REPORT OF SOFTWARE ASSIGNMENT-1

MEMBERS- AYUSH KUMAR-2023CS10600

VIRAAJ NAROLIA-2023CS10552

Problem statement:

Given:

• a set of rectangular logic gates g1, g2...gn

• width and height of each gate gi

Write a program to assign locations to all gates in a plane so that:

• no two gates are overlapping

• the bounding box of the entire circuit (smallest rectangle that encloses all gates) has minimum area.

ALGORITHM1: FFDH (First Fit Decreasing Height)

Step1: We iterate from maximum width of a gate to sum of widths of all gates.

Step2a: In each iteration we sort the rectangles in decreasing order of height.

Step 2b: Then place the rectangles in a single layer within the box of a certain width until we reach the width limit. If we reach width limit, a new layer is created at the maximum height of previous layer and then we place further rectangles on that layer and repeat this process until all rectangles are placed.

Step3: We choose the bounding box whose area is minimum.

Time complexity analysis:

Step1: Iterating from maximum width of a gate to sum of widths of all gates:

- Time Complexity: O(s) where s is the sum of widths of all gates except the maximum width.
- Explanation: The loop runs for s iterations, so the time complexity would be O(s).

Step2a: Sorting Rectangles by Height:

- Time Complexity: O(nlogn)
- Explanation: The algorithm starts by sorting the rectangles in decreasing order of their height. Sorting n rectangles takes O(nlogn) time.

Step 2b: Placing Rectangles in Layers:

- Time Complexity: O(n×m)
- Explanation: After sorting, the algorithm places each rectangle into the first layer where it fits. If it doesn't fit in the current layer, a new layer is created. In the worst case, placing each rectangle might require checking every existing layer. If there are m layers, the worst-case time complexity for placement is O(n×m)
- However, typically m (the number of layers) is much smaller than n, making the placement step effectively linear, O(n), in many practical scenarios.

Combining all the steps, the overall time complexity of the FFDH algorithm is:

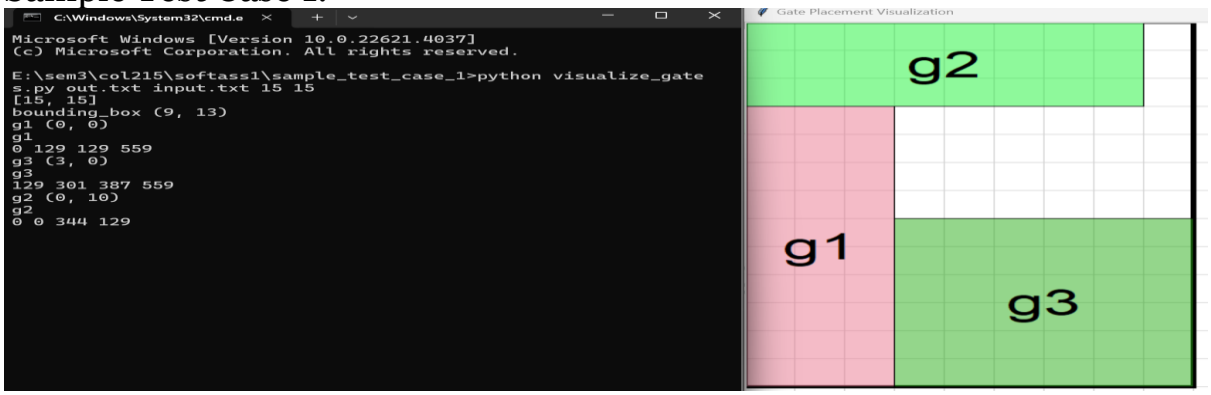
- Total Time Complexity: O(s*(nlogn+n×m))

Given that the number of layers m is generally much smaller than the number of rectangles n, the sorting step O(nlogn) often dominates, resulting in the overall time complexity:

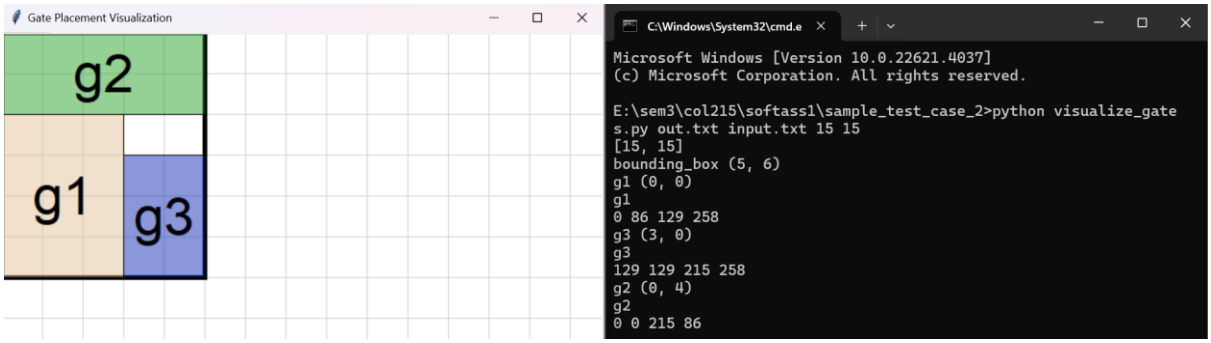
- Overall Time Complexity: O(s*nlogn)

Snapshots of the visualization:

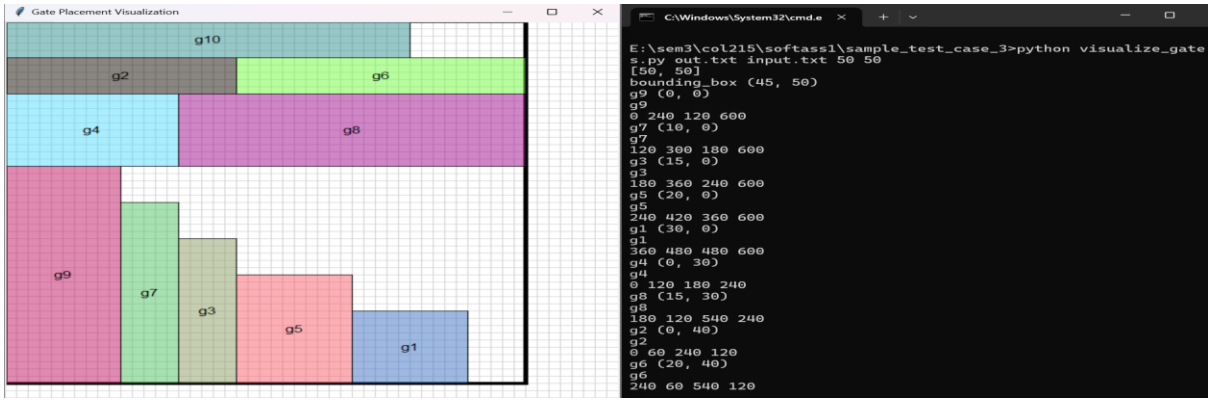
Sample Test Case 1:



