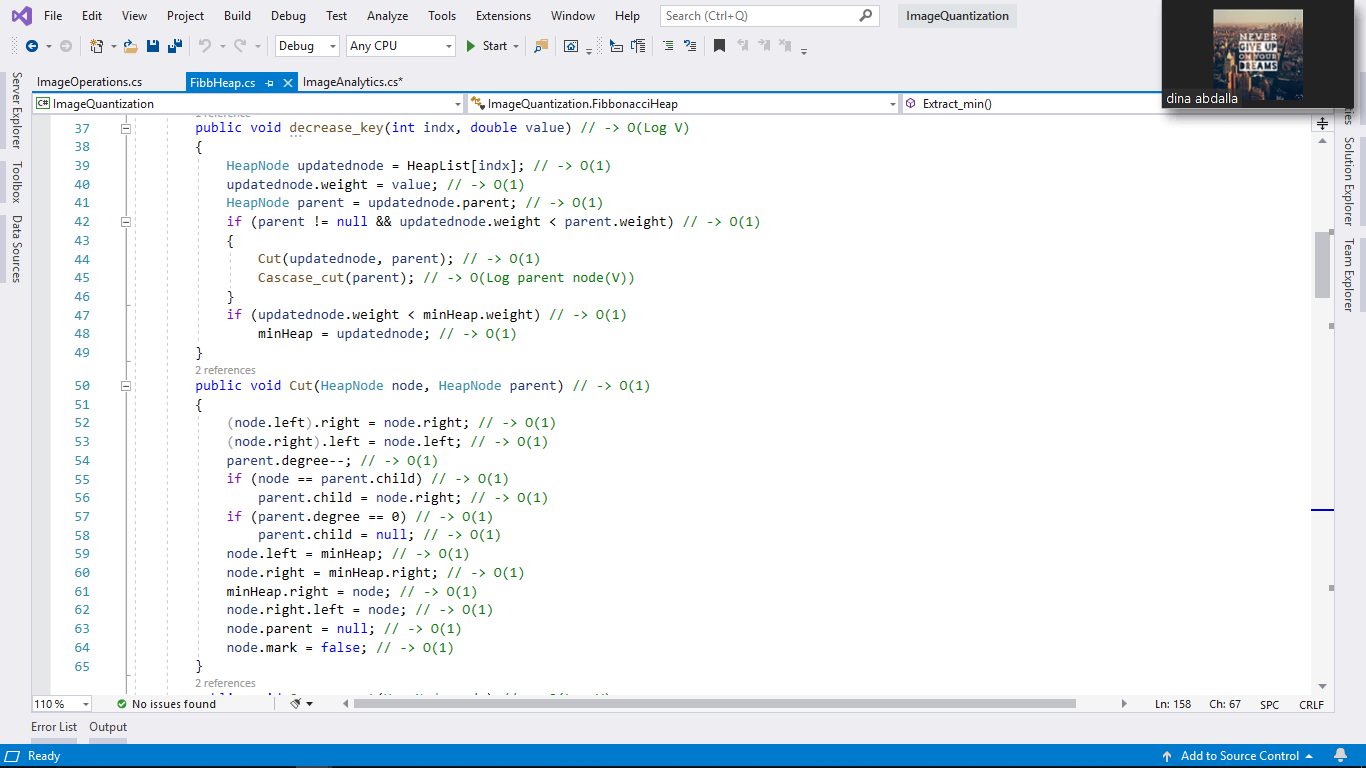
# “ImageAnalytics.cs”

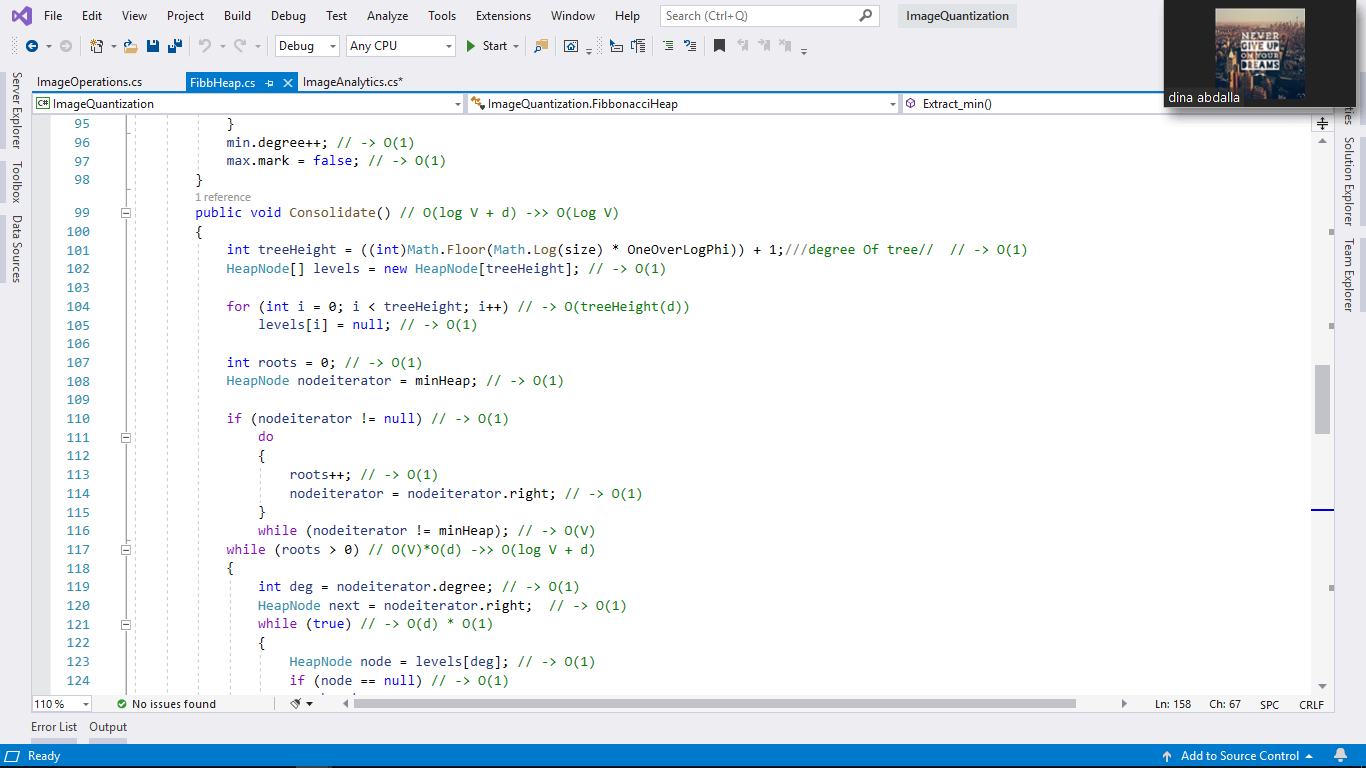
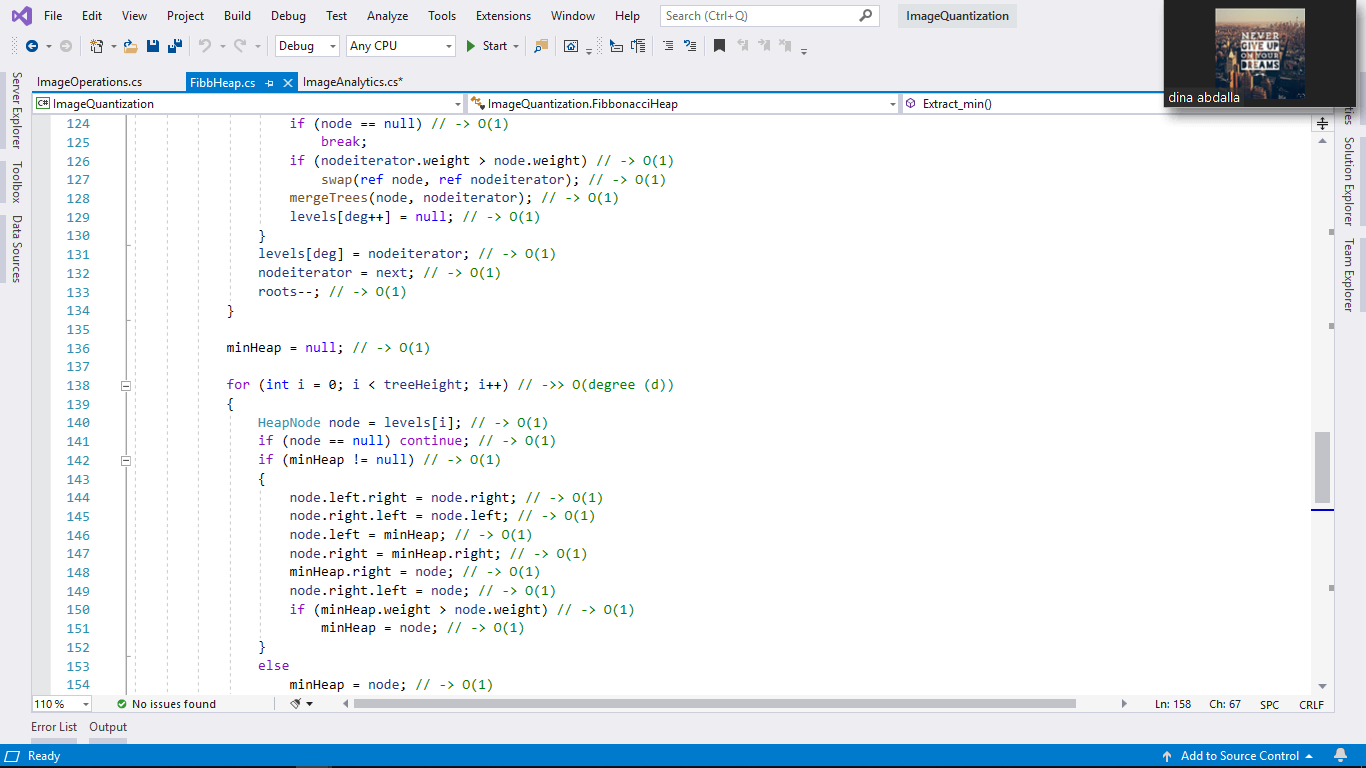
* ImageAnalytics class.
* **Description** : Library of static functions Its goal is to get the Quantized Image in an efficient way.
* public static long Find\_Distinct\_Color(RGBPixel[,] ImageMatrix)

