ANALYSIS REPORT

PSEUDOCODE

Functions Used:

1. readwritefile()
2. greedy_SeparatingPoints()
3. checksplitH()
4. checksplitV()

Create:

1. List vert
2. List horz
3. List solution
4. int XC[], YC[]

Function readwritefile()

```
1.  File folder=new File("input")
2.  File[] listofFiles= folder.listFiles()
3.  for(i=0 to listofFiles.length)
4.          if(file.isFile())
5.                  String filename= file.getName()
            numlines=0
6.                  br= new BufferedReader(new
                    FileReader("input/"+filename);
7.                  line= br.readLine();
8.                  numlines: get number of lines from first line of file
9.                  Create array XC[], YC[] of length numlines
10.                 Read X and Y coordinates line by line into XC, YC
11.                 br.close
            end if

    //output file creation
12. outfilename= "greedysolution_"
13. if i<10
14.         outfilename= outfilename + "0" + (i+1)
    else
15.         outfilename= outfilename + (i+1)


    //create file in output_greedy folder
16. File outfile= new File("output_greedy", outfilename)
17. outfile.createNewFile();
18. bw: BufferedWriter object to write into files


    //call algorithm to select lines
19. greedy_SeparatingPoints(XC, YC, 0, XC.length-1)
```

```
    //write the number of lines and solution to //file
20. bw.write(solution.size())
21. for z=0 to solution.size()
            bw.write(solution.get(z)+”\n”);
    end for
22. bw.close
    end for
    end method
```

Analysis:

1. Line 1 and 2 are initialization and takes O(1) time.
2. Line 3 has 1 initialization, comparisons up to number of files in folder, say “f”.
3. Line 4 to 18 will run for “f” times.
4. Line 19 is the call to greedy_SeparatingPoints() which will itself take time. The call to this method will be for the number of files in the folder.
5. Line 20 to 22 will be executed “f” times.

Method 2

```
    greedy_SeparatingPoints(int X[], int Y[], int start,  int end)
1.  length= X.length
2.  boolean splitH= false, splitV= false
3.  diff= end- start
4.  if diff>0
5.          vertical=0, horizontal=0
6.          firstvalue=X[start]
7.          lastvalue=X[end]
8.          splitmid= firstvalue+ lastvalue
9.          if (splitmid%2 !=0)
10.                 vertical= splitmid/2
11.                 horizontal= splitmid/2
            else
12.                 vertical= splitmid- 1/2
13.                 horizontal= splitmid- 1/2
            end if
14.         if (vert.isEmpty())
15.                 vert.add= (vertical)
16.                 solution.add(“v” + vertical)
            else
17.                 splitV= checksplitV(X,Y,vertical)
18.                 if (splitV== true)
19.                         vert.add= (vertical)
20.                         solution.add(“v” + vertical)
                    end if
            end if
21.         splitH= checksplitH(X,Y, horizontal)
22.         if (splitH== true)
23.                 horz.add= (horizontal)
24.                 solution.add(“h” + horizontal)
```

```
    end if
    upstart= vertical- 0.5
    downend= vertical- 1.5
    greedy_SeparatinfPoints(X, Y, start, downend)
    greedy_SeparatinfPoints(X, Y, upstart, end)
end if
end
```

Method 3:

```
boolean checksplitH(X1,Y1,splitpoint)
flagsplit=0
for i=0 to Y1.length
    for j=i+1 to Y1.length
        if flagsplit==0
            if ((Y1[i]>splitpoint && Y1[j]<splitpoint) ||
        (Y1[i]<splitpoint && Y1[j]>splitpoint))
                x1=X1[i]
                x2=X1[j]
                y1=Y1[i]
                y2=Y1[j]
                splitvertfound=0
                for k=0 to vert.size()
                    if splitvertfound=0
                        vertical= vert.get(k)
                        if (x1<vertical && x2>vertical)
                            splitvertfound=1
                        end if
                    end if
                end for
                if splitvertfound=0
                    if horz.isEmpty()
                        flagsplit=1
                    end if
                flagcount=0
                for l=0 to horz.size()
```

```
                    if flagcount=0
                        horzval=horz.get(l)
                        if ((y1>horzval && y2<horzval)||
                    (y1<horzval && y2>horzval))
                            flagcount=1
                        end if
                    end if
                end for
                if flagcount=1
                    flagsplit=0
                else
                    flagsplit=1
                end if
                end if
            end if
        end if
    end for
end  for
if flagsplit=0
    return false
else
    return true
end method
```

Method 4:

```
boolean checksplitV(X1[], Y1[], splitpoint)
flagsplit=0
x1=splitpoint-1.5
x2=splitpoint-0.5
for k=0 to horz.size()
    if flagspit=0
        betweenline= horz.get(k)
        if((Y1[x1]>betweenline && Y1[x2]<betweenline) ||
    (Y1[x1]<betweenline && Y1[x2]>betweenline))
            flagsplit=1
        else
            flagsplit=0
        end if
    end if
end for
if flagsplit=1
    return false
else
    return true
end method
```

Analysis of the procedures:
greedy_SeparationPoints()

1. The greedy_SeparationPoints() executes for each file in the folder (f times).
2. It also executes recursively for every split.
3. Since the split is done from 0 to 9 it is called n times, i.e. number of coordinate points.
4. The function List.add() has time complexity O(1).
5. Also isEmpty() has constant time.

checksplitH()

1. The checksplitH() function is called for every horizontal splitpoint to check whether it should be added to solution.
2. Line 1 takes O(1).
3. The two for loops are nested hence every line in this loop takes n^2.
4. Line 11 is a for loop, will execute till size of the vertical list. Hence say if size is n. Every statement in this loop will execute for n^3 times.
5. Line 16to 20 runs n^2 times.
6. Line 20 is for loop, hence statement 21 to 24 runs n^3 times.
7. Line 25 to 27 runs n^2 times.
8. Line 28 to 30 takes O(1) time.
9. The list's get()takes O(1) time. Hence complexity of this function is O(n^3).

checksplitV()

This function checks whether vertical split line is to be added in the solution.

1. Line 1 to 3 takes O(1).
2. The for loop in line 4 has 1 initialization comparisons equal to number of points in horz solution say n.
3. Line inside for runs O(n) times (Line 5 to 10).
4. Line 10 to 12 runs O(1) times.

This function takes O(n) times.

The overall complexity of the algorithm goes to n^4, as separating points function i.e. greedy_SeparatingPoints() takes O(n) and checksplitH() takes O(n^3). Therefore total time is O(n^4).

If we also consider number of files say n, then complexity rises to O(n^5).

- Algorithm analysis using instance
  Points 5

Input: 5

1	1
2	5
3	3
4	2
5	4

The algorithm gives optimum solution.
Output: 3
V 2.5
h 2.5
h 3.5

- Algorithm fails for
  The algorithm does not give optimum solution for 10 points.
  It gives one split point extra.

Input: 10

1	10
2	6
3	8
4	1
5	3
6	7
7	2
8	9
9	5
10	4

The algorithm gives following output:
7
v 5.5
h 5.5
v 2.5
h 2.5
v 1.5
h 4.5
h 7.5

The above solution is not optimal. The number of lines that can be used to split the above 10 points is

6

v	5.5
h	4.5
h	6.5

v	7.5
v	1.5
v	4.5

This above solution is better. This happens because my algorithm first visited lower half, i.e. points on X coordinate 0 to 5. Hence the horizontal line 2.5and 4.5 is drawn first.

When 6-10 points are visited and when v 7.5 is selected, we find that the points are already splitted and line is not drawn or selected in solution.