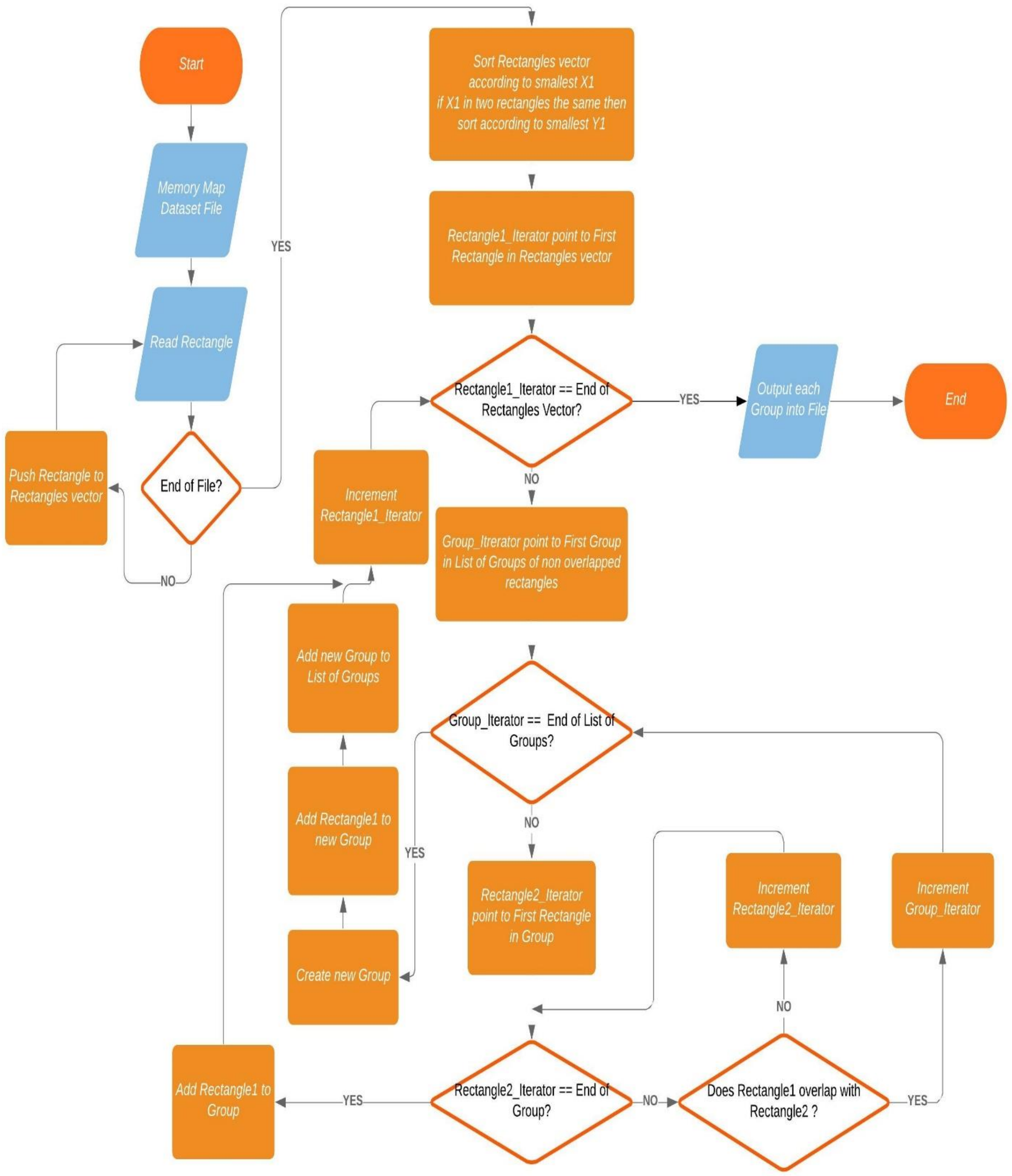
Report

1. Platform Used:

- Linux (Ubuntu 18.04)