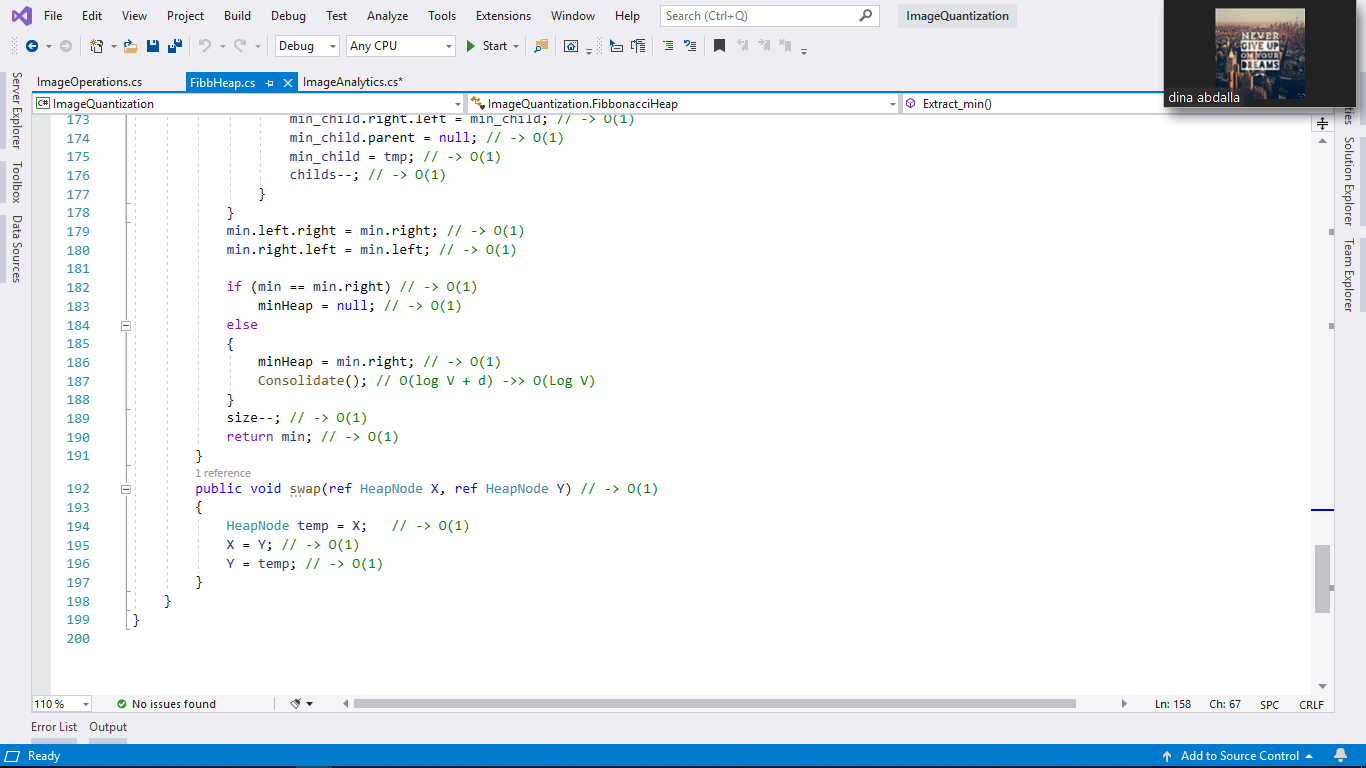
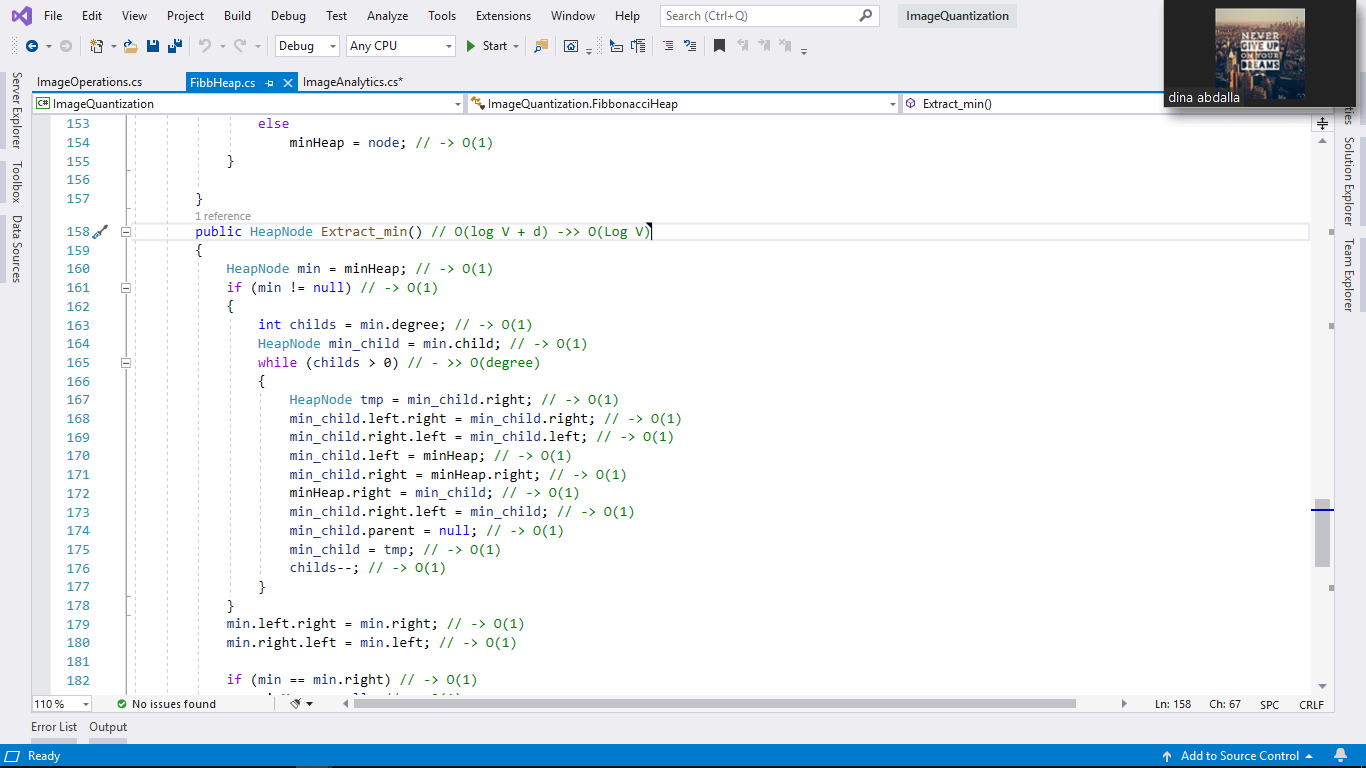