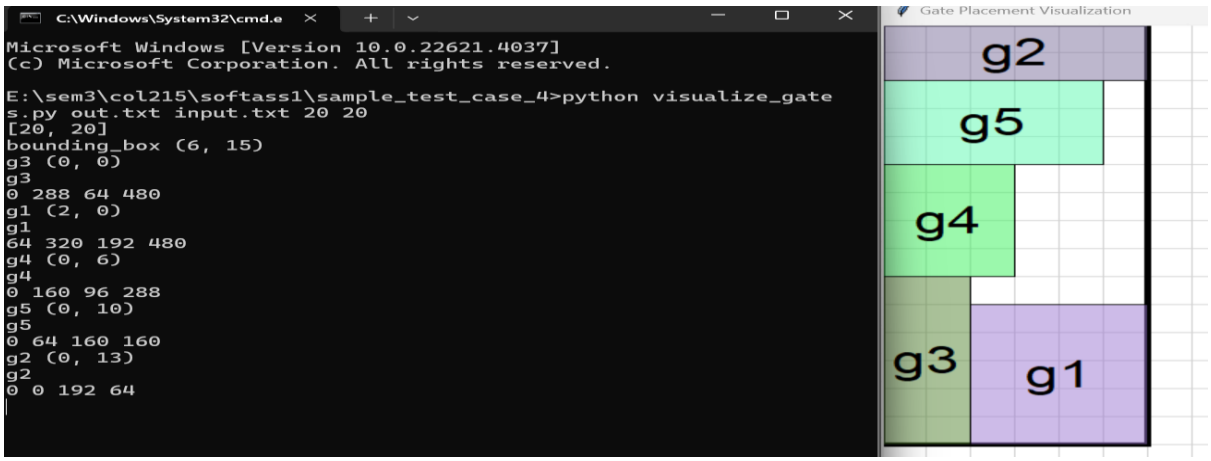
Sample Test Case 2:



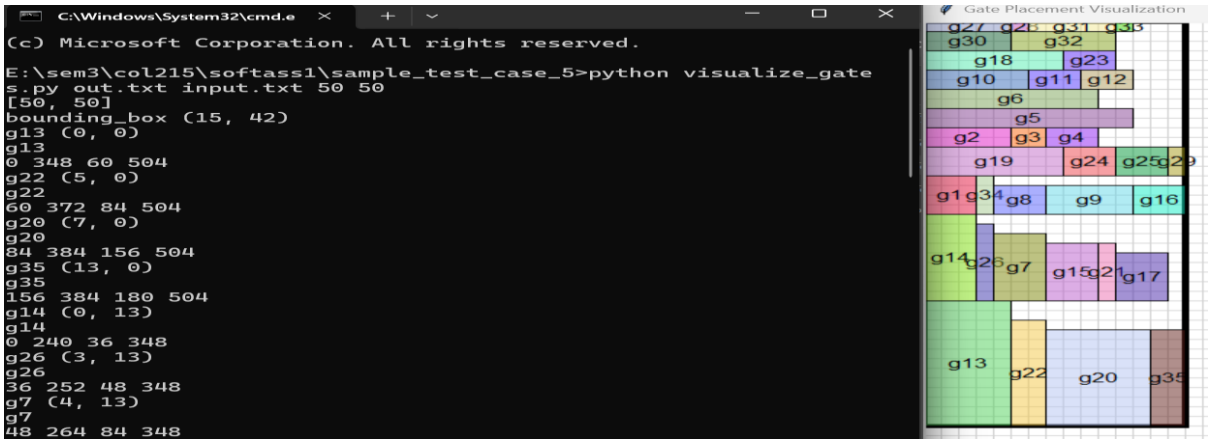
Sample Test Case 3:



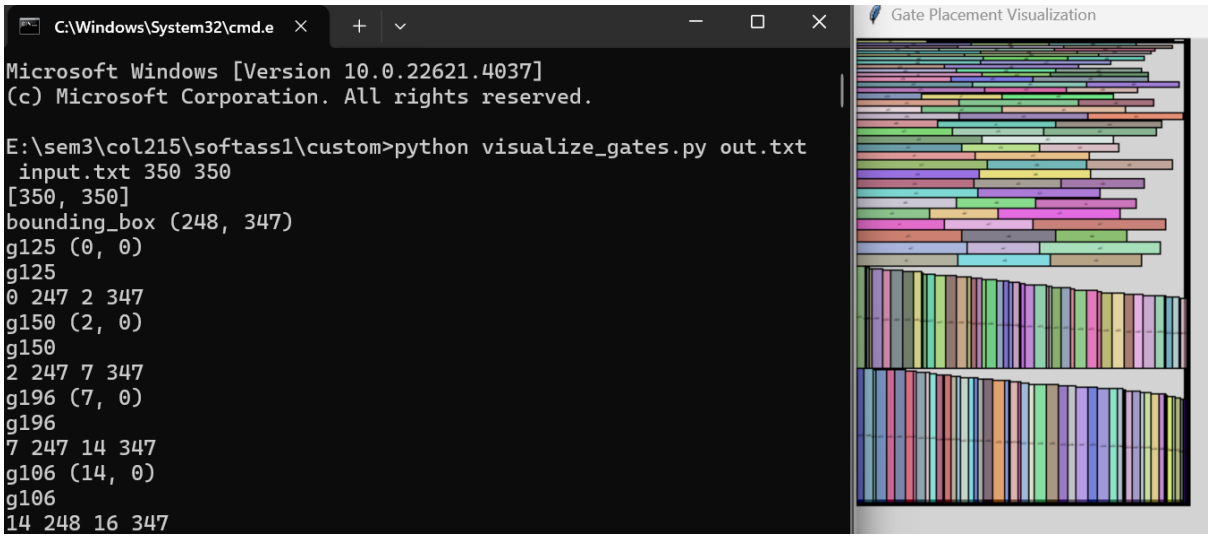
Sample Test case 4:



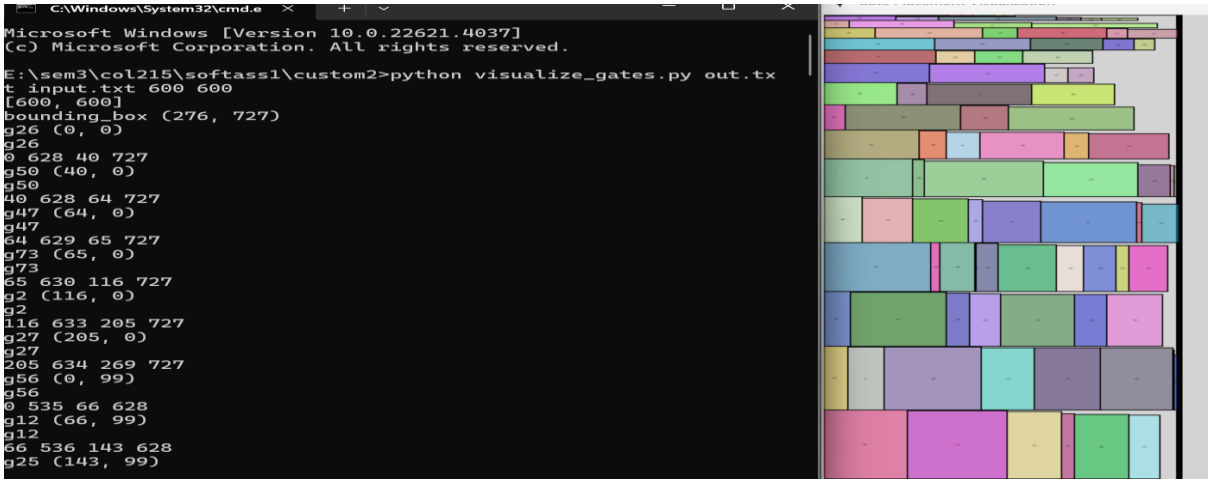
Sample Test Case 5:



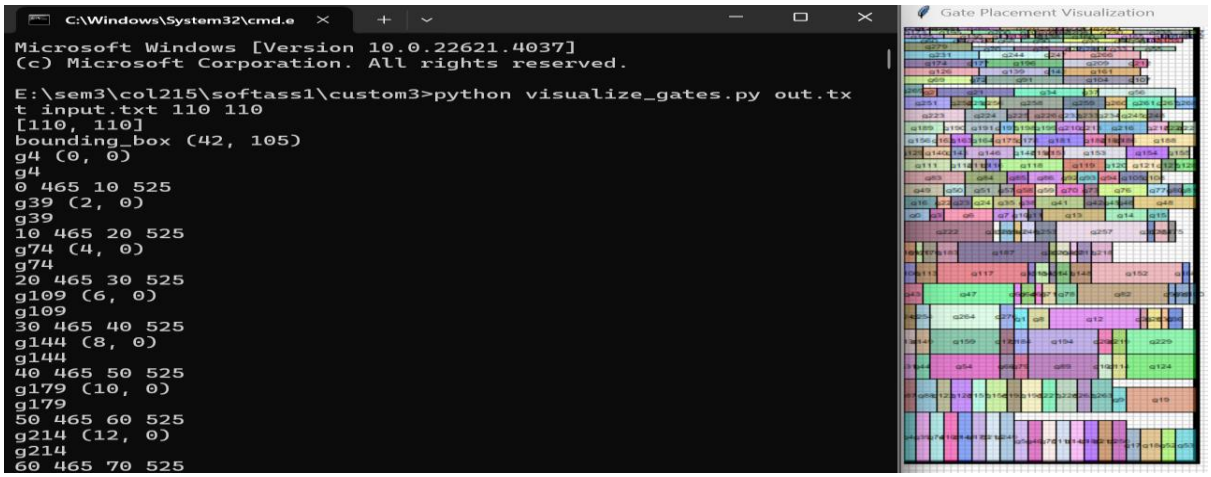
Custom Test Case 1:



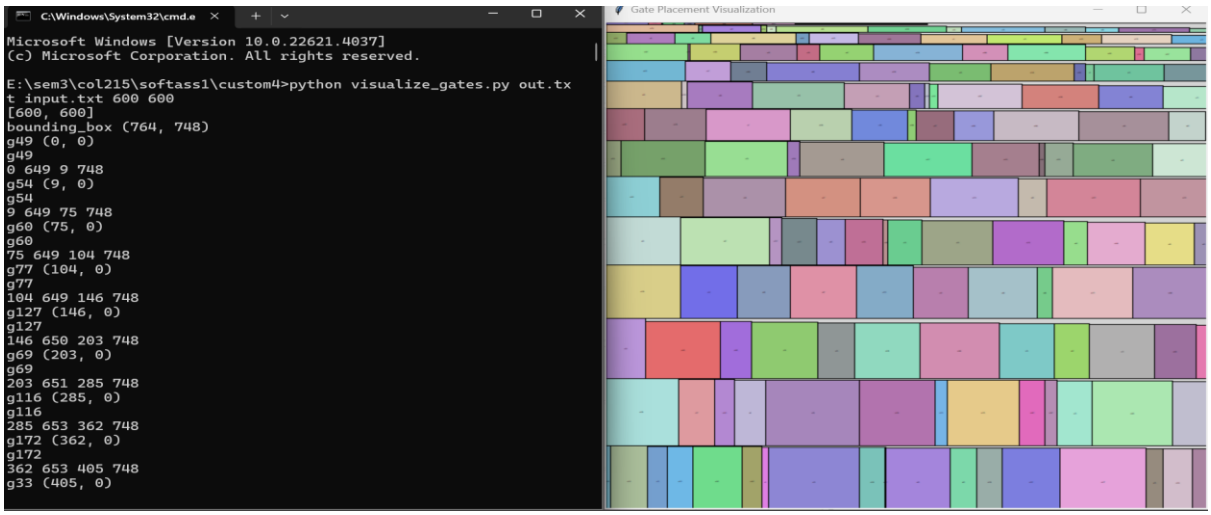
Custom Test Case 2:



Custom Test Case 3:



Custom Test Case 4:



ALGORITHM 2: Sleator Algorithm

Step1: Sort the rectangles in order of decreasing height and take a bin of width ‘w’.

Step2: Place all rectangles of width>w/2 one upon another like a stack and let ho be the height of the stack.

Step3: By taking ho as a shelf place further rectangles on this until width of bin is reached.

Step4: Draw a line at width w/2 and let hl be the maximum height in left side (i.e. 0<=width<=w/2) and hr be the maximum height in right side (i.e. w/2<width<=w).

Step5: If hl<=hr. Take new level to be hl and place further rectangles on this level and only in left side. Else If hr<hl. Take new level to be and place further rectangles on this level and only in right side.

Step6: Update your hl and hr and again repeat until all rectangles are placed.

Time Complexity Analysis:

Step1: Iterating from maximum width of a gate to sum of widths of all gates:

- Time Complexity: O(s) where s is the sum of widths of all gates except the maximum width.
- Explanation: The loop runs for s iterations, so the time complexity would be O(s).

Step2a: Sorting Rectangles by Height:

- Time Complexity: O(nlogn)
- Explanation: The algorithm starts by sorting the rectangles in decreasing order of their height. Sorting n rectangles takes O(nlogn) time.

Step 2b: Placing Rectangles in Layers:

- Time Complexity: O(n×m)
- Explanation: After sorting, the algorithm places each rectangle into the first layer where it fits. If it doesn't fit in the current layer, a

new layer is created. In the worst case, placing each rectangle might require checking every existing layer. If there are m layers, the worst-case time complexity for placement is O(n×m)
- However, typically m (the number of layers) is much smaller than n, making the placement step effectively linear, O(n), in many practical scenarios.

Combining all the steps, the overall time complexity of the Sleator algorithm is:

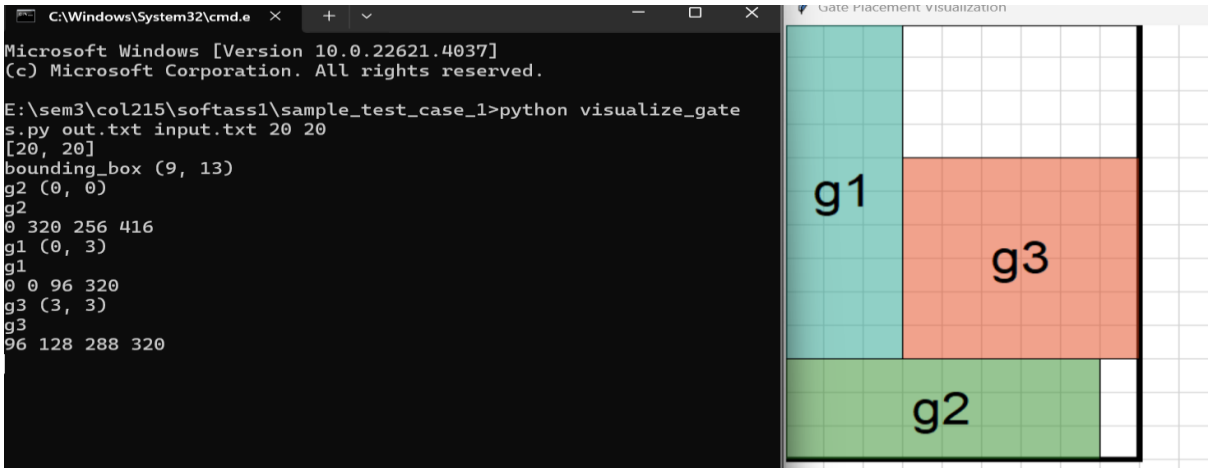
- Total Time Complexity: O(s*(nlogn+n×m))

