Analysis and Design of Algorithms Project

Team members:

- Mohammed Mohammed Alaa
- Ahmed Helmy
- Raphael Serwanis
- Marian Ayman
- Abdallah Mahmoud
- Marc Gouda

Complexity Analysis of Alpha trim code:

```
void kth(int[] arr)
        {
            int small = 255;                                    1
            int big = 0;                                        1
            int index = 0;                                      1

            for (int i = 0; i < arr.Length; i++)                N
            {
                if (arr[i] == 0)                                1
                    continue;

                else if (small > arr[i])                        1
                {
                    small = arr[i];                             1
                    index = i;                                  1
                }
            }
            arr[index] = 0; // smallest                         1

            index = 0;                                          1

            for (int i = 0; i < arr.Length; i++)                N
            {
                if (big < arr[i])                               1
                {
                    big = arr[i];                               1
                    index = i;                                  1
                }
            }
            arr[index] = 0; // biggest                          1
        }
```

By summation → O(n)

```
void creatingWindow(int[] window, byte[,] imageMatrix, int t, ref int rowFlag, ref
int colFlag, ref int shifter)
        {
            int counter = 0;                                                    1
            for (int i = rowFlag; i < t + rowFlag; i++)                         N
            {
                for (int j = 0; j < t; j++)                                     N
                {                                                               1
                    window[counter] = imageMatrix[i, j + shifter];              1
                    counter++;                                                  1
                    colFlag = j + shifter;
                }

            }
            shifter++;                                                          1
        }
```

By summation → O(n^2)

```
        public byte[,] AlphaFilter(int t, int Size, byte[,] imageMatrix, int
method)
        {
            int rowflag = 0;                                                    1
            int colflag = 0;                                                    1
            int shifter = 0;                                                    1
            int[] window = new int[Size];                                       1
            byte [,] newImage = new byte[imageMatrix.GetLength(0),
imageMatrix.GetLength(1)];

            while (true)                                                        n^2
            {
                int sum = 0;                                                    1

                // o(m^2) where m < n
                creatingWindow(window, imageMatrix, t, ref rowflag, ref colflag,    m^2
ref shifter);

                // n                                                            m
                if (method == 1)
                {
                    // order of m+k where k = 255 so, order of m
                    countingSort(window);                                       m

                    // order of m                                               1
                    for (int i = 0; i < window.Length - 2 * t; i++)
                    {
                        sum += window[i + t];
                    }                                                           1
                }                                                               m^2
                else if (method == 2)                                           m
                {                                                               m
                    // m square
                    for (int i = 0; i < t; i++)
                        kth(window);
                    for (int i = 0; i < window.Length; i++)                     1
                        sum += window[i];                                       1
                }
                int newPixel = sum / window.Length - 2 * t;
                newImage[rowflag + t / 2, shifter] = (byte)newPixel;

                if (colflag == (imageMatrix.GetLength(1) - 1) && rowflag ==
imageMatrix.GetLength(0) - t)
                {
                    return newImage;

                }
                else if (colflag == (imageMatrix.GetLength(1) - 1))
                {
                    colflag = 0;
                    shifter = 0;
                    rowflag++;
                }

            }


        }
```

```
1
1
1

1

1
1
1
```

By summation → O(n^2 * m^2)

Objective:

- Order of Alpha trim function is n^2 * m^2, where n is the size of the image matrix and m is the trim value.
- Creating window function order is dominating whether the filter works with kth or counting sort so to compare between them fairly, it will be not considered.
- Working with Kth algorithm gives order of m^2, however working with counting sort gives order of m+k, where k is a constant equal 255, so its order is m. Overall, it is obvious that counting sort works faster than kth as seen in figure 3. For small Ws kth seems to be faster but when Ws becomes bigger than 3, counting sort shows domination with nearly steady time less than 300, unlike kth algorithm which has a linear function.

