COL 215 Assignment 2

Design Decisions:

Class Definitions and Circuit Packing Strategy:

Gate Class:
Each gate has an ID, width, height, and bottom left coordinates (x, y) to track its position.

Circuit Class:
This class is responsible for managing the placement of gates:

check_fit(): Checks if a gate can fit horizontally within a specified level.

place_gate(): Places a gate in the level at the calculated coordinates.

add_new_level(): Adds a new level when no existing level can accommodate the gate.

pack_gate(): Tries to pack the gate into existing levels or creates a new level if necessary.

total_height(): Returns the total height used by all levels in the circuit.

ffdh() Function:
Implements the First-Fit Decreasing Height algorithm by: Sorting gates in descending order of height and Iteratively packing gates into the circuit using `pack_gate()`.

`print_packing()` Function:
Writes the packing results to an output file, including the bounding box dimensions and gate placements.

Time Complexity Analysis:

check_fit(): O(m) where `m` is the number of gates in the specified level. This function checks if a gate can fit in a given level by summing the widths of gates already in that level. In the worst case, it will iterate through all gates in the level.

place_gate(): O(m) where `m` is the number of gates in the specified level. This function calculates the `y` coordinate for the gate by summing the heights of gates in all previous levels. It then appends the gate to the level. The time complexity is determined by iterating through previous levels to calculate the total height.

`add_new_level`**():** `O(n)`, where `n` is the number of levels in the circuit. This method creates a new level and calculates the `y` coordinate of the new gate by summing the heights of all existing levels. In the worst case, this requires iterating through all levels.

pack_gate(): • `O(n * m)`, where `n` is the number of levels and `m` is the number of gates in each level. This function tries to place a gate in each level. If it doesn't fit in any existing level, it calls `add_new_level()`. In the worst case, this function may check all levels before adding a new one. Thus, for each gate, it may take `O(n * m)` time.

total_height(): • `O(n * m)`, where `n` is the number of levels and `m` is the maximum number of gates in a level. This method computes the total height of the circuit by iterating through each level and calculating the maximum height for each level. Since it checks all gates in each level, the complexity is `O(n * m)`.

ffdh(): • `O(n log n + n^2)`, where `n` is the number of gates. The sorting step has a time complexity of `O(n log n)`. Each gate is processed using `pack_gate()`, which has a time complexity of `O(n^2)` in the worst case (since each gate may be compared against every other gate in terms of placement). Combining these gives `O(n log n + n^2)`.

`print_packing()`: • `O(n * m)`, where `n` is the number of levels and `m` is the number of gates in each level. This function iterates through each level and each gate to write the placements to the file. The complexity depends on the total number of gates.

Overall Complexity Analysis:

- The most time-consuming function in the overall process is `ffdh()` due to the sorting (`O(n log n)`) and the potential quadratic complexity (`O(n^2)`) during packing. This makes the overall time complexity for the algorithm `O(n log n + n^2)`, which is effectively `O(n^2)` for large `n` values.

Time complexity= O(n**2)

Space Complexity Analysis:

- The space complexity is `O(n)` for storing the gates and `O(m)` for storing the levels, which in the worst case is `O(n)`. Thus, the overall space complexity is `O(n)`.

Output for self-generated test cases:

Test Case 1:

Input File:

G1 1 1

G2 1 1

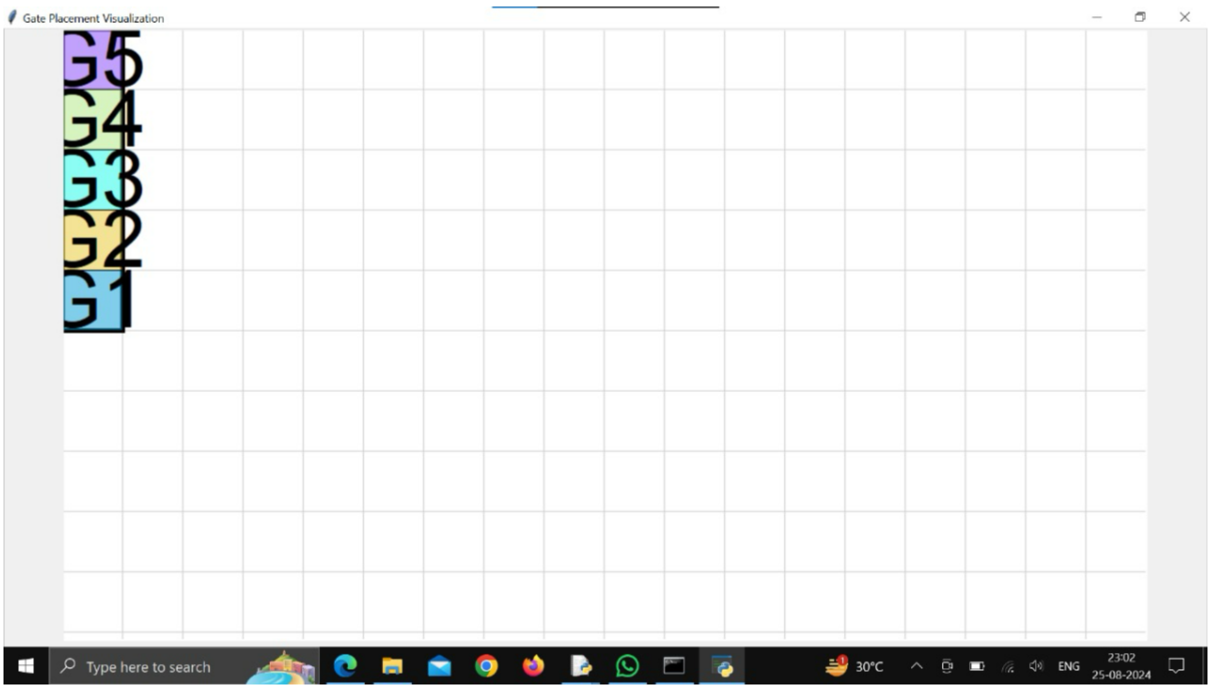
G3 1 1

G4 1 1

G5 1 1

Reasoning: This test case uses identical gates to evaluate how the algorithm handles uniform gate sizes and checks if it places them efficiently without waste.

Output:



Test Case 2:

Input:

G1 10 10

G2 9 9

G3 8 8

G4 7 7

G5 6 6

G6 5 5

G7 4 4

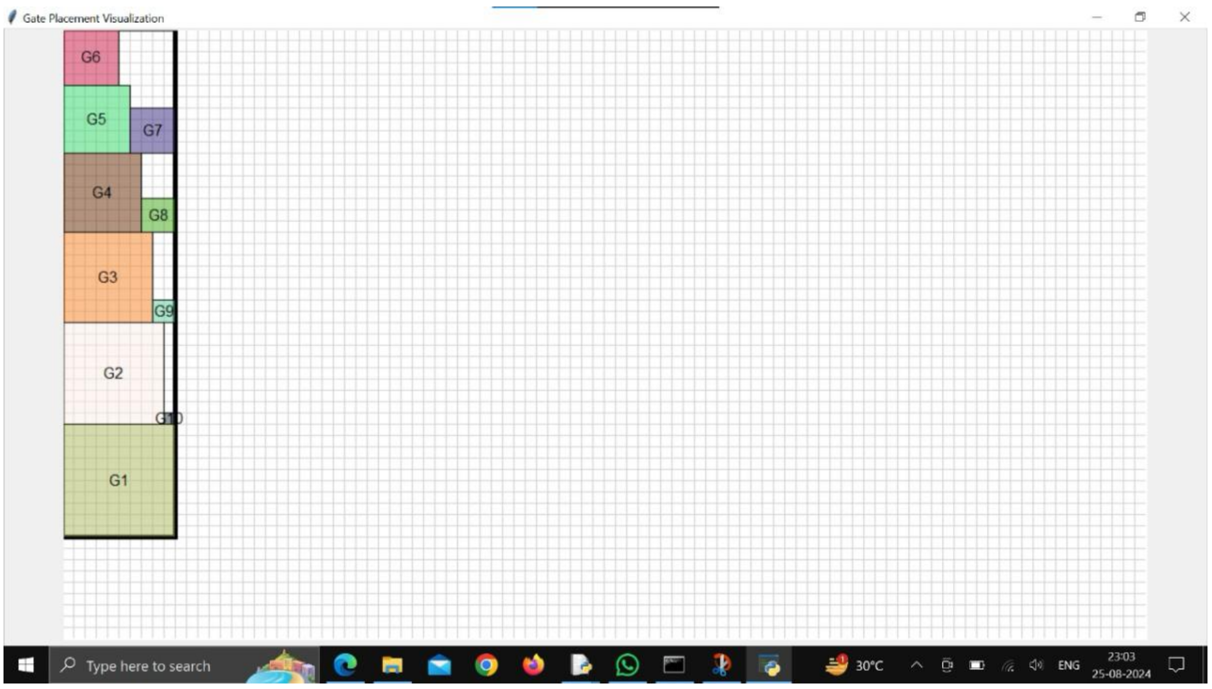
G8 3 3

G9 2 2

G10 1 1

Reasoning: This case tests the algorithm's ability to pack gates of progressively decreasing sizes, which helps in evaluating how well the algorithm manages varying dimensions and utilizes space.