2. Flow Chart:


Start
Memory Map Dataset File
Read Rectangle
End of File?
YES
NO
Push Rectangle to Rectangles vector
Sort Rectangles vector according to smallest X1 if X1 in two rectangles the same then sort according to smallest Y1
Rectangle1_Iterator point to First Rectangle in Rectangles vector
Rectangle1_Iterator == End of Rectangles Vector?
YES
Output each Group into File
End
NO
Increment Rectangle1_Iterator
Group_Itrerator point to First Group in List of Groups of non overlapped rectangles
Group_Iterator == End of List of Groups?
YES
NO
Add new Group to List of Groups
Add Rectangle1 to new Group
Create new Group
Rectangle2_Iterator point to First Rectangle in Group
Increment Rectangle2_Iterator
Increment Group_Iterator
Add Rectangle1 to Group
YES
Rectangle2_Iterator == End of Group?
NO
Does Rectangle1 overlap with Rectangle2 ?
NO
YES

3. PseudoCode:

```
1. open data set file
2. while there are lines in the file:
    1. Read rectangle
    2. Push rectangle to rectangles vector
3. sort rectangles vector according to smallest X1 if there are rectangles have the same X1
   then sort them according to smallest Y1
4. for each rectangle1 in rectangles vector:
    1. for each group in list of groups:
        1. for each rectangle2 in group:
            1. if (rectangle1 overlap with rectangle2) then:
                1. break nearest for loop to go to next group
        2. if (reached end of group) means no rectangle in group overlap with
           rectangle1 then:
            1. add rectangle1 to this group
            2. break nearest for loop to go to next rectangle1 in
               rectangles vector
    2. if (reached end of list of groups) means rectangle1 did not fit in any of the groups
       then:
        1. create new group
        2. add rectangle1 to new group
        3. add new group to list of groups
5. for each group in list of groups:
    1. open output file
    2. for each rectangle in group:
        1. output rectangle to file
```

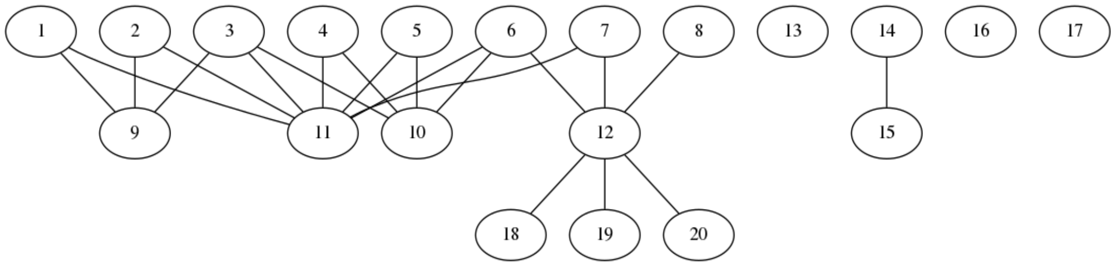
-So, this algorithm has complexity of O(n ^ 2) as in the worst case if all rectangles overlap with each other the algorithm would compare each rectangle with all other rectangles.

4. Output Table:

Data Set	Number of inputs	Number of output groups	Runtime (sec)	Memory usage (MB)
data_set_1.txt	5	1	0.004	0.083
data_set_2.txt	7	2	0.003	0.092
data_set_3.txt	20	2	0.005	0.095
data_set_4.txt	39	3	0.005	0.109
data_set_5.txt	77	3	0.003	0.116
data_set_6.txt	136	5	0.004	0.152
data_set_7.txt	216	4	0.005	0.152
data_set_8.txt	460	5	0.009	0.208
data_set_9.txt	741	12	0.013	0.317
data_set_10.txt	981	5	0.015	0.327
data_set_11.txt	5793	6	0.198	1.476
data_set_12.txt	6775	6	0.277	1.559
data_set_13.txt	7538	5	0.335	1.684
data_set_14.txt	8774	7	0.446	2.626
data_set_15.txt	9188	6	0.474	2.646
data_set_16.txt	25263575			