* **Description :**  responsible for extracting all unique colors and pushing them into a list.
* **Time Complexity : Θ(W\*H).**
* public static double MinimumSpanningTree()
* **Description :** Finding the minimum spanning tree in the quantization process in which we need to find the minimum costs between the colors to be able to group them after that using Fibonacci heap.
* **Time Complexity : Θ(E Log V).**

# “FibbonacciHeap.cs”







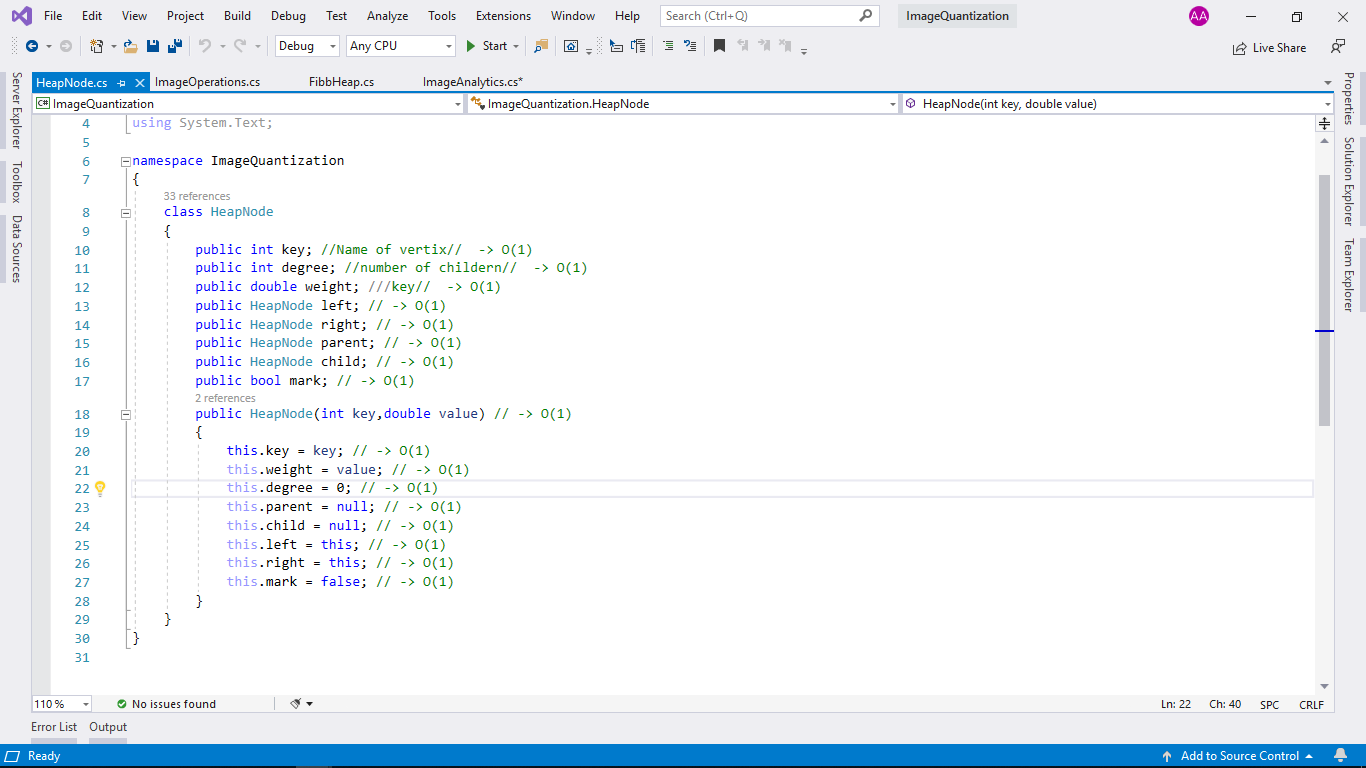
# Summary Of Fibonacci Heap :

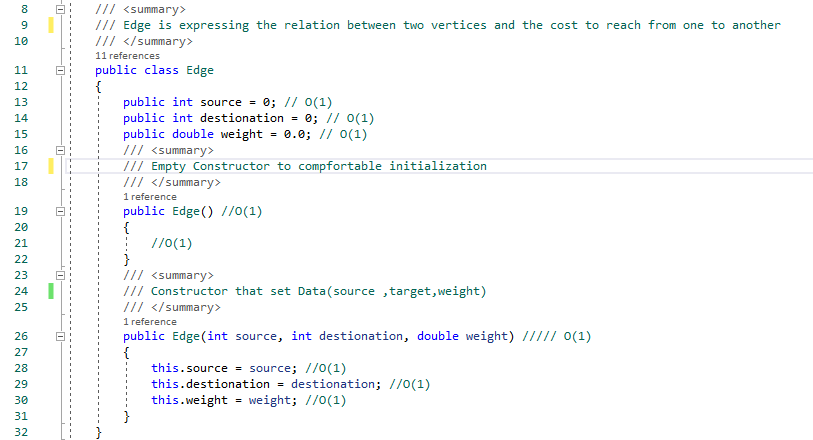
|  |  |  |
| --- | --- | --- |
| Inseart () | Insert a node in a Fibonacci heap. | O(1) |
| decrease\_key() | To decrease the value of any element in the heap. | O(Log V) |
| Cut() | Using in decrease\_key(). | O(1) |
| Cascase\_cut() | Using in decrease\_key(). | O(Log V) |
| mergeTrees() | Union of two Fibonacci heaps . | O(1) |
| Consolidate() | Using in Extract\_min(). | O(Log V) |
| Extract\_min() | deleting the minimum node and setting the min pointer to the minimum value in the remaining heap. | O(Log V) |

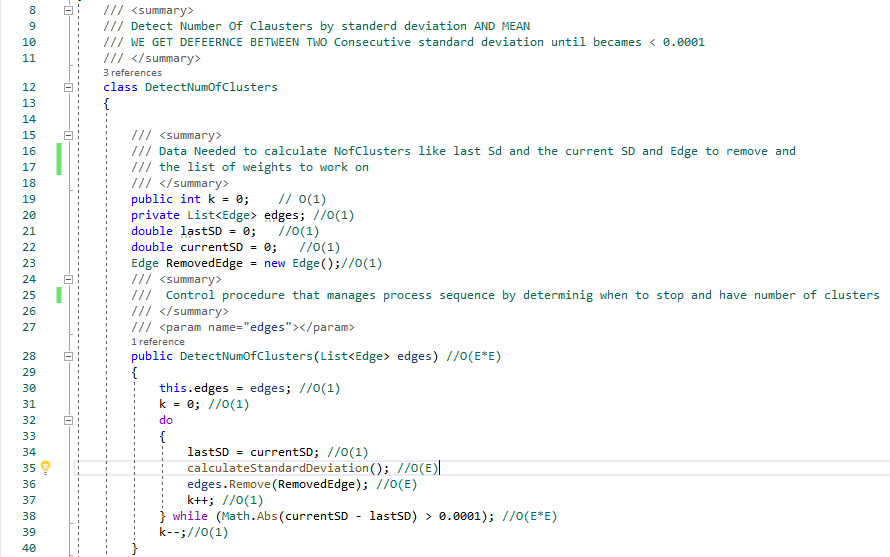
# 

# “HeapNode.cs”

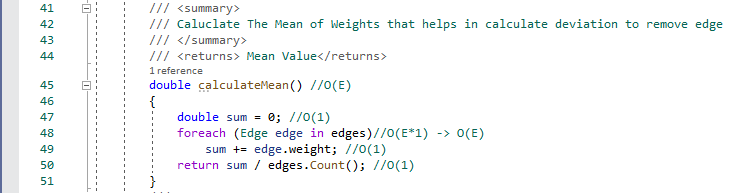
* HeapNode class.
* **Description** : class holds the data of each node.