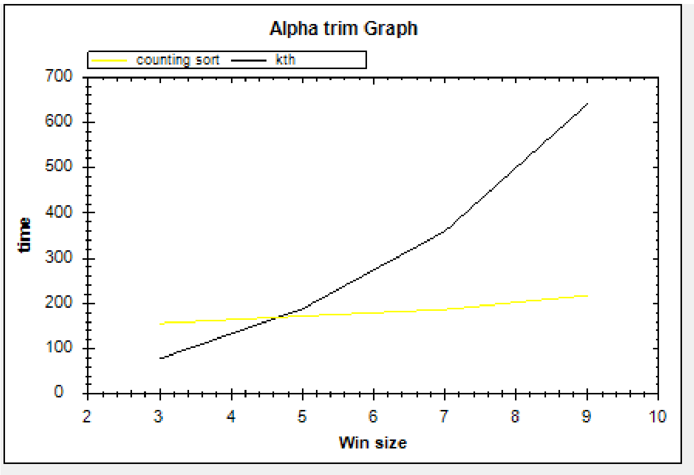

Figure 1 ● counting sort ● kth

Complexity Analysis of Adaptive median code:

```
        public void Adeptive_median(int Ws, int Size, byte[,] imageMatrix, int
t, int method)
        {

            int rowflag = 0;                                                        1
            int colflag = 0;                                                        1
            int shifter = 0;                                                        1
            int Zmin, Zmax, Zmed, Zxy;                                              1
            int[] window = new int[Size];                                           1
            //byte[,] newImage = new byte[imageMatrix.GetLength(0),                 1
imageMatrix.GetLength(1)];

            while (true)
            {
                // o(m^2) where m < n
                creatingWindow(window, imageMatrix, t, ref rowflag, ref colflag,    m^2
ref shifter);


                if (method == 0)                                                    1

                    // order of m+k where k = 255 so, order of m                    m
                    countingSort(window);
                //nlogon                                                            1
                else if (method == 1)
                    quickSort(window, 0, window.Length - 1);                        mlogm

                Zmin = window[0];                                                   1
                Zmax = window[window.Length - 1];                                   1
                Zmed = window[(window.Length + 1) / 2];                             1
                Zxy = imageMatrix[rowflag + t / 2, shifter];                        1

                int A1 = Zmed - Zmin;                                               1
                int A2 = Zmax - Zmed;                                               1
                int newpixelVal = 0;                                                1
                if (A1 > 0 && A2 > 0)                                               1
                {
                    int B1 = Zxy - Zmin;                                            1
                    int B2 = Zmax - Zxy;                                            1
                    if (B1 > 0 && B2 > 0)                                           1
                    {
                        newpixelVal = Zxy;                                          1
                    }
                    else                                                            1
                    {
                        newpixelVal = Zmed;
                    }
                }
                else
                {
                    if (t + 2 <= Ws)                                                1
                    {
                        if (colflag == (imageMatrix.GetLength(1) - 2))              1
                        {
                            shifter--;                                                      1
                        }
                        else if (colflag == (imageMatrix.GetLength(1) - 1))                 1
                        {
                            shifter -= 2;                                                   1
                        }

                        if (rowflag == imageMatrix.GetLength(0) - t)                        1
                        {
                            rowflag -= 2;                                                   1
                        }
                        else if (rowflag == imageMatrix.GetLength(0) - t - 1)               1
                        {
                            rowflag -= 1;                                                   1
                        }
                        t = t + 2;                                                          1
                        window = new int[t * t];                                            1
                        continue;                                                           1
                    }
                    else
                    {                                                                       1
                        newpixelVal = Zmed;
                    }
                }

                //newImage[rowflag + t / 2, shifter] = (byte)newpixelVal;
                imageMatrix[rowflag + t / 2, shifter] = (byte)newpixelVal;                  1

                if (colflag == (imageMatrix.GetLength(1) - 1) && rowflag ==
imageMatrix.GetLength(0) - t)                                                               1
                {
                    break;                                                                  1
                }
                else if (colflag == (imageMatrix.GetLength(1) - 1))                         1
                {
                    colflag = 0;                                                            1
                    shifter = 0;                                                            1
                    rowflag++;                                                              1
                }
            }
        }
```

By summation → O(n^2 * m^2)

Objective:

- Order of Adaptive median function is n^2 * m^2, where n is the size of the image matrix and m is the trim value.
- Same as alpha trim function, creating window function will be not considered.
- Order of counting sort is m as mentioned before, while order of quick sort is m log m. Overall, working with counting sort is faster and better than quick sort. From figure 4, quick sort has a linear relation with Ws. Although, quick sort takes less time than counting sort with Ws like 3 and 5, counting sort takes much less time than QS with bigger Ws.

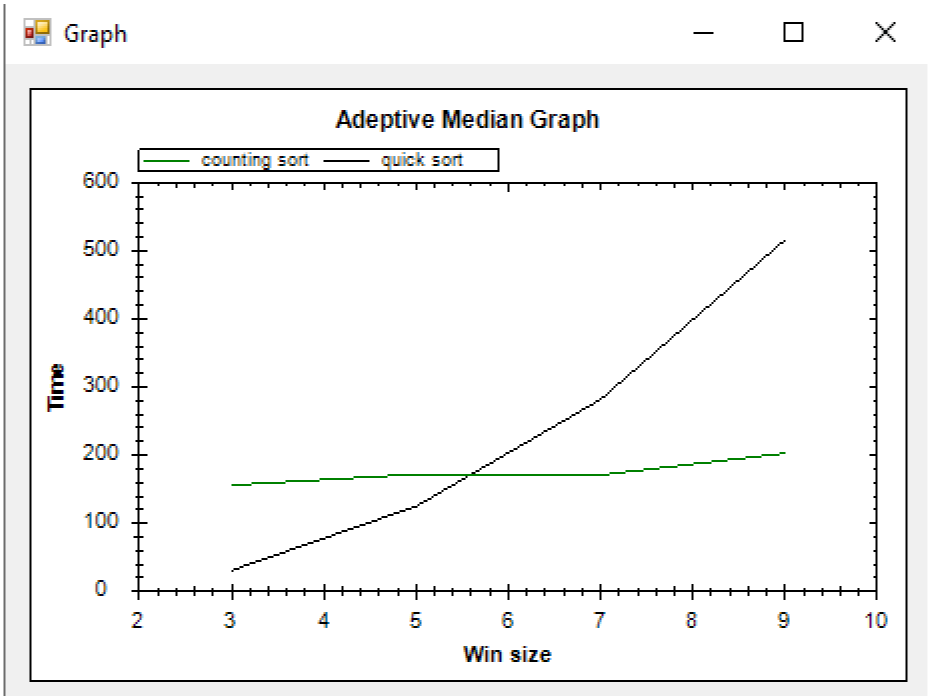

Figure 2, ● counting sort ● quick sort