5. Testing Methodology:

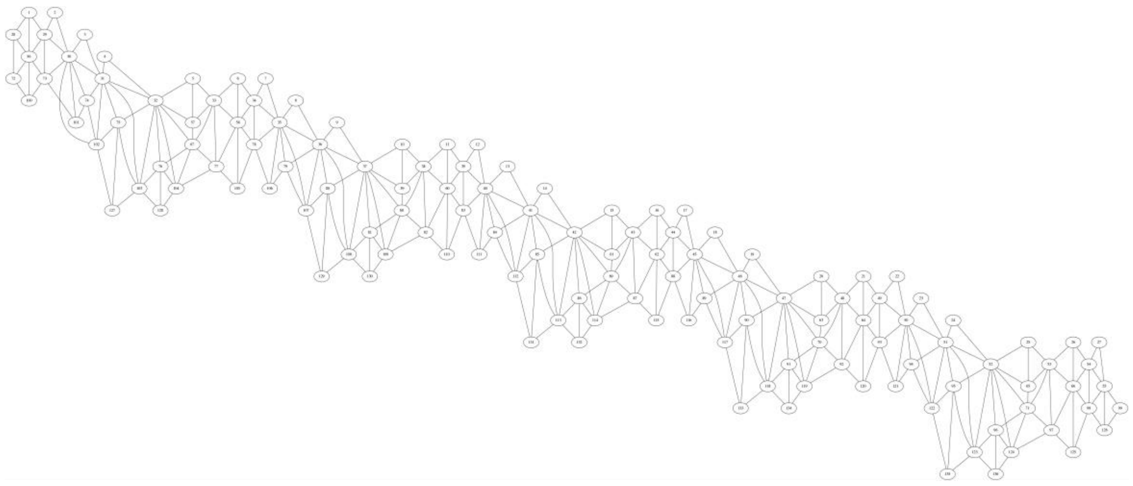
- I used a tool named graphviz which is a graph visualization software. By representing each two overlapping rectangles as two vertices that has edge between them. For example for data_set_3 the output would look like this:



- Vertices numbers represent order of insertion of rectangles in groups by the algorithm (So rectangle with label 1 would go first then rectangle 2 and so on, so until rectangle 8 all go in first group then insertion of rectangle 9 would create a second group).

- So, for this data set the algorithm output 2 groups and as seen from visualization that is the best solution.

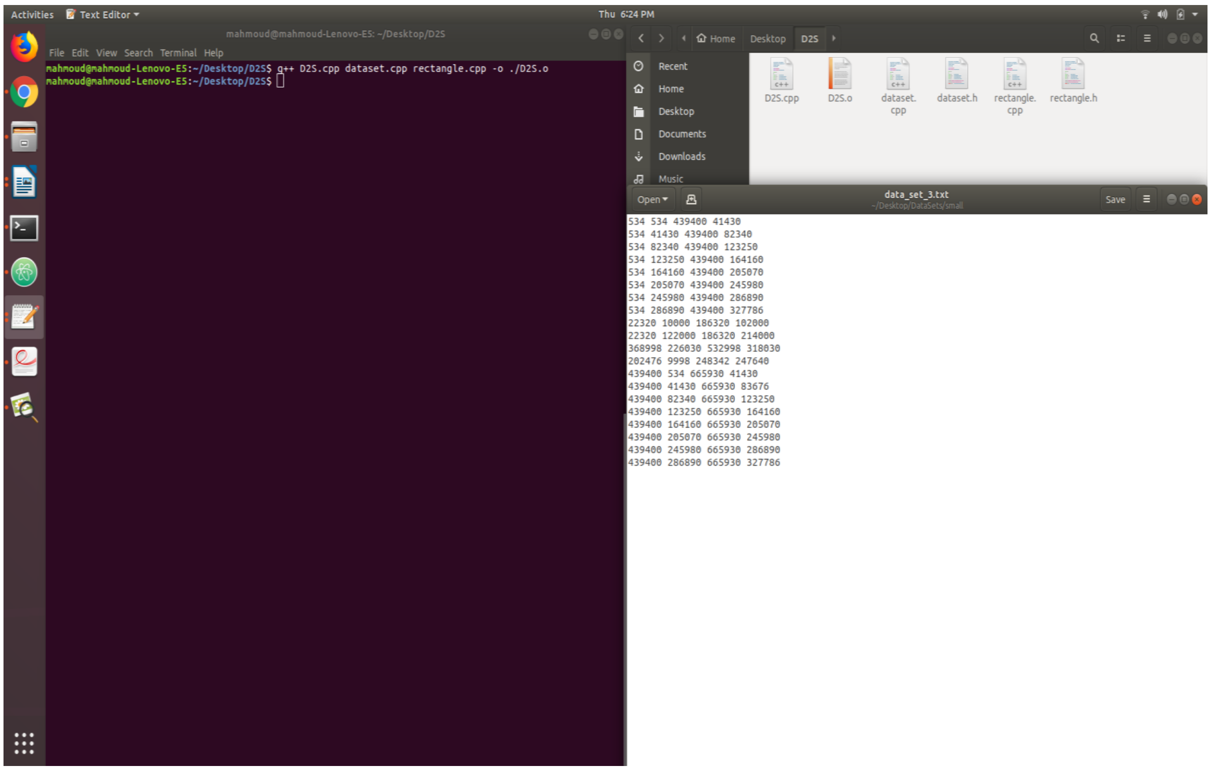
- Of Course, for large data set it would be hard to see from visualization what is the best solution. For example this is data_set_6 , output from algorithm is 5 groups but by sorting rectangles according to highest degree (degree of rectangle represent number of rectangles overlap this rectangle), the output would be 4 groups.