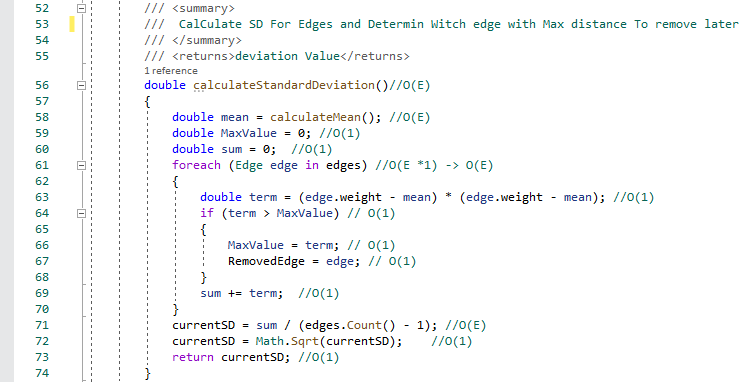
* **“Edeg.cs”**
* Edge class.
* **Description :** class holds the data of each edge(relation).
* Public Edge(int source, int destination, double weight ) : -
* **Description** : - it is just initialize function(constructor).
* **Time Complexity** :- **Θ(1)**

* **“DetectNumOfClusters.cs”**
* DetectNumOfClusters Class
* **Description** : Class that helps to get number of clusters to group them later built on that number( means number of grouping ).
*  public DetectNumOfClusters(List<Edge> edges)
* **Description :** .It depends on making disjoint trees by removing edge with max SD in each iteration. It responsible for extracting K → ( Number of clusters ) using Standard Deviation and Mean until two consecutive Standard Deviation became <0.001 .
* **Time Complexity : Θ(E\*E).**

* double calculateMean()

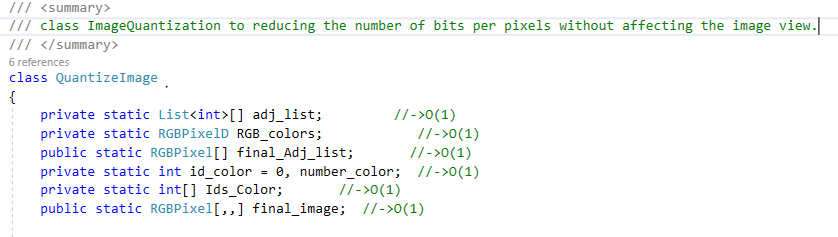
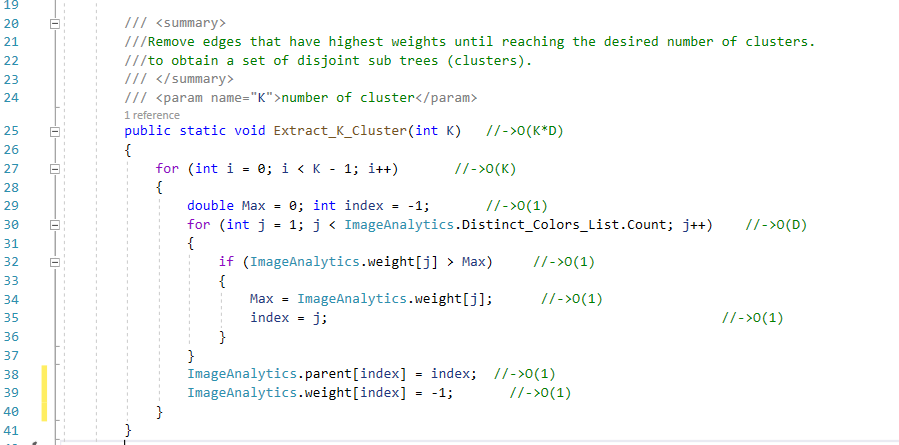
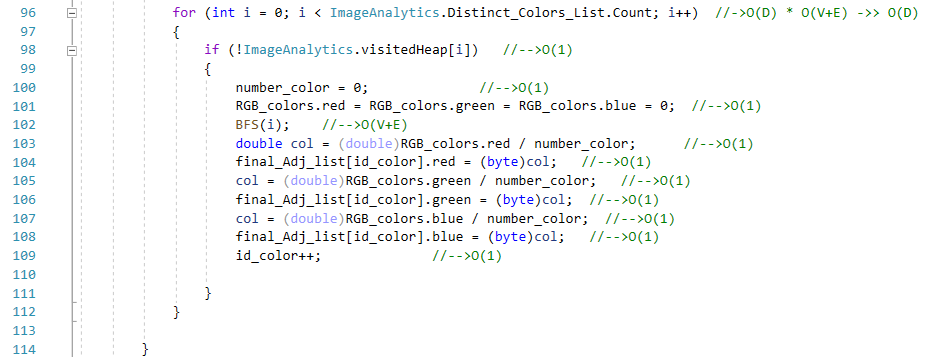
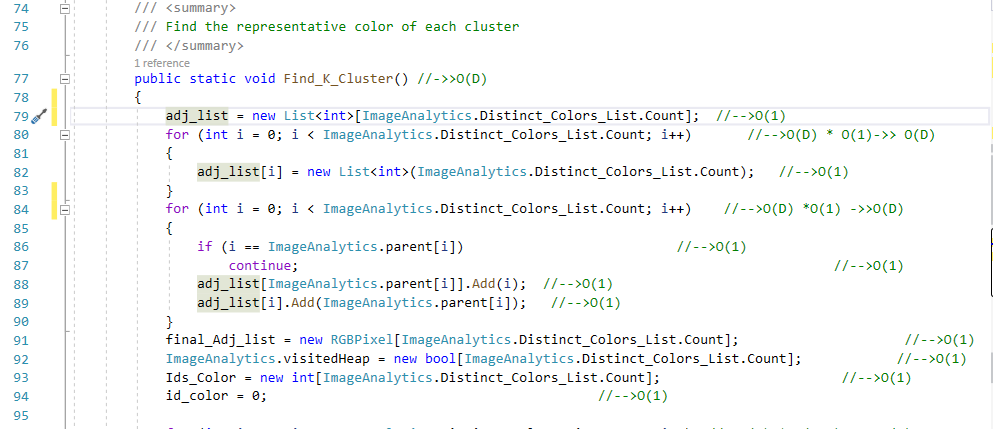


