REPORT OF SOFTWARE ASSIGNMENT-2

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

• the input and output pin locations (x and y co-ordinates) on the boundary of each gate gi.p1, gi.p2, ..., gi.pm (where gate gi has m pins)

• the pin-level connections between the gates

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

• no two gates are overlapping

• the sum of estimated wire lengths for all wires in the whole circuit is minimized

Algorithm 1:

Step 1: First, we form the clusters of interconnected gates. By interconnected gates we can explore every gate of that cluster using DFS (depth first search) and starting from any gate.

Step2: Then we applied our different algorithms on each such cluster and finally placed all such clusters side by side.

Step3: Sort the rectangles in order of decreasing number of right connections.

Step4: 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.

Step5: 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: For cluster formation:

We are applying DFS on List of Gates to form clusters. In this we are starting from a gate1 and then going to every gate connected to gate1 and we mark the gate1 as visited and then applying recursion on every connected gate. So, the worst-case scenario will be when all the gates are interconnected (only one cluster) so in this case we will be visiting every gate n times (n=no. of gates) and there are n gates, so complexity will be O(n^2)

Step2: 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.

Step3: Placing Rectangles:

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

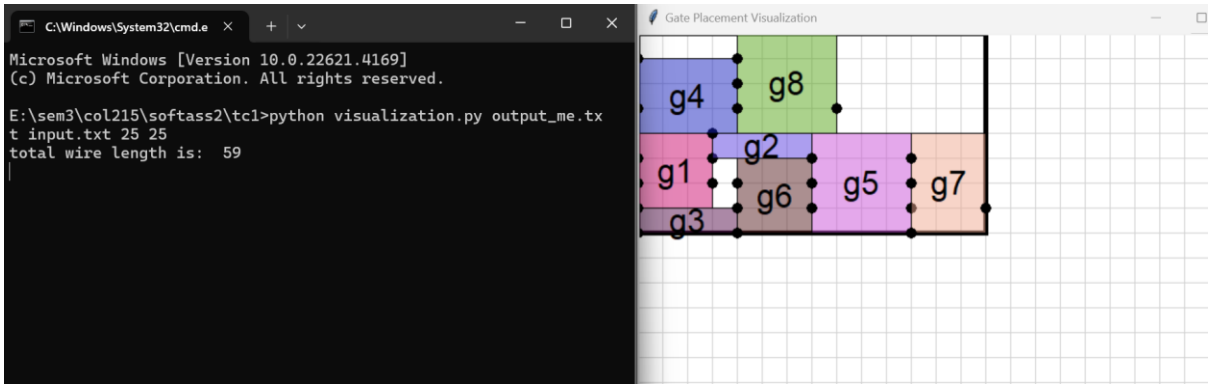
Overall time complexity- O(n x N^2 x width x heightxC+nt^2)

nt^2 is for cluster formation.

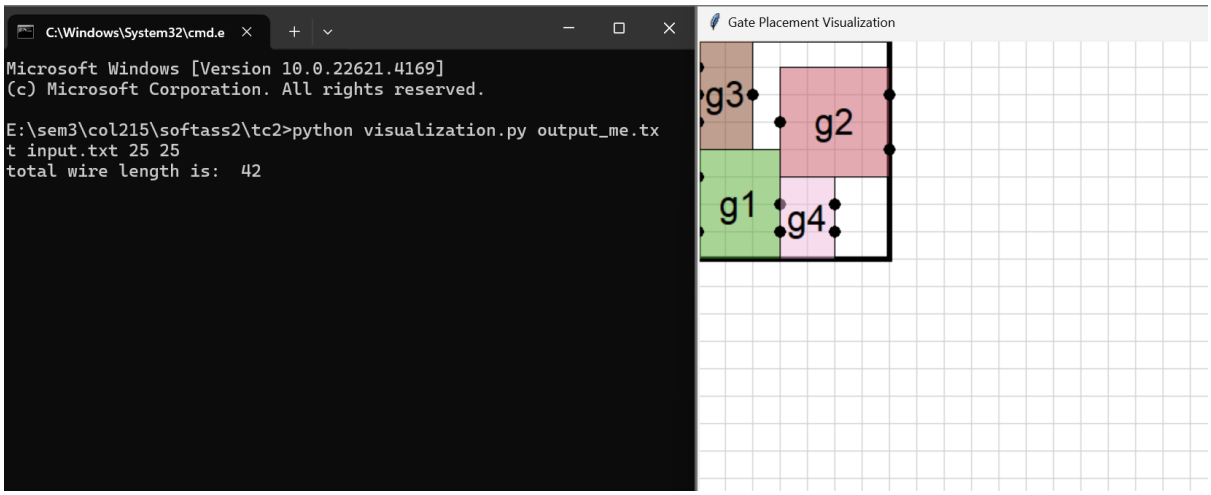
C is number of clusters.