- So, I generated random rectangles don’t exceed 20 rectangles and see what the algorithm output for those rectangles would be, also visualize rectangles and see if the output was the right solution.

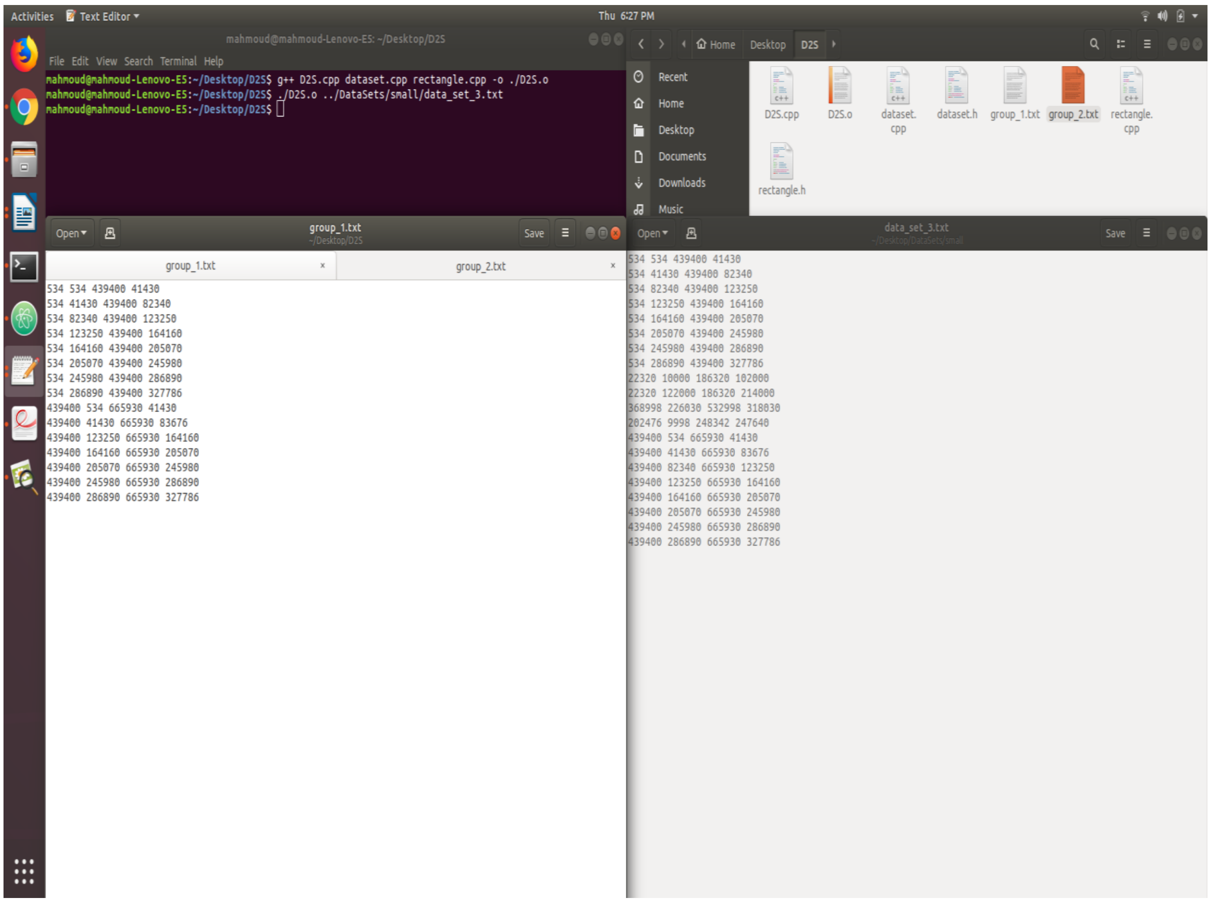
6. Screen Shots:

- Data_Set_3:



```
mahmoud@mahmoud-Lenovo-E5:~/Desktop/D2S$ g++ D2S.cpp dataset.cpp rectangle.cpp -o ./D2S.o
mahmoud@mahmoud-Lenovo-E5:~/Desktop/D2S$ 
```

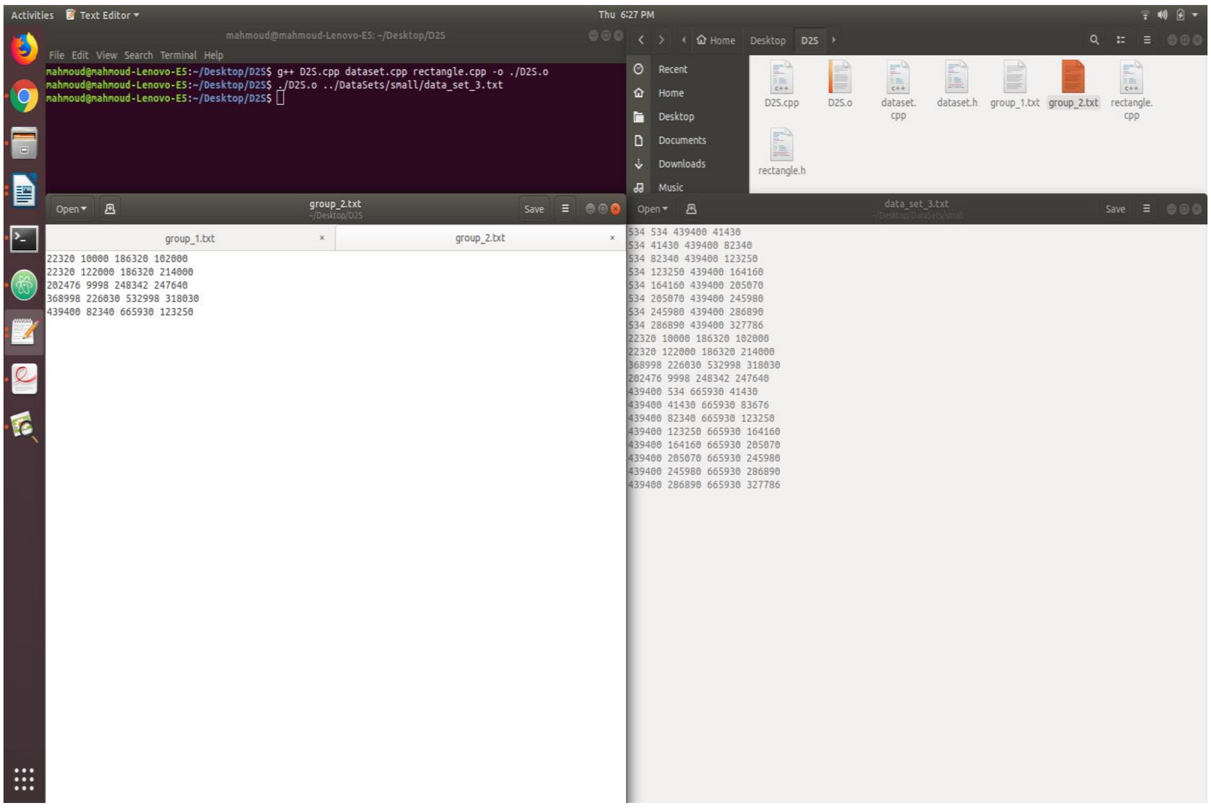
```
534 534 439400 41430
534 41430 439400 82340
534 82340 439400 123250
534 123250 439400 164160
534 164160 439400 205070
534 205070 439400 245980
534 245980 439400 286890
534 286890 439400 327786
22320 10000 186320 102000
22320 122000 186320 214000
368998 226030 532998 318030
202476 9998 248342 247640
439400 534 665930 41430
439400 41430 665930 83676
439400 82340 665930 123250
439400 123250 665930 164160
439400 164160 665930 205070
439400 205070 665930 245980
439400 245980 665930 286890
439400 286890 665930 327786
```



```
mahmoud@mahmoud-Lenovo-E5:~/Desktop/D2S$ g++ D2S.cpp dataset.cpp rectangle.cpp -o ./D2S.o
mahmoud@mahmoud-Lenovo-E5:~/Desktop/D2S$ ./D2S.o ../DataSets/small/data_set_3.txt
mahmoud@mahmoud-Lenovo-E5:~/Desktop/D2S$ 
```

```
534 534 439400 41430
534 41430 439400 82340
534 82340 439400 123250
534 123250 439400 164160