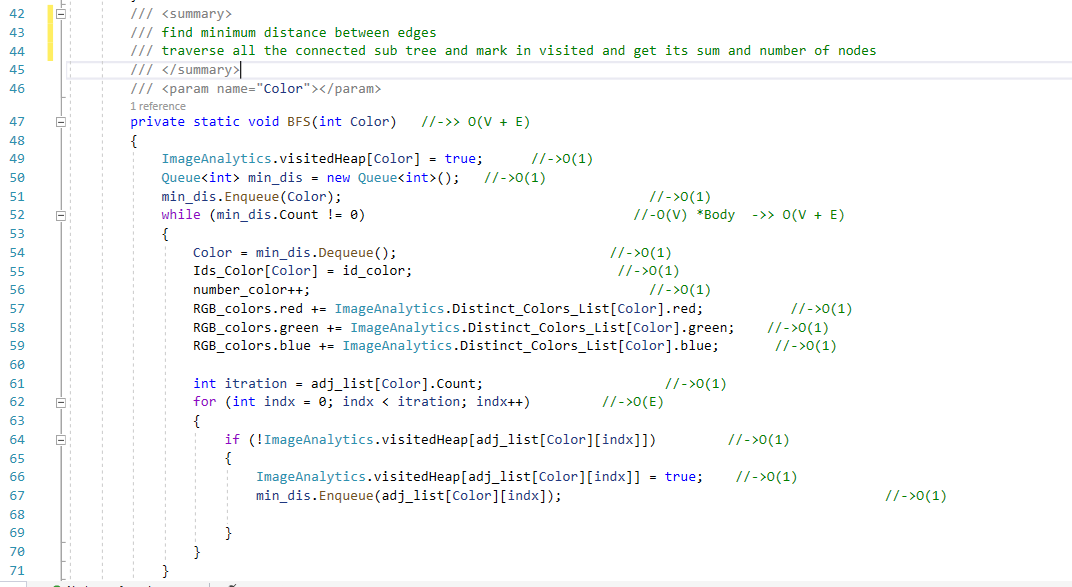
* **Description :** this gets the mean(Of list) by summing weights of edges on their count To use in standard Deviation..
* **Time Complexity : Θ(E).**
* double calculateStandardDeviation()

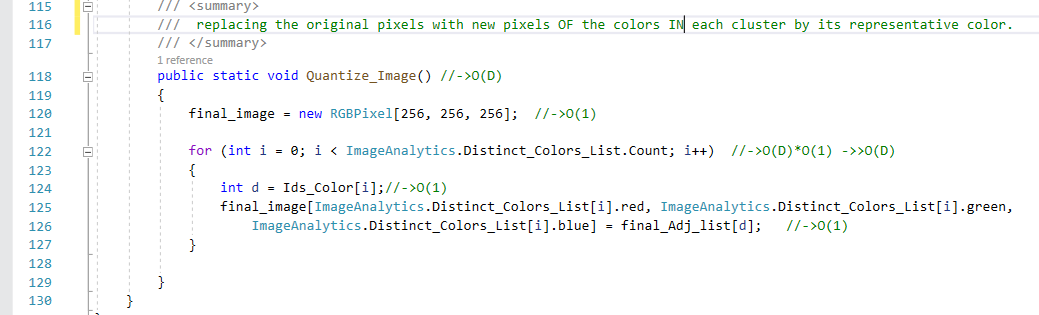


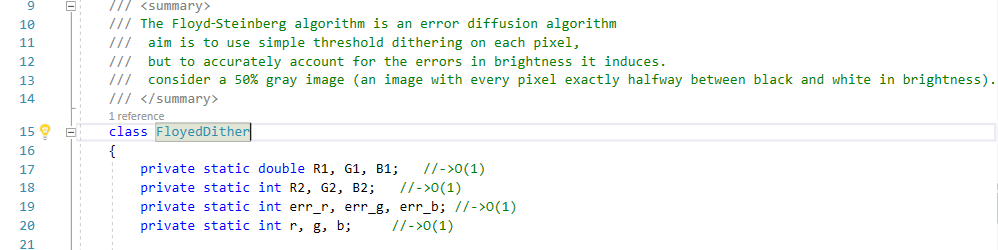
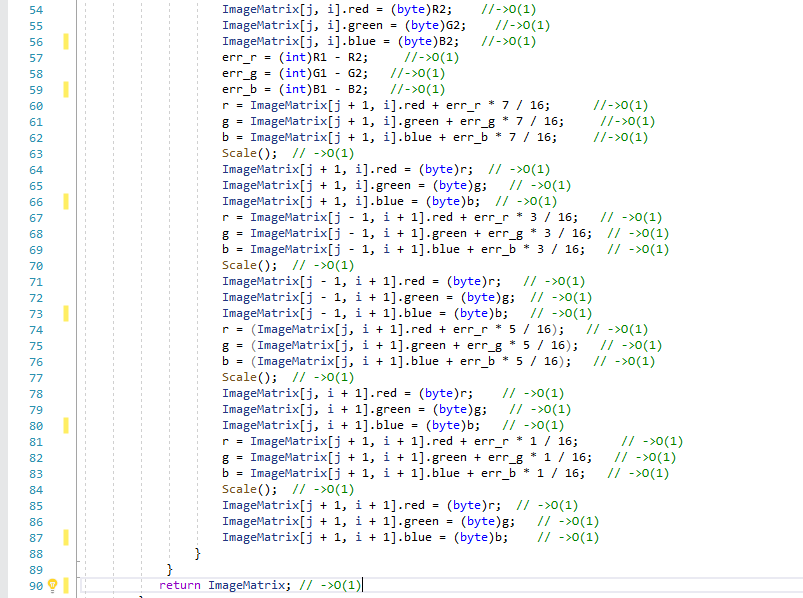
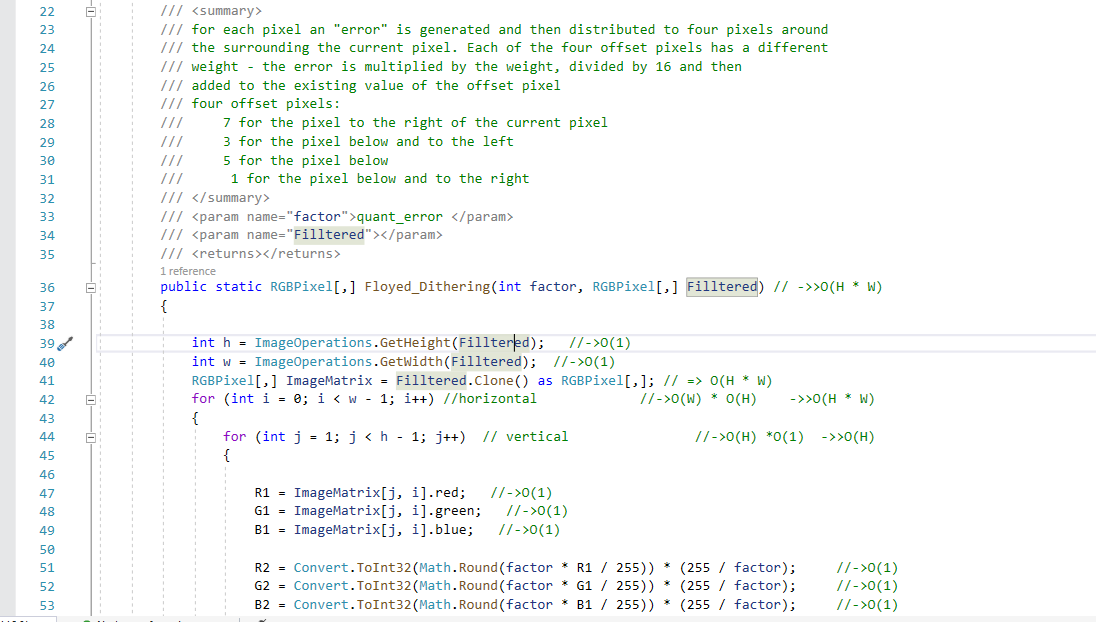
* **Description :** this gets the Standard Deviation of list using mean and determining which each with max SD to remove later.
* **Time Complexity : Θ(E).**

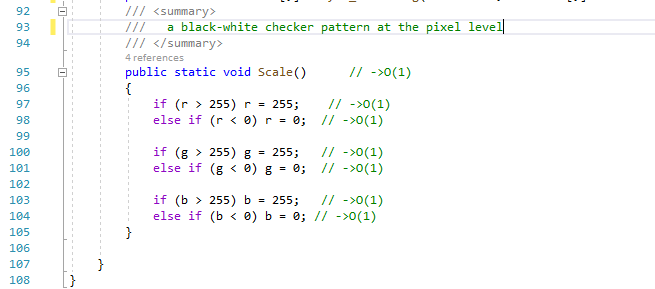
# “QuantizeImage.cs”