Snapshots of the visualization:

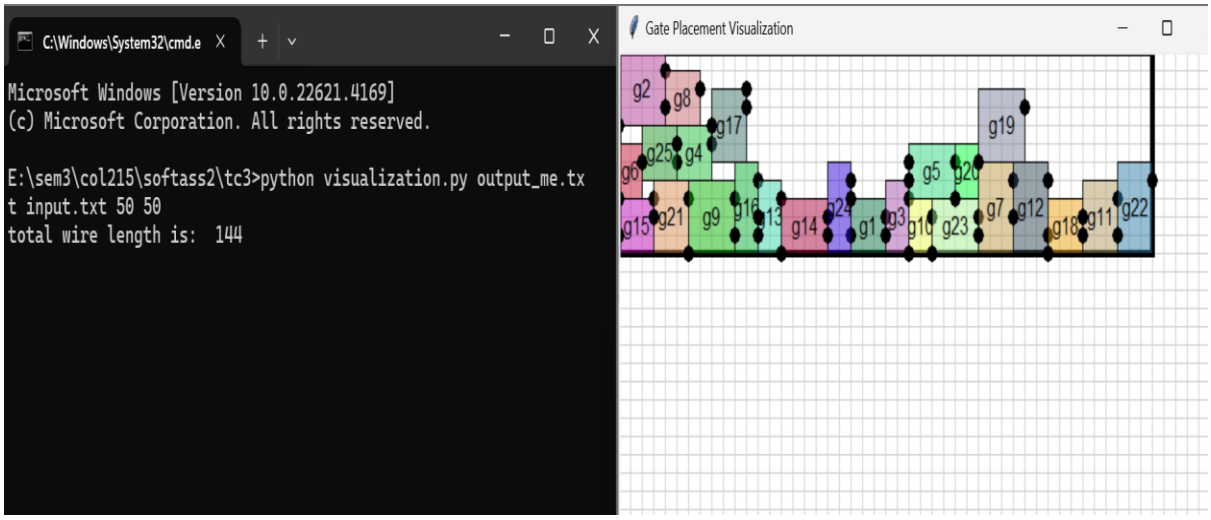
Sample Test Case 1:



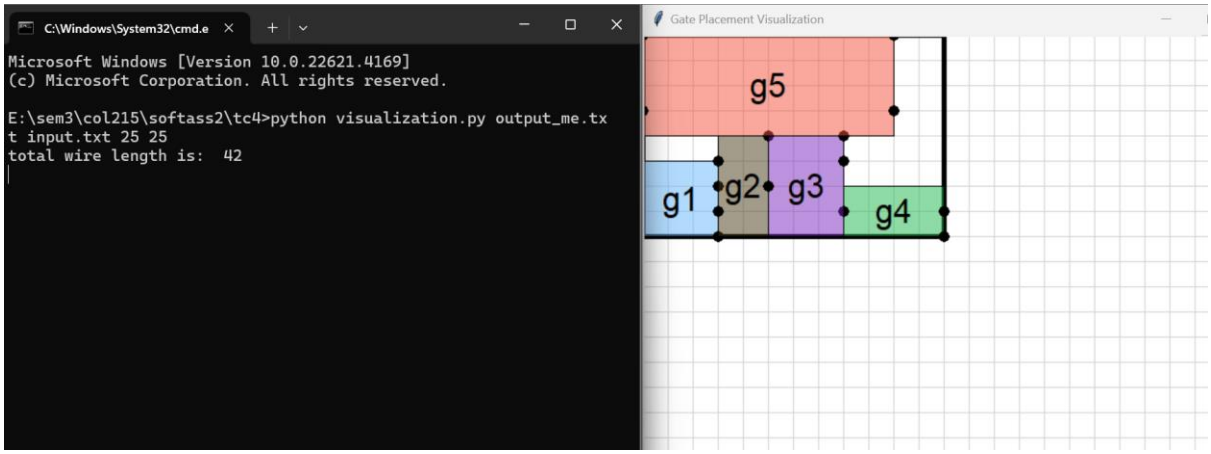
Sample Test Case 2:



