RUHUNA JOURNAL OF SCIENCE
Vol 8: 67- 75, June 2017
eISSN: 2536-8400
DOI: http://doi.org/10.4038/rjs.v8i1.27

Appendix

The implementation of the proposed algorithm in C programming language is given below.

```
/* Inclusion of header files. */
#include<stdio.h>
#include<conio.h>
#include<math.h>

/*
For an example, let there are 17 vertices in the polygon. And coordinates of the vertices are;
(40, 10), (50, 40), (80, 10), (130, 20), (140, 70), (150, 20), (250, 90), (190, 130), (140, 80), (90,
70), (80, 130), (30, 180), (20, 140), (50, 120), (20, 100), (40, 60), (10, 50).
Note that (x[i], y[i]) stores the coordinates of the vertex with index i.
*/
double x[]={40,50,80,130,140,150,250,190,140,90,80,30,20,50,20,40,10};
double y[]={10,40,10,20,70,20,90,130,80,70,130,180,140,120,100,60,50};
int points=17;

double yg[50];
int ygn=0;

double yh[50];

double sum1[50];
double sum2[50];

double gap1[50];
double gap2[50];
int n1,n2;

/*
This function sorts the y-coordinates of the vertices in ascending order.
*/
void sortArray(double p[],int n)
{
int i,j;
int min;
double temp;
        for(i=0;i<n-1;i++)
        {
                min=i;
                for(j=i+1;j<n;j++)
                {
                        if(p[min]>p[j])
                        {
                                min=j;
                        }
                }
                if(i!=min)
                {
                        temp=p[i];
                        p[i]=p[min];
                        p[min]=temp;
                }
        }
}

/*
This function selects y-coordinates of the vertices and passes them to function sortArray.
*/
void findYg()
{
int i;
        for(i=0;i<points;i++)
        {
                yg[i]=y[i];
        }
        sortArray(yg,points);
}

/*
This function is used to decide the position of a given vertex (x[v], y[v]) with respect to a
horizontal line y = yg[u].
*/
int pos(int u,int v)
{
        if(y[v]>yg[u])
        {
                return 1;
        }
        else if(y[v]<yg[u])
        {
                return -1;
        }
        else
        {
                return 0;
        }
}

/*
This function computes the area bounded by two horizontal lines. In other words, the sum of
areas of each trapezoid between the horizontal lines is computed.
*/
double getAreaS(int i)
{
double area;

        area=0.5*(yg[i+1]-yg[i])*(sum1[i]+sum2[i+1]);

return area;
}

/*
```

```
This function adds each area bounded by pair of horizontal lines and computes the total area of
the polygon.
*/
double getArea()
{
int i;
double area=0;

        for(i=0;i<ygn-1;i++)
        {
                area+=getAreaS(i);
        }
return area;
}

/*
This function computes elements of arrays gap1 and gap2.
*/
void findGaps(int u)
{
int i,j;
double xc;
int pi,pj,pa,pb,a,b;

        for(i=0;i<points;i++)
        {
                j=(i+1)%points;
                pi=pos(u,i);
                pj=pos(u,j);

                if(pi*pj<0)
                {
                        xc=x[i]+(x[j]-x[i])*(yg[u]-y[i])/(y[j]-y[i]);
                        gap1[n1]=xc;
                        n1++;
                        gap2[n2]=xc;
                        n2++;
                }
                else if((pi==0) && (pj==0))
                {
                        a=i-1;
                        b=j+1;

                        if(a==-1)
                        {
                                a=points-1;
                        }
                        if(b==points)
                        {
                                b=0;
                        }

                        pa=pos(u,a);
                        pb=pos(u,b);
                        if(pa*pb>0)
                        {
                                if(pa>0)
                                {
                                    gap1[n1]=x[i];
                                    n1++;
                                    gap1[n1]=x[j];
                                    n1++;
                                }
                                else
                                {
                                    gap2[n2]=x[i];
                                    n2++;
                                    gap2[n2]=x[j];
                                    n2++;
                                }
                        }
                        else
                        {
                                if(pa>0)
                                {
                                    gap1[n1]=x[i];
                                    n1++;
                                    gap2[n2]=x[j];
                                    n2++;
                                }
                                else
                                {
                                    gap1[n1]=x[j];
                                    n1++;
                                    gap2[n2]=x[i];
                                    n2++;
                                }
                        }
                }
        }

        for(i=0;i<points;i++)
        {
                if(y[i]==yg[u])
                {
                        a=i-1;
                        b=i+1;

                        if(a==-1)
                        {
                                a=points-1;
                        }
                        if(b==points)
                        {
                                b=0;
                        }
```

```
            pa=pos(u,a);
            pb=pos(u,b);

            if(pa*pb<0)
            {
                gap1[n1]=x[i];
                n1++;
                gap2[n2]=x[i];
                n2++;
            }
        }
    }
}

/*
This function computes the total length of horizontal segments represented in array gap.
*/
void computeSum(double sum[],int index,double gap[],int n)
{
int i;
    sum[index]=0;

    for(i=0;i<n;i++)
    {
        if(i%2)
        {
            sum[index]+=gap[i];
        }
        else
        {
            sum[index]-=gap[i];
        }
    }
}

/*
This function first sorts the arrays gap1 and gap2 by invoking function sortArray.
Then this function computes elements of arrays sum1 and sum2 by invoking function
computeSum.
Note that sum1 and sum2 represents the total lengths of horizontal segments represented by
arrays gap1 and gap2 respectively.
*/
void findSums()
{
int i;
    for(i=0;i<ygn;i++)
    {
        n1=0;
        n2=0;

        findGaps(i);
        sortArray(gap1,n1);
        sortArray(gap2,n2);
```

```
                    computeSum(sum1,i,gap1,n1);
                    computeSum(sum2,i,gap2,n2);
          }
}

/*
This function removes the duplicate points in the sorted array Yg.
*/
void refineYg()
{
int i;

          for(i=0;i<points;i++)
          {
                    yh[i]=yg[i];
          }

          yg[0]=yh[0];
          ygn++;

          for(i=1;i<points;i++)
          {
                    if(yh[i-1]!=yh[i])
                    {
                              yg[ygn]=yh[i];
                              ygn++;
                    }
          }
}

/*
This is the main method of the program.
*/
void main()
{
findYg();
refineYg();
findSums();

printf("%f",getArea());

getch();
}
```