* QuantizeImage class.
*  **Description** : class used to reducing the number of bits per pixels without affecting the image view.
* public static void Extract\_K\_Cluster(int K)
* **Description :** Remove K-1 edges that have highest weights until reaching the desired number of clusters to obtain a set of disjoint sub trees (clusters).
* **Time Complexity : Θ(K\*D).**
* public static void Find\_K\_Cluster()
* **Description :** Find the representative color of each cluster.
* **Time Complexity : Θ(D).**

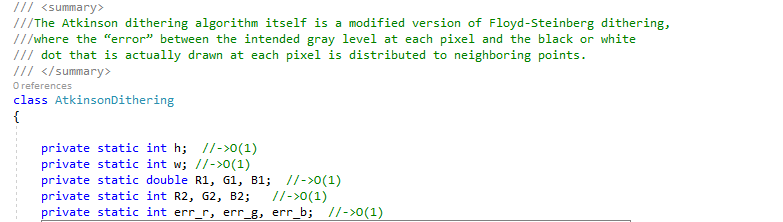
* private static void BFS(int Color)
* **Description :** traverse all the connected subtree and mark in visited and get its sum and number of nodes
* **Time Complexity : Θ(V+E).**
* public static void Quantize\_Image()

****

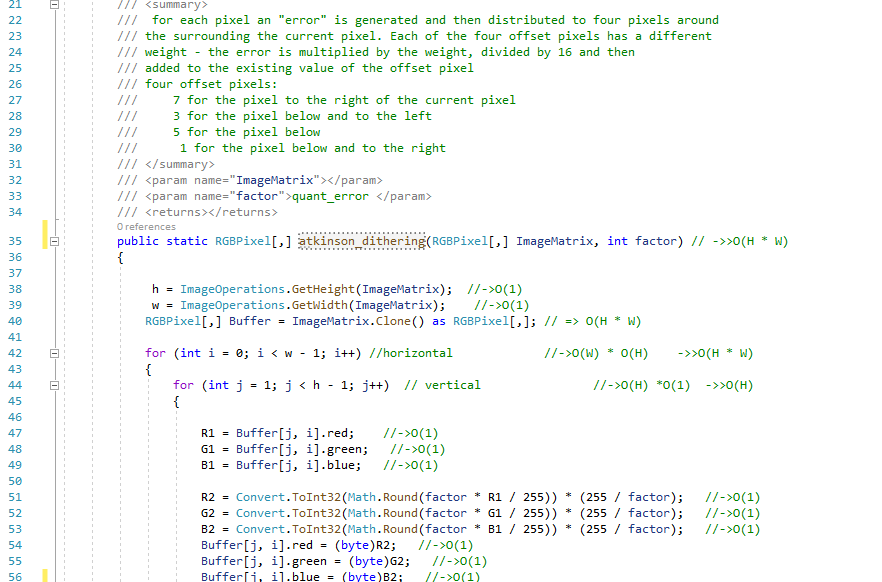
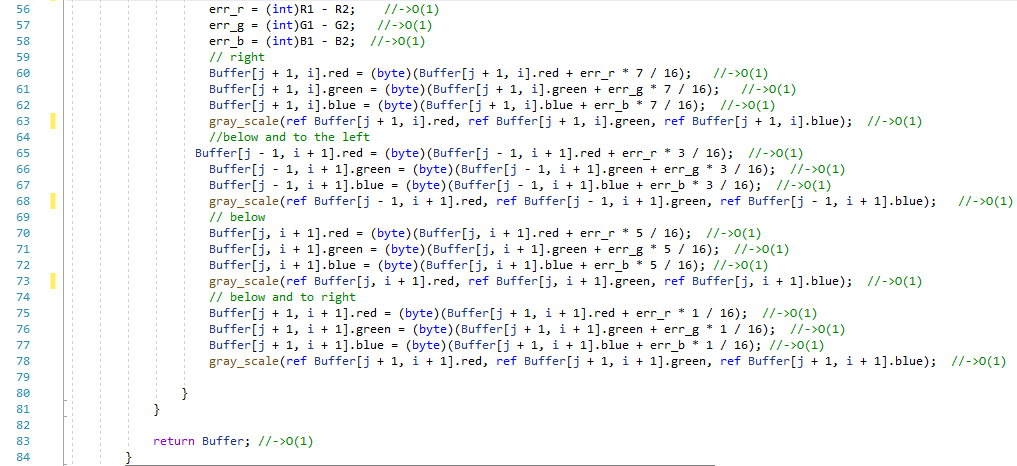
* **Description:** replacing the original pixels with new pixels based colors in clusters.
* **Time Complexity: O(D).**
* **Extra Work (Filters)**
* **Class FloyedDither:**
* **Description :** The Floyd‑Steinberg algorithm is an error diffusion algorithm aim is to use simple threshold dithering on each pixel.
* ******public static RGBPixel[,] Floyed\_Dithering(int factor, RGBPixel[,] Filltered)**
  + **Description:** for each pixel an "error" is generated and then distributed to four pixels around the surrounding the current pixel.Each of the four offset pixels has a different weight - the error is multiplied by the weight, divided by 16 and then added to the existing value of the offset pixel and check in pixel if in range 0-255, four offset pixels:
    - 7 for the pixel to the right of the current pixel
    - 3 for the pixel below and to the left
    - 5 for the pixel below
    - 1 for the pixel below and to the right
  + **Time Complexity: O(H \* W).**
* **public static void Scale()**

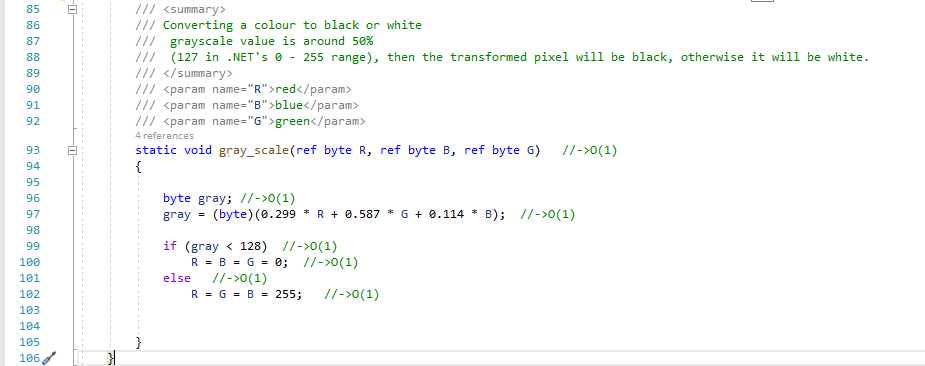
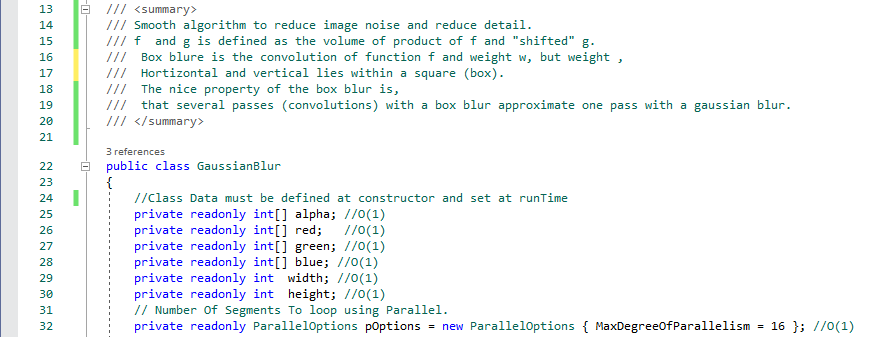
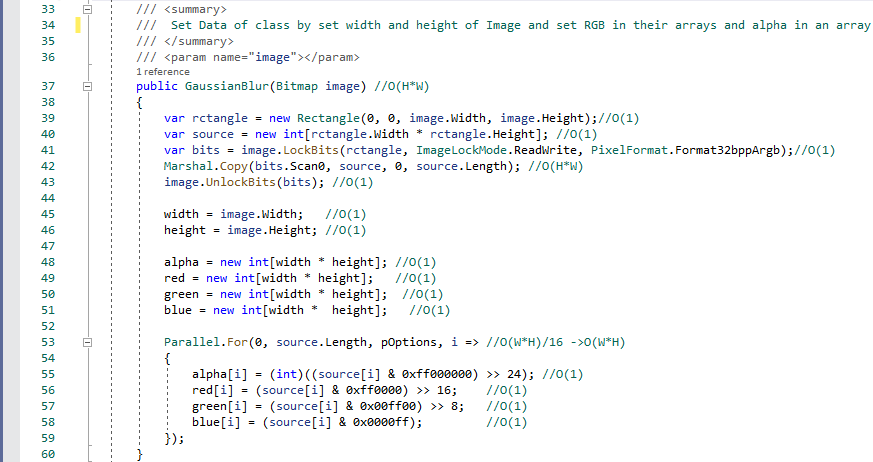
****