Sample test Case 3:



Sample Test Case 4:



Intuition: The intuition behind right connection sorting is to reduce the redundant wiring so that there is no wastage in wires from first going left and then upward and then right. Placing gates with most right connection also ensures that maximum number of gates with which it is connected are placed near this gate. We are also placing gates in a way such that it first checks the right edge of so formed bounding box from bottom to top and then checks the top edge of so formed bounding box from left to right which ensures that interconnected gates are placed as close as possible to minimize the wire length.

Analysis:

Even though its complexity is a bit high, it provides better and efficient solution for minimizing the wire length. So, we will use this algorithm when number of gates are less than 700. We are placing the gates that are connected to each other as close as possible which ultimately minimizes the wire length used to connect the gate pins. This algorithm would take huge amount of time if we use this for more than 700 gates, but it will provide better result than the second algorithm that we would use when the number of gates is greater than 700.

Test Case	Wire length calculated (our calculator)	Wire length (Visualization)	Wire Length (Expected)
Sample test Case 1	48	59	51
Sample test Case 2	42	42	47
Sample test Case 3	134	144	<200
Sample test Case 4	42	42	49

ALGORITHM 2:

Step 1: First, we form the clusters of interconnected gates. By interconnected gates we mean that we can explore every gate of that cluster using DFS (depth first search) and starting from any gate.

Step2: Then we applied our different algorithms on each such cluster and finally placed all such clusters side by side.

Step3: Sort the rectangles in order of decreasing height and take a bin of width ‘w’.

Step4: Place all rectangles of width>w/2 one upon another like a stack and let ho be the height of the stack.

Step5: By taking ho as a shelf place further rectangles on this until width of bin is reached.

Step6: 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).

Step7: 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.

Step8: Update your hl and hr and again repeat until all rectangles are placed.

Time Complexity Analysis:

Step1: For cluster formation:

We are applying DFS on List of Gates to form clusters. In this we are starting from a gate1 and then going to every gate connected to gate1 and we mark the gate1 as visited and then applying recursion on every connected gate. So, the worst-case scenario will be when all the gates are interconnected (only one cluster) so in this case we will be visiting every gate n times (n=no. of gates) and there are n gates, so complexity will be O(n^2)

Step2: 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).

Step3a: 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 3b: 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*C + nt^2)