Given that the number of layers m is generally much smaller than the number of rectangles n, the sorting step O(nlogn) often dominates, resulting in the overall time complexity:

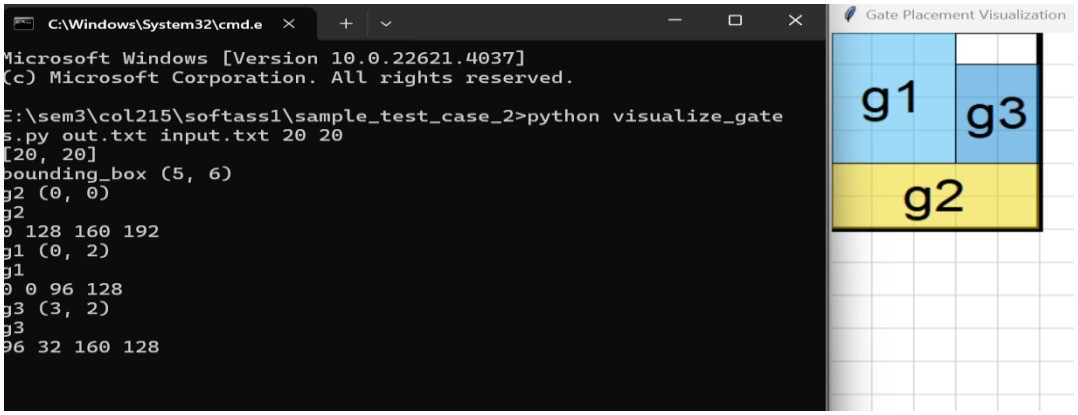
- Overall Time Complexity: O(s*nlogn)

Snapshots of the visualization:

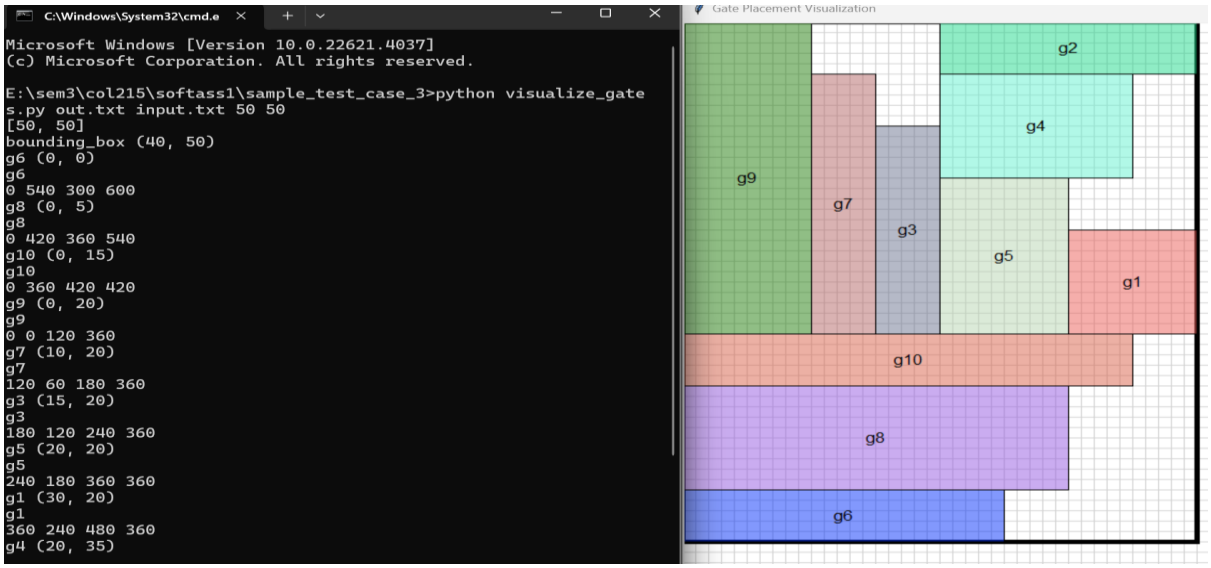
Sample Test Case 1:



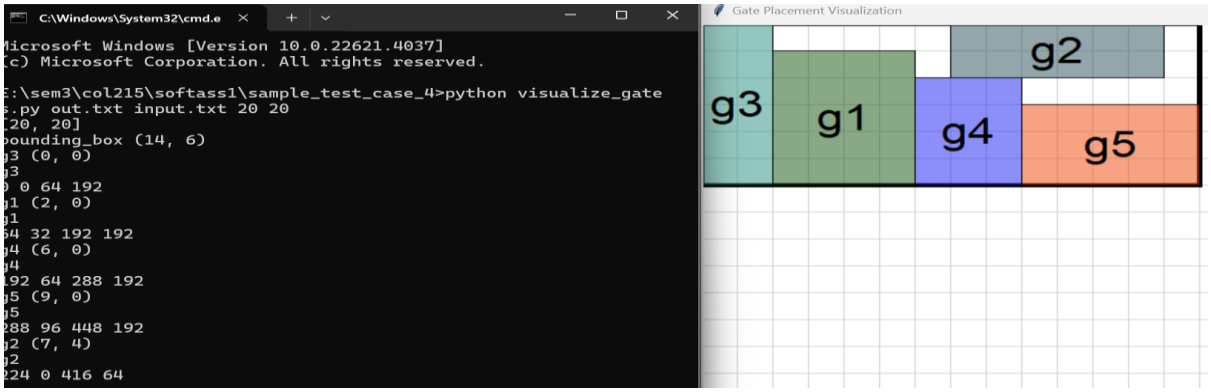
Sample Test Case 2:



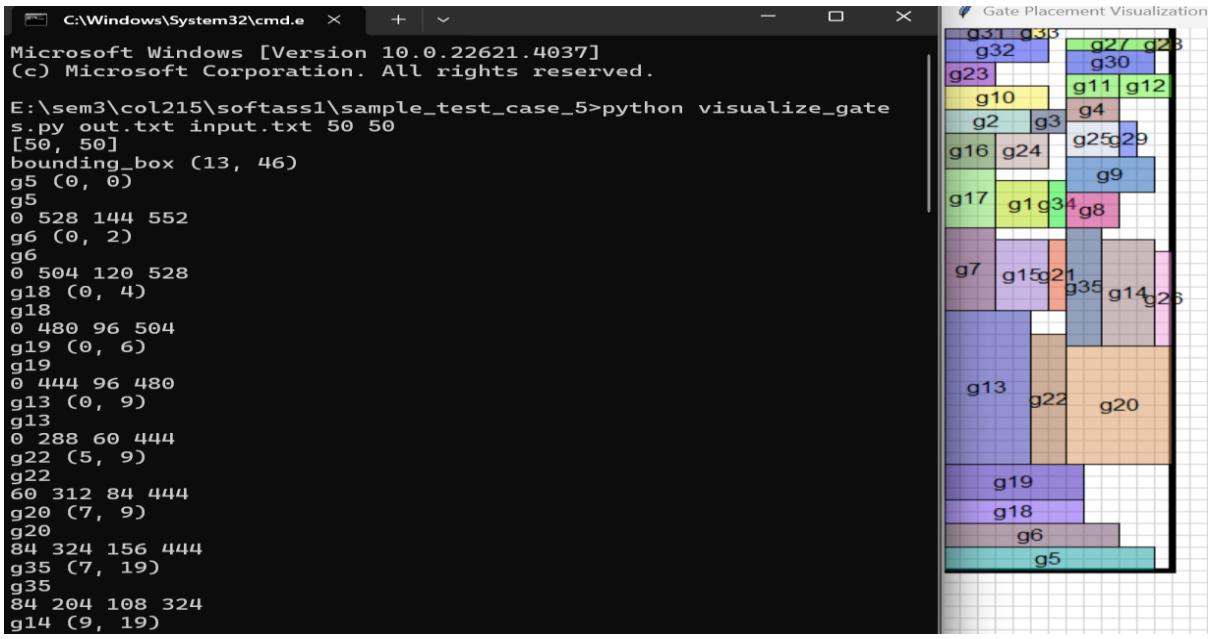
Sample test Case 3:



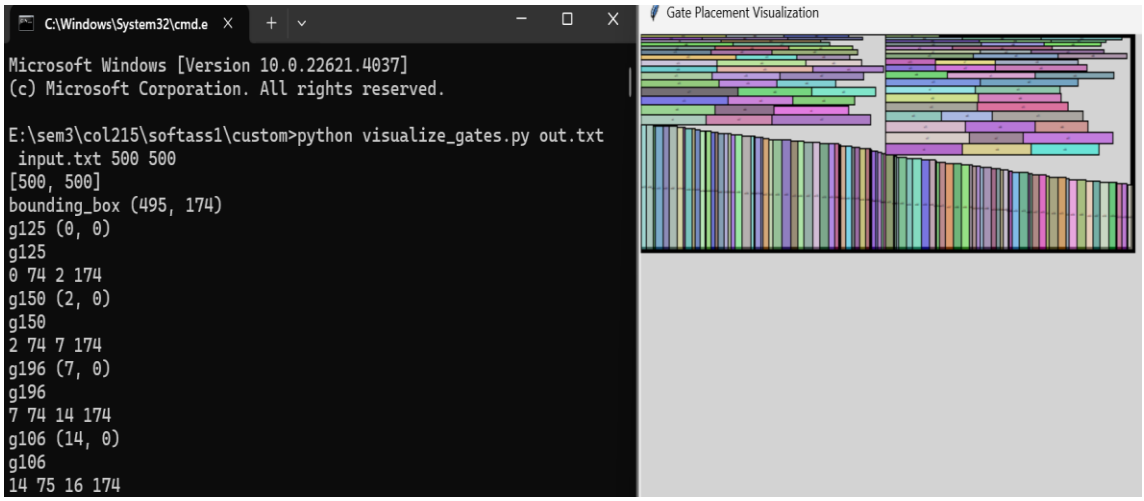
Sample Test Case 4:



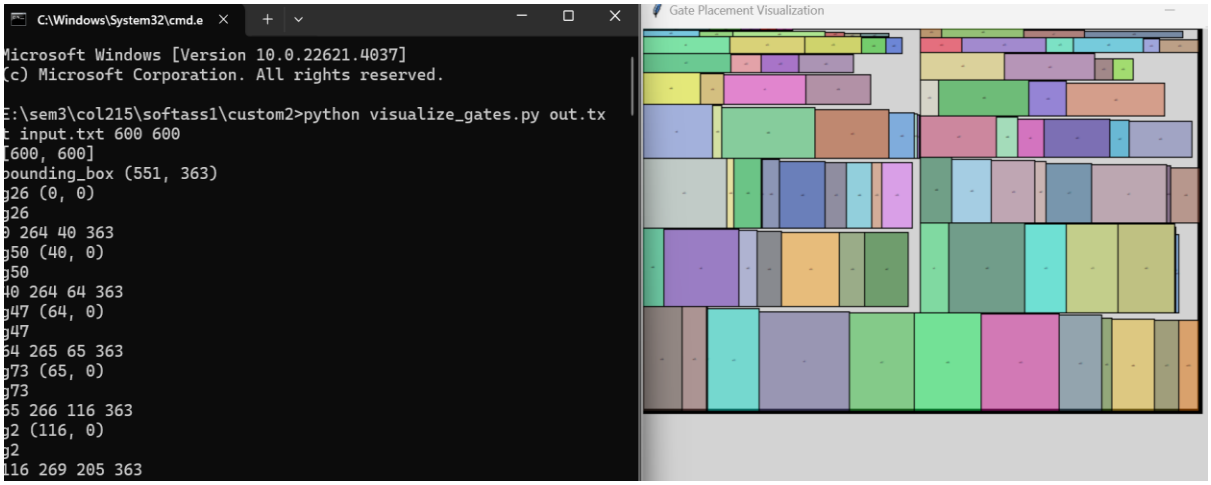
Sample Test Case 5:



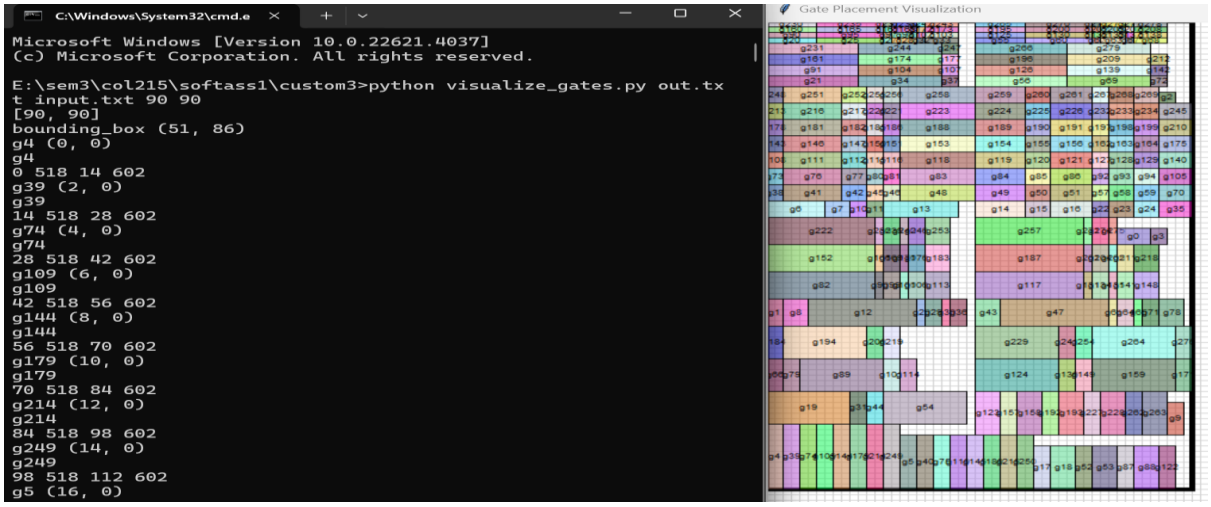
Custom Test Case 1:



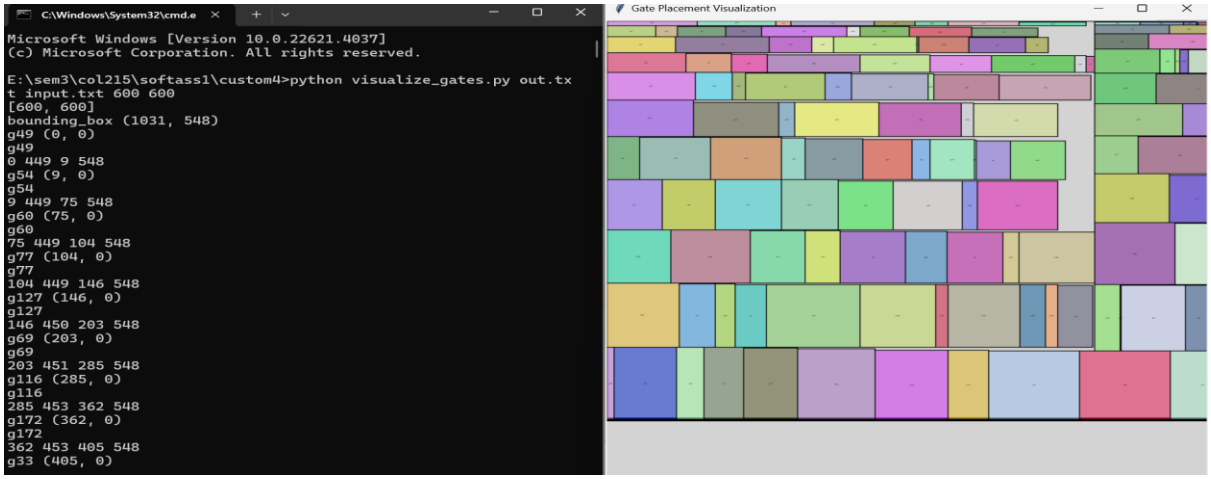
Custom Test Case 2:



Custom Test Case 3:



Custom Test Case 4:



Algorithm3: Grid packing algorithm

Step1: Sort the rectangles in order of decreasing height.

Step2: Take a grid of initially 1x1 size. This will represent our bin and we initialise this with False representing that it has not yet filled at that cell.

Step3: Start iterating in grid for y from 0 to length of grid and for each y iterate for X from 0 to length of grid. And for each pair of (x,y) check if you can pace the rectangle with its bottom left corner on (x,y). If you can then place it and break the nested loop. If not then continue iterating. If after nested loops you have not placed your rectangle it means that your grid is not of appropriate size so, increase the dimensions of your grid and then place the rectangle. Continue this process until all rectangles are placed.

Time complexity analysis:

Step1: Sorting Rectangles by Height:

- Time Complexity: O(nlogn)
- Explanation: The algorithm starts by sorting the rectangles in decreasing order of their height. Sorting n rectangles takes O(nlogn) time.

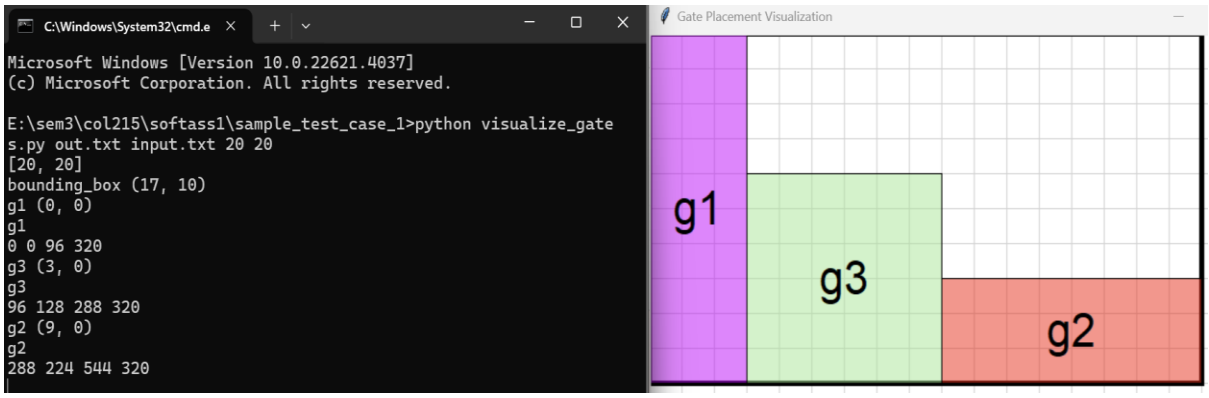
Step2: Placing Rectangles:

- O(n×N^2×width×height)
  where NxN is the final grid size after all placements
  n- for number of rectangles
  N^2- for iterating in grid (taking worst case scenario)
  Width, height- for placing a rectangle in the grid. Taking max height and width.

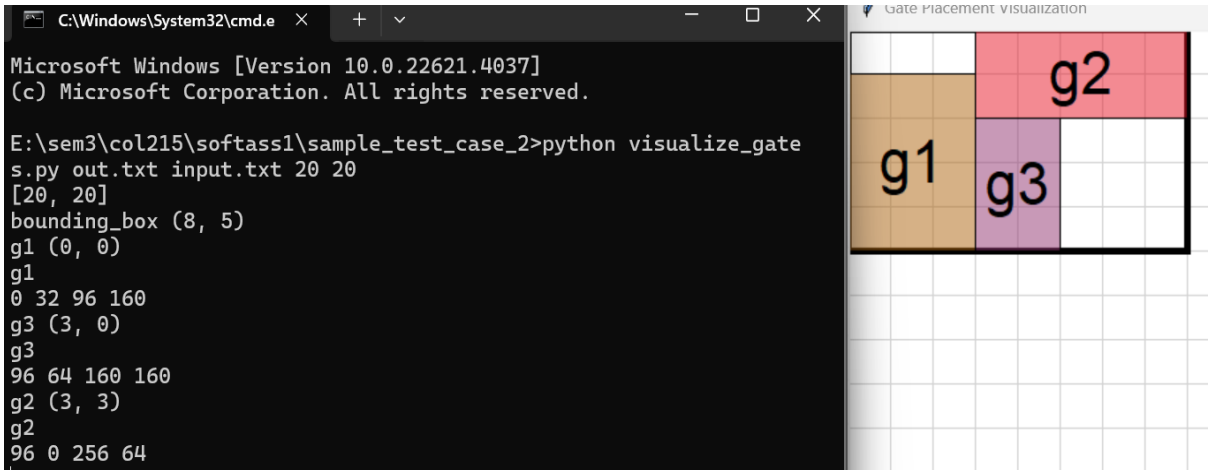
Overall time complexity- O(n x N^2 x width x height)

Snapshots of the visualization:

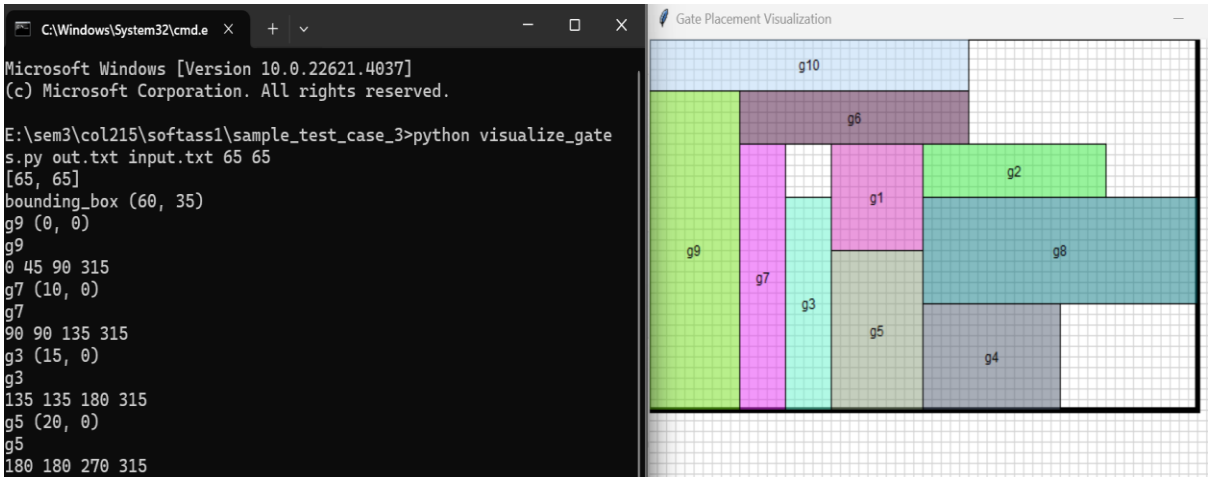
Sample Test Case 1:



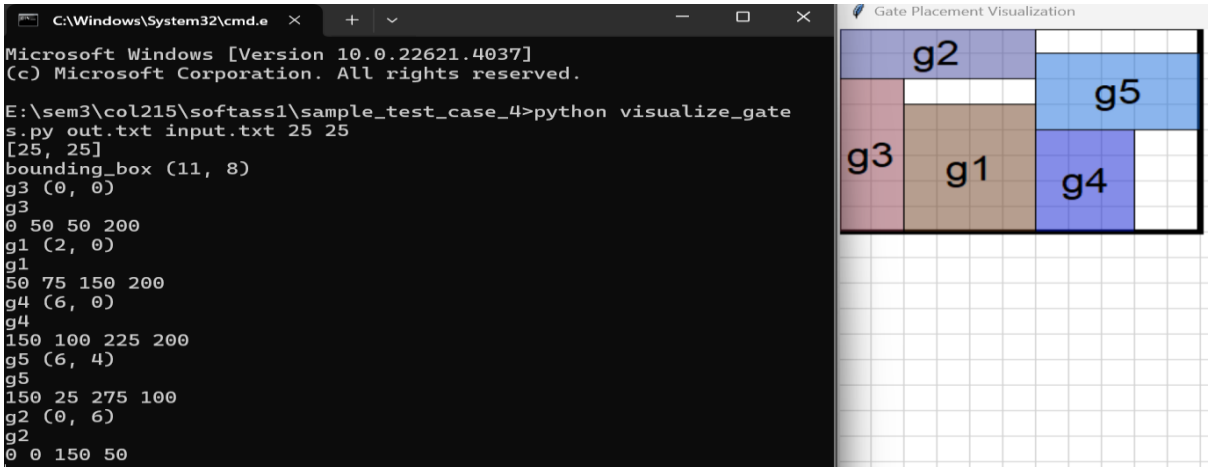
Sample Test Case 2:



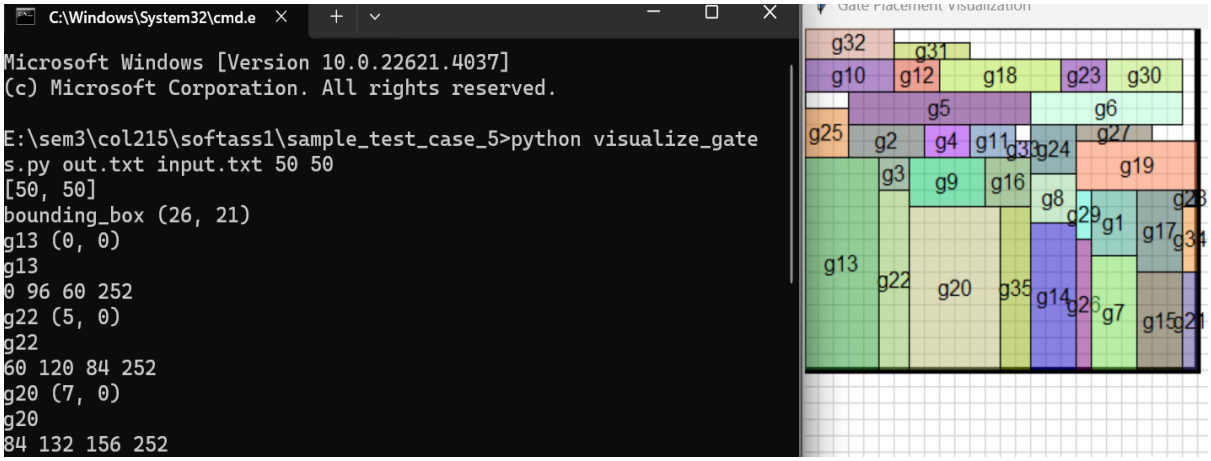
Sample test Case 3:



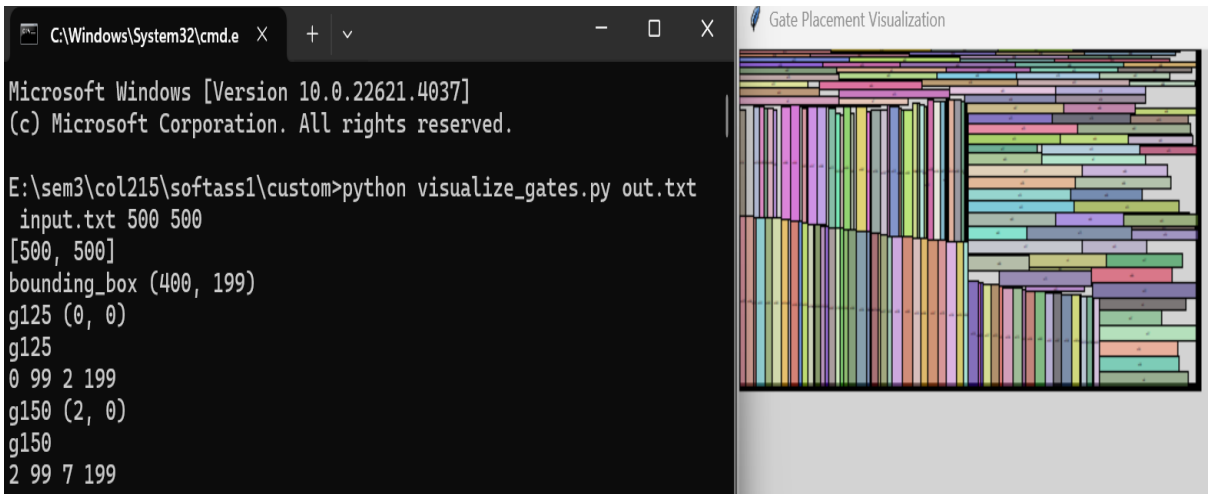
Sample Test Case 4:



Sample Test Case 5:

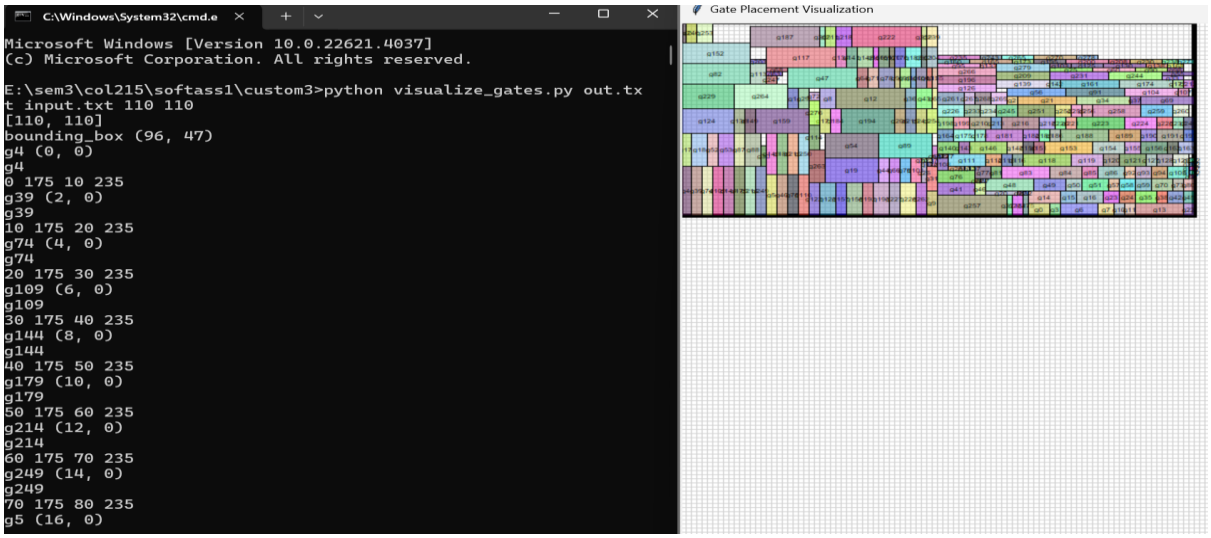


Custom Test Case 1:

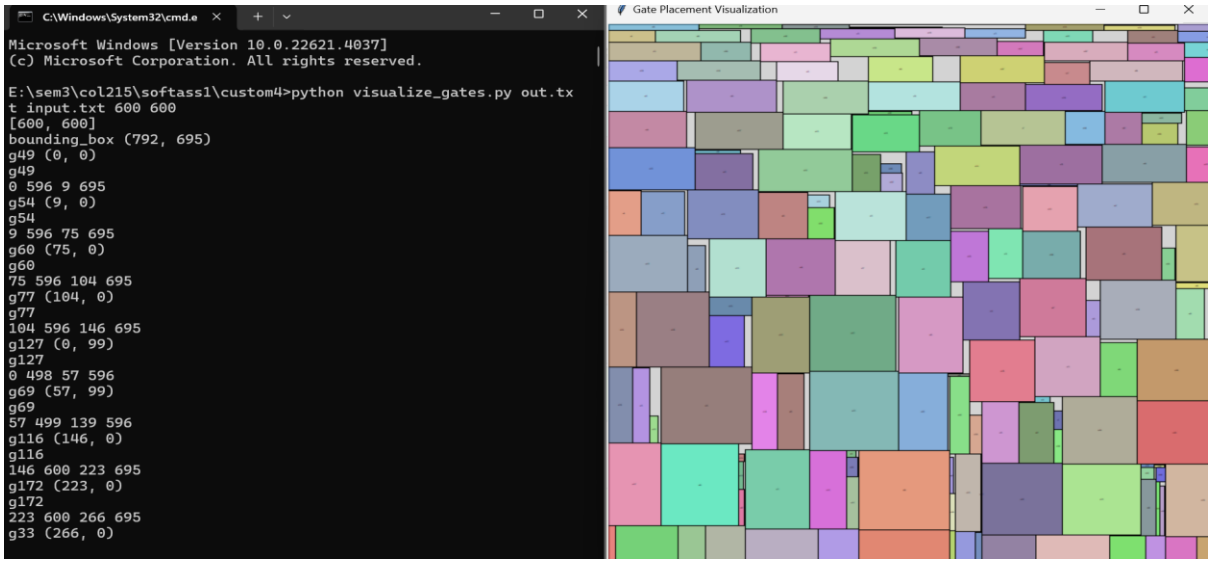


Custom Test Case 2:

Custom Test Case 3:



Custom Test Case 4:



Analysis of algorithms:

1)FFDH:

It works better when there is less variation in heights of rectangles because then there will be less wastage of space when we move to new level, for eg- Sample Test case 2,5 and Custom testcases.

It gives bad output when there is significant difference in heights of rectangles when they are already arranged in sorted decreasing order, for eg- Sample Test case 1,3 and 4.

Test Case	Sum of Area of all gates	Output Area	Packing Efficiency (in%)
Sample test Case 1	90	117	76.92
Sample test Case 2	28	30	93.33
Sample test Case 3	1625	2250	72.22
Sample test Case 4	71	90	78.88
Sample test Case 5	500	630	79.36
Custom Test case 1	73341	86056	85.22
Custom Test case 2	181608	200652	90.50
Custom Test case 3	4000	4410	90.70
Custom Test case 4	524855	571472	91.84

2)Sleator algorithm:

It is less effective when there is significant variation in heights of rectangles when they are already arranged in sorted decreasing order and also less effective when widths are around w/2 width. Because in that case the final height will be more in order to place all rectangles.

It is almost effective in every case except the sample test case 1.

Test Case	Sum of Area of all gates	Output Area	Packing Efficiency (in%)
Sample test Case 1	90	117	76.92
Sample test Case 2	28	30	93.33
Sample test Case 3	1625	2000	81.25
Sample test Case 4	71	84	84.52
Sample test Case 5	500	598	83.61
Custom Test case 1	73341	86130	85.15
Custom Test case 2	181608	200013	90.79
Custom Test case 3	4000	4386	91.19
Custom Test case 4	524855	564988	92.89

3)Grid algorithm:

It is less efficient in some testcases. When the rectangles are not sufficient to fill the whole grid i.e. the grid is small to accommodate further rectangles but if after increase its size it becomes too big for remaining rectangles. For eg-sample test case1,2,3,4 and custom test case2.

But in remaining testcases it is very efficient for eg-sample testcase 5 and custom test case 1,3 and 4.

Test Case	Sum of Area of all gates	Output Area	Packing Efficiency (in%)
Sample test Case 1	90	170	52.94
Sample test Case 2	28	40	70
Sample test Case 3	1625	2100	77.38
Sample test Case 4	71	88	80.68
Sample test Case 5	500	546	91.57
Custom Test case 1	73341	79600	92.13
Custom Test case 2	181608	291879	62.22
Custom Test case 3	4000	4512	88.65
Custom Test case 4	524855	550440	95.35

Our algorithms would give better results for higher number of logic gates.