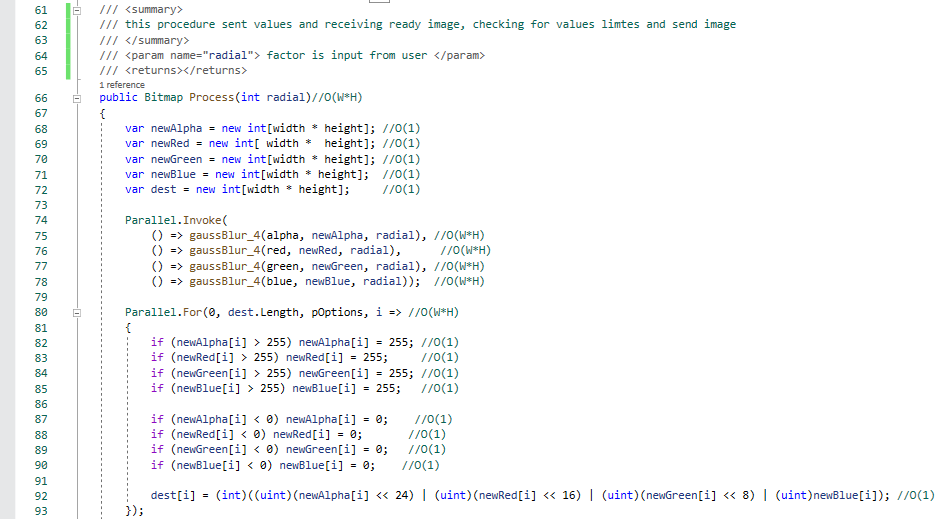
* + - **Description: black-white checker pattern at the pixel level**
    - **Time Complexity: O(1).**
* **Class Atkinson Dithering:**
* **Description : The** Atkinson dithering algorithm itself is a modified version of Floyd-Steinberg dithering, where the “error” between the intended gray level at each pixel and the black or white dot that is actually drawn at each pixel is distributed to neighboring points.

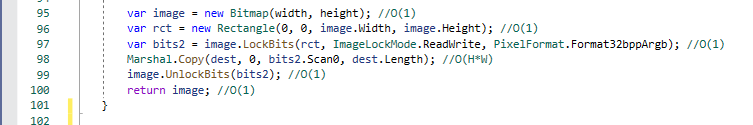
****

* **public static RGBPixel[,] atkinson\_dithering(RGBPixel[,] ImageMatrix, int factor)**

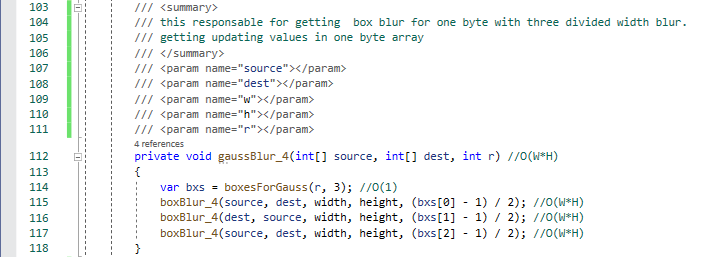
****

* + **Description:** for each pixel an "error" is generated and then distributed to four pixels around the surrounding the current pixel.Each of the four offset pixels has a different weight - the error is multiplied by the weight, divided by 16 and then added to the existing value of the offset pixel,convert each pixel to white or black four offset pixels:
    - 7 for the pixel to the right of the current pixel
    - 3 for the pixel below and to the left
    - 5 for the pixel below
    - 1 for the pixel below and to the right
  + **Time Complexity: O(H \* W).**
* **static void gray\_scale(ref byte R, ref byte B, ref byte G)** 
  + - **Description: convert each pixel to black or white.**
    - **Time Complexity: O(1).**
* **“GaussienFilter.cs”**
* GaussianBlur class
* **Description :**-Gaussian filter is a filter of blurring an image. Gaussian blur has the effect of reducing the image's high-frequency components; a Gaussian blur is thus a [low pass filter](https://en.wikipedia.org/wiki/Low_pass_filter).
* ****public GaussianBlur(Bitmap image)
* **Description :** This constructor take image and divide it into four arrays (Red, Green, Blue, Alpha).
* **Time Complexity: O(H\*W),** It takes **Linear Time** O(N)**.**Because H\*W is the size of array.
* **public Bitmap Process(int radial)**

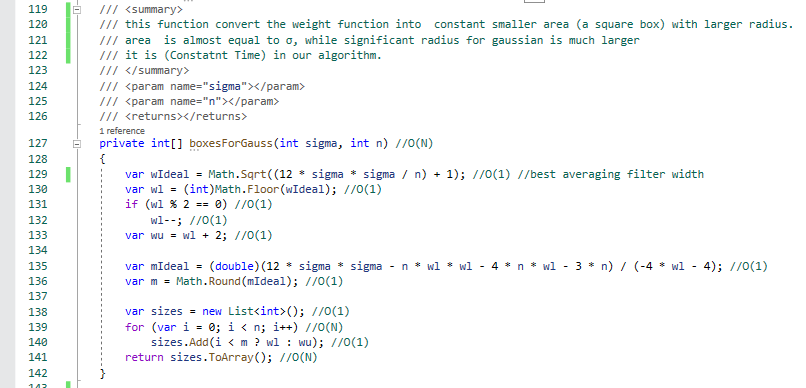
****

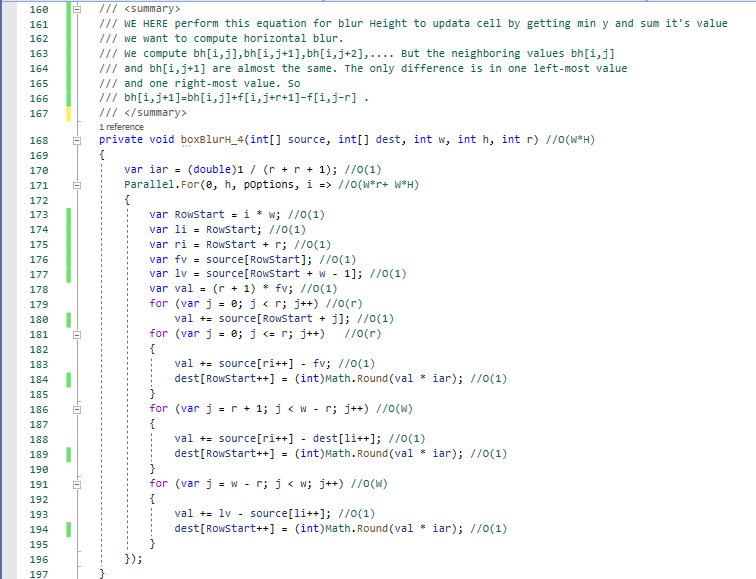
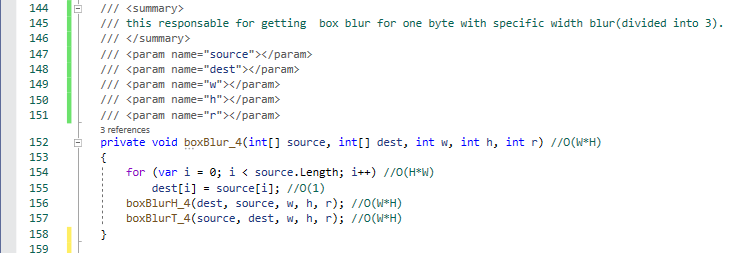
****