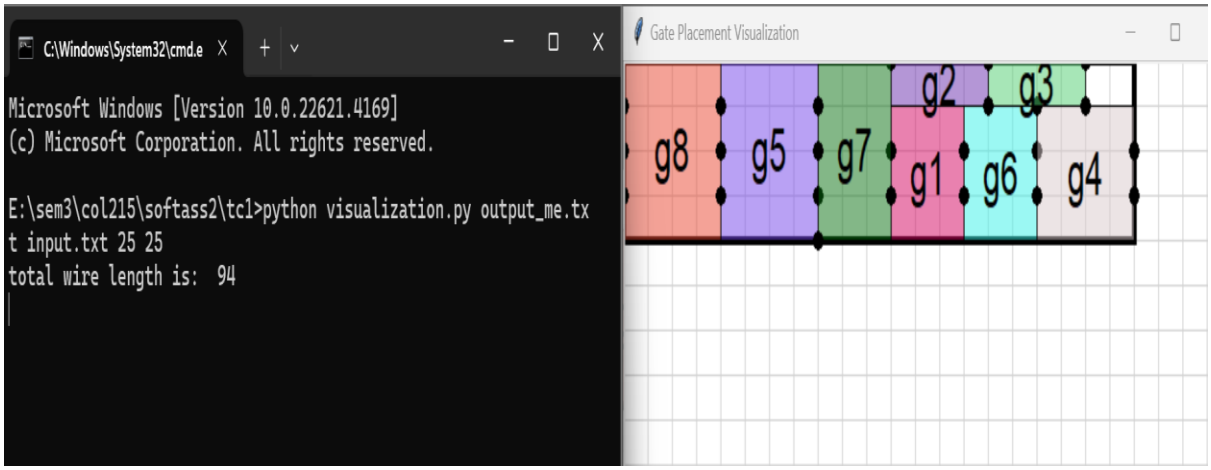
  C is number of clusters formed

  nt^2 is for cluster formation.

  Where nt is the Total number of gates.

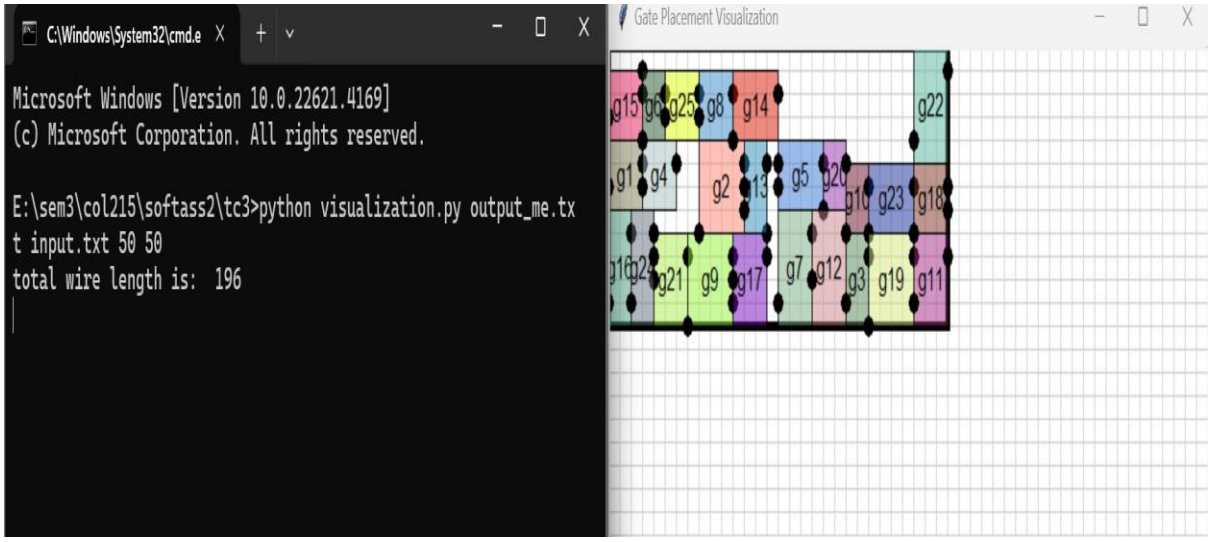
Snapshots of the visualization:

Sample Test Case 1:

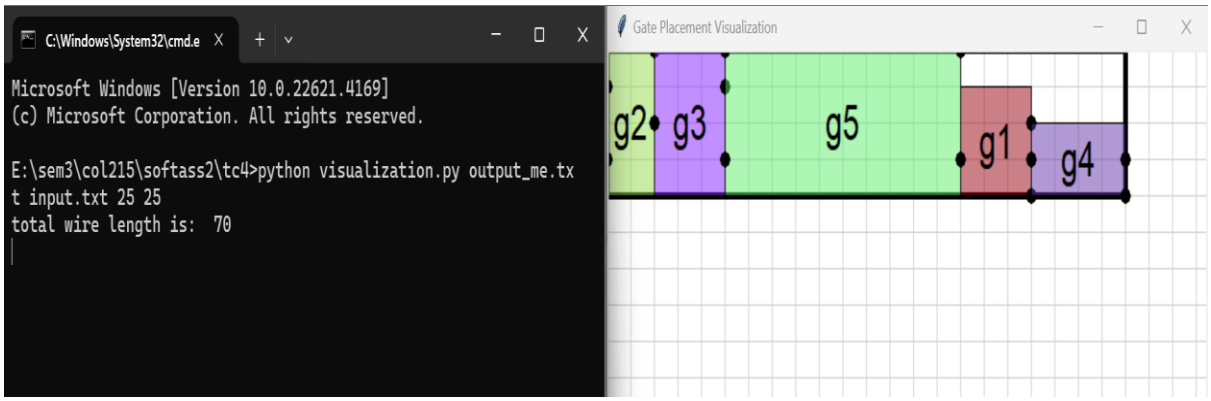


Sample Test Case 2:

Sample test Case 3:



Sample Test Case 4:



Intuition:

The intuition behind using this algorithm for large number of gates is that there will be more number of clusters so, there will relatively be lesser number of gates in a single cluster this algorithm would pack them compactly which will eventually reduce the wire length.

Analysis:

This algorithm is giving better results in large number of gates but it is likely to give less efficient result in smaller number of gates because we are not considering the number of connections, just placing the gates compactly of a cluster. It is time efficient and would run in less than 3 minutes even when the number of gates is thousand.

Test Case	Wire length calculated	Wire length (Visualization)	Wire Length (Expected)
Sample test Case 1	80	94	51
Sample test Case 2	45	45	47
Sample test Case 3	178	196	<200
Sample test Case 4	70	70	49

Algorithm of our wire length calculator:

We are calculating wire length for each cluster and then summing them up. For one cluster we are first finding a list of useful pins (means that they have at least one wire associated with them) and then we are applying DFS on first pin to find the group of pins interconnected including first pin and mark those pins as visited and then we are increasing our pointer to second pin. If it is marked as visited it means that it is a part of some previously found group of pins so we skip this iteration. But if it is not marked as visited then it means that is a part of some new pin group so we apply DFS on this pin to find all the interconnected pins including this pin too. We continue to do this until the pointer reaches the end of the array of useful pins.

Time Complexity Analysis:

We are applying DFS on List of used pins to form pin clusters. In this we are starting from a pin1 and then going to every pin connected to pin1 and we mark the pin1 as visited and then applying recursion on every connected gate. So, the worst-case scenario will be when all the pins are interconnected (only one cluster) so in this case we will be visiting every pin p times (p=no. of pins) and there are p pins, so complexity will be O(p^2).

Custom Test Cases

Custom_input_1 has 407 gates whose calculated wire length is 2277 by our code. We chose this test case to show that algorithm 1 is time effective when the number of gates is around 400. It took around 5-10 seconds to give the output.

Custom_input_2 has 476 gates whose calculated wire length is 193293 by our code. We chose this test case to show that algorithm 1 is time effective when the number of gates is around 500. It took around 15-20 seconds to give the output.

Custom_input_3 has 663 gates whose calculated wire length is 28408 by our code. We chose this test case to show that algorithm 1 is time effective when the number of gates is around 700. It took around 20-25 seconds to give the output.

Custom_input_4 has 1000 gates whose calculated wire length is 1720885 by our code. We chose this test case to show that algorithm 2 works faster when there are gates more than 700. It took around a minute to give the output. It was to test the running time of our algorithm 2.

But if we use algorithm 1 for Custom_input_4, it took around 10 minutes but provided a better solution. Calculated wire length is 1634794 by this algorithm.

Custom_input_5 has 1000 gates whose calculated wire length is 15723 by our code. We chose this test case to show that algorithm 2 works faster when there are gates more than 700. It took around 10 seconds to give the output.

But if we use algorithm 1 for Custom_input_5, calculated wire length is 17513 by algorithm 1, it took about 2 minutes.

Conclusion:

Our main algorithm is algorithm1 which is more accurate but takes more time, while algorithm 2 is not that accurate but runs very fast as compared to algorithm 1.

In comparatively small test cases where number of gates are less than 700, algorithm 1 gives better results also it is not that slow that is gives output around 2 to 3 minutes.

In large test cases around 1000 gates, when size of gates and number of connections are large then algorithm 1 gives better results but takes more time. While algorithm 2 gives faster and better results if the number of connections is less.