534 164160 439400 205070
534 205070 439400 245980
534 245980 439400 286890
534 286890 439400 327786
439400 534 665930 41430
439400 41430 665930 83676
439400 123250 665930 164160
439400 164160 665930 205070
439400 205070 665930 245980
439400 245980 665930 286890
439400 286890 665930 327786
```

```
534 534 439400 41430
534 41430 439400 82340
534 82340 439400 123250
534 123250 439400 164160
534 164160 439400 205070
534 205070 439400 245980
534 245980 439400 286890
534 286890 439400 327786
22320 10000 186320 102000
22320 122000 186320 214000
368998 226030 532998 318030
202476 9998 248342 247640
439400 534 665930 41430
439400 41430 665930 83676
439400 82340 665930 123250
439400 123250 665930 164160
439400 164160 665930 205070
439400 205070 665930 245980
439400 245980 665930 286890
439400 286890 665930 327786
```



```
mahmoud@mahmoud-Lenovo-E5:~/Desktop/D2S$ g++ D2S.cpp dataset.cpp rectangle.cpp -o ./D2S.o
mahmoud@mahmoud-Lenovo-E5:~/Desktop/D2S$ ./D2S.o ../DataSets/small/data_set_3.txt
mahmoud@mahmoud-Lenovo-E5:~/Desktop/D2S$
```