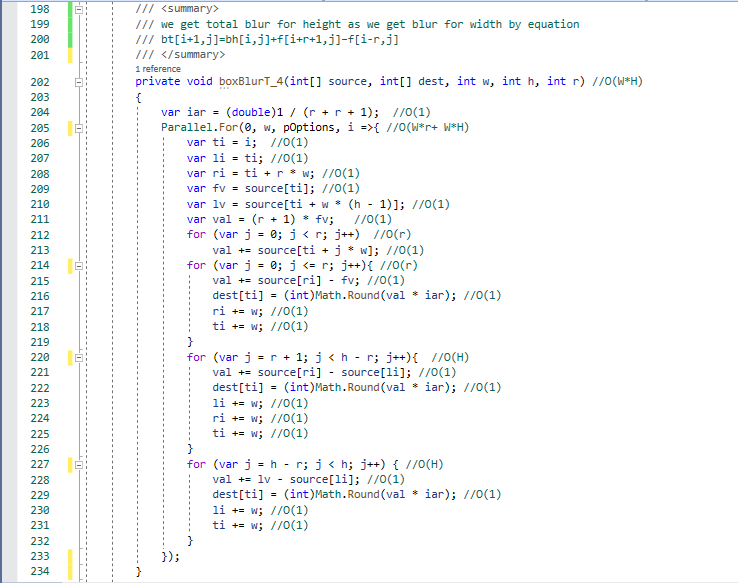
* **Description :** Process function responsible to return blurred image.process Function manage steps. It takes the factor to change image depends on and sent it to helper function that called (gaussBlur\_4) that update ( or blur) image. Process Function then check values limits and return image.
* **Time Complexity: O(H\*W),** It takes **Linear Time** O(N)**.**Because H\*W is the size of array.
* private void gaussBlur\_4(int[] source, int[] dest, int r)



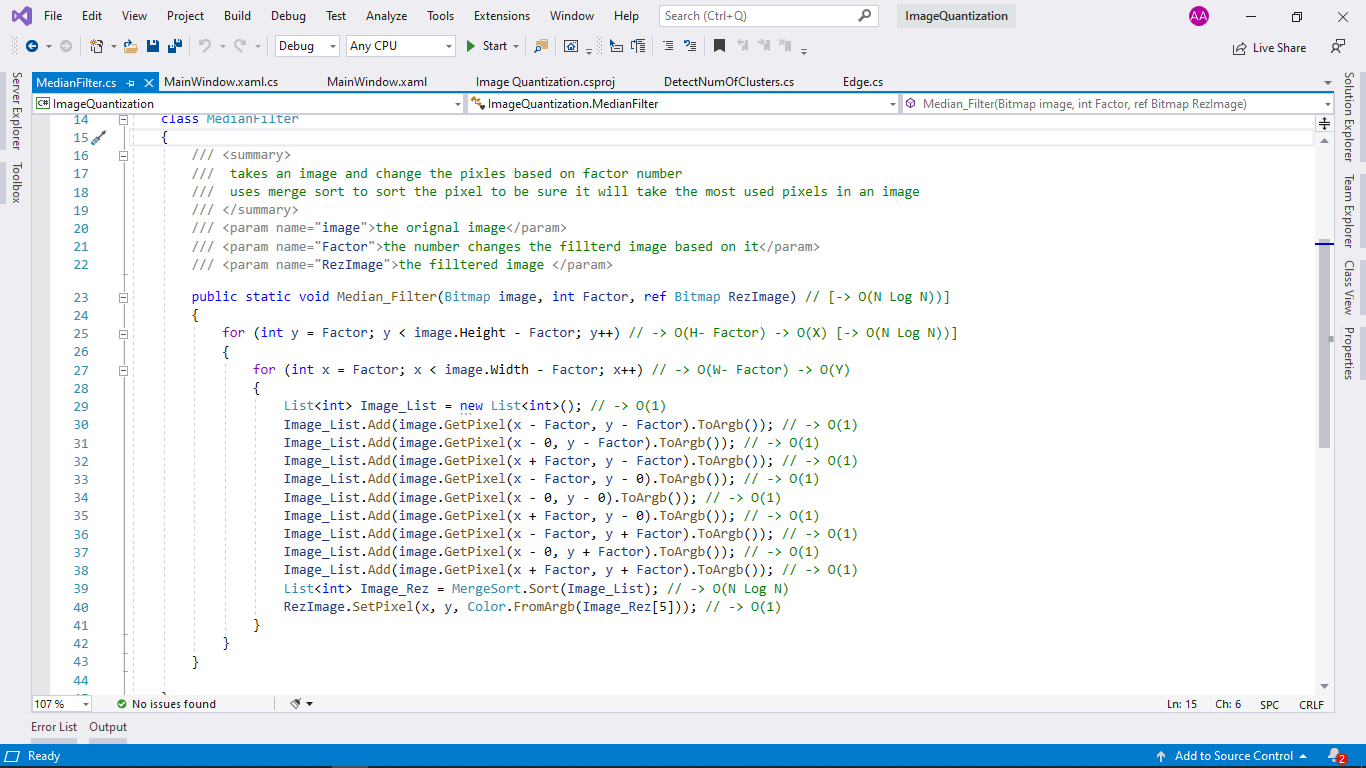
* **Description :** gaussBluer responsible to update ( blur ) in one array .gaussBluer uses helper function (boxesForGauss) that divide weight ( r) into 3 box blur and then send the array with every weight box to helper function(boxBlur\_4) to change in.
* **Time Complexity: O(H\*W),** It takes **Linear Time** O(N)**.**Because H\*W is the size of array.
* private int[] boxesForGauss(int sigma, int n)



* **Description :** This function converts weight into smaller area (A square area).
* **Time Complexity:**O(N) ,but in our code always N=3, so it’s constant -> **O(1)**.
* private void boxBlur\_4(int[] source, int[] dest, int w, int h, int r)
* **Description :** This function uses helper function (boxBlurH\_4) to update (or Blur) in an array with one weight part horizontally. And then use function (boxBlurT\_4) to update also vertically.
* **Time Complexity :O(H\*W),** It takes **Linear Time** O(N)**.**Because H\*W is the size of array.
* private void boxBlurH\_4(int[] source, int[] dest, int w, int h, int r)
* **Description :** this function plays in array Horizontally to blur with specific weight using this equation :-
* bh[i,j+1]=bh[i,j]+f[i,j+r+1]−f[i,j−r].
* **Time Complexity :O(H\*W),** It takes **Linear Time** O(N)**.**Because H\*W is the size of Array.
* private void boxBlurT\_4(int[] source, int[] dest, int w, int h, int r)



* **Description :** this function plays in array Vertically to blur with specific weight using this equation :-
* b[ i+1,j ] = bt[ i , j ]+ f[ i+r+1 ,j ] − f [ i-r, j ].
* **Time Complexity :O(H\*W),** It takes **Linear Time** O(N)**.**Because H\*W is the size of array.
* **“MedianFilter.cs”**
* **Description :** Medianfilter is another very common filter, which reduces noise and preserves edges. It can fix pixels or even small areas with damaged or missing color.
* public static void Median\_Filter(Bitmap image, int Factor, ref Bitmap RezImage)



* **Description :** takes an image and change the pixles based on factor number uses merge sort to sort the pixel to be sure it will take the most used pixels in an image:-
* **Time Complexity :**  **O(N Log N))**