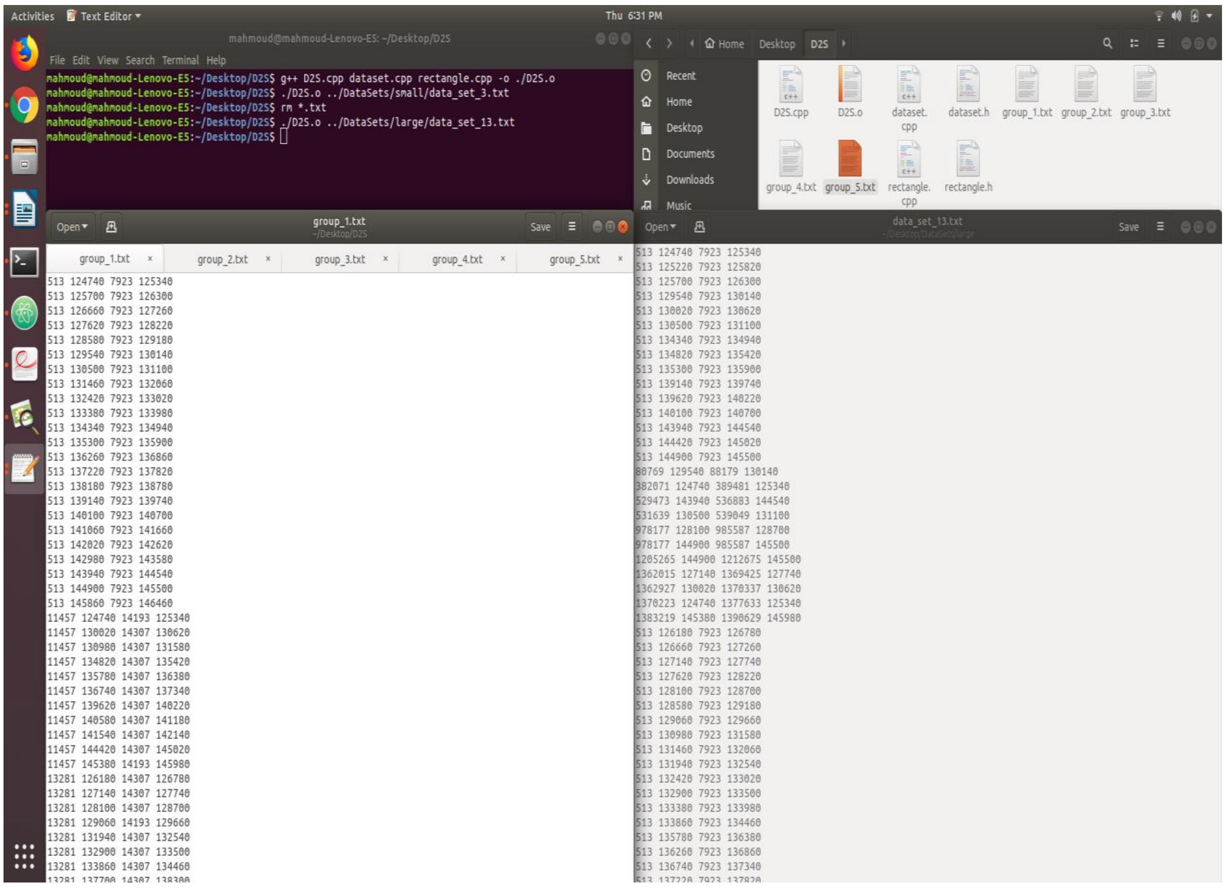
group_2.txt

```
22320 10000 186320 102000
22320 122000 186320 214000
202476 9998 248342 247640
368998 226030 532998 318030
439400 82340 665930 123250
```

data_set_3.txt

```
534 534 439400 41430
534 41430 439400 82340
534 82340 439400 123250
534 123250 439400 164160
534 164160 439400 205070
534 205070 439400 245980
534 245980 439400 286890
534 286890 439400 327786
22320 10000 186320 102000
22320 122000 186320 214000
368998 226030 532998 318030
202476 9998 248342 247640
439400 534 665930 41430
439400 41430 665930 83676
439400 82340 665930 123250
439400 123250 665930 164160
439400 164160 665930 205070
439400 205070 665930 245980
439400 245980 665930 286890
439400 286890 665930 327786
```

- Data_Set_13:



```
mahmoud@mahmoud-Lenovo-E5:~/Desktop/D2S$ g++ D2S.cpp dataset.cpp rectangle.cpp -o ./D2S.o
mahmoud@mahmoud-Lenovo-E5:~/Desktop/D2S$ ./D2S.o ../DataSets/small/data_set_3.txt
mahmoud@mahmoud-Lenovo-E5:~/Desktop/D2S$ rm *.txt
mahmoud@mahmoud-Lenovo-E5:~/Desktop/D2S$ ./D2S.o ../DataSets/large/data_set_13.txt
mahmoud@mahmoud-Lenovo-E5:~/Desktop/D2S$
```

group_1.txt

```
513 124740 7923 125340
513 125700 7923 126300
513 126660 7923 127260
513 127620 7923 128220
513 128580 7923 129180
513 129540 7923 130140
513 130500 7923 131100
513 131460 7923 132060
513 132420 7923 133020
513 133380 7923 133980
513 134340 7923 134940
513 135300 7923 135900
513 136260 7923 136860
513 137220 7923 137820
513 138180 7923 138780
513 139140 7923 139740
513 140100 7923 140700
513 141060 7923 141660
513 142020 7923 142620
513 142980 7923 143580
513 143940 7923 144540
513 144900 7923 145500
513 145860 7923 146460
11457 124740 14193 125340
11457 130020 14307 130620
11457 130980 14307 131580
11457 134820 14307 135420
11457 135780 14307 136380
11457 136740 14307 137340
11457 139620 14307 140220
11457 140580 14307 141180
11457 141540 14307 142140
11457 144420 14307 145020
11457 145380 14193 145980
13281 126180 14307 126780
13281 127140 14307 127740
13281 128100 14307 128700
13281 129060 14193 129660
13281 131940 14307 132540
13281 132900 14307 133500
13281 133860 14307 134460
13281 137700 14307 138300
```

data_set_13.txt

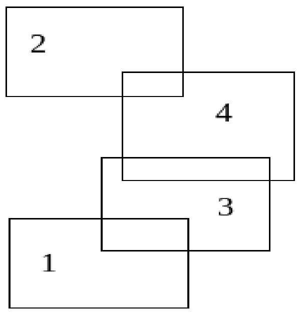
```
513 124740 7923 125340
513 125220 7923 125820
513 125700 7923 126300
513 129540 7923 130140
513 130020 7923 130620
513 130500 7923 131100
513 134340 7923 134940
513 134820 7923 135420
513 135300 7923 135900
513 139140 7923 139740
513 139620 7923 140220
513 140100 7923 140700
513 143940 7923 144540
513 144420 7923 145020
513 144900 7923 145500
80769 129540 88179 130140
382071 124740 389481 125340
529473 143940 536883 144540
531639 130500 539049 131100
978177 128100 985587 128700
978177 144900 985587 145500
1205265 144900 1212675 145500
1362015 127140 1369425 127740
1362927 130020 1370337 130620
1370223 124740 1377633 125340
1383219 145380 1390629 145980
513 126180 7923 126780
513 126660 7923 127260
513 127140 7923 127740
513 127620 7923 128220
513 128100 7923 128700
513 128580 7923 129180
513 129060 7923 129660
513 130980 7923 131580
513 131460 7923 132060
513 131940 7923 132540
513 132420 7923 133020
513 132900 7923 133500
513 133380 7923 133980
513 133860 7923 134460
513 135780 7923 136380
513 136260 7923 136860
513 136740 7923 137340
513 137220 7923 137820
```

7. Notes:

- The algorithm does not guarantee to output minimum number of groups for all data sets. As algorithm depends on the order of insertion of rectangles into groups, because when a rectangle enter group it makes restrictions on which rectangles in data set should enter the group.

- So that is why in the beginning of algorithm I sort rectangles according to their X1. But this also does not guarantee to output minimum number of groups, but it output smaller number of groups than insertion without sorting.

- For example, this data set the algorithm would output more number of groups:



- For this data set the algorithm would insert rectangles by the order shown as it sorts them according to their X1 so after inserting first two rectangles in first group it would need another two groups for the remaining rectangles. So, it would output three groups. Although right minimum number of groups is two groups. If the algorithm sorted according to Y1 it would output two groups only.

- I tried to sort rectangles:

- in descending order according to their degree (degree of rectangle represent number of rectangles overlap this rectangle)
- according to smallest y1 then smallest x1. (4 groups only in data_set_4 and 6 groups only in data_set_14)
- according to smallest x1 then smallest y1.
- according to smallest area.
- according to projection of bottom left point on 45-degree line.

- But sorting according to smallest x1 then smallest y1 produced smaller number of groups in most of the data sets.

- Also, this algorithm can't find solution for huge data set as algorithm has complexity of O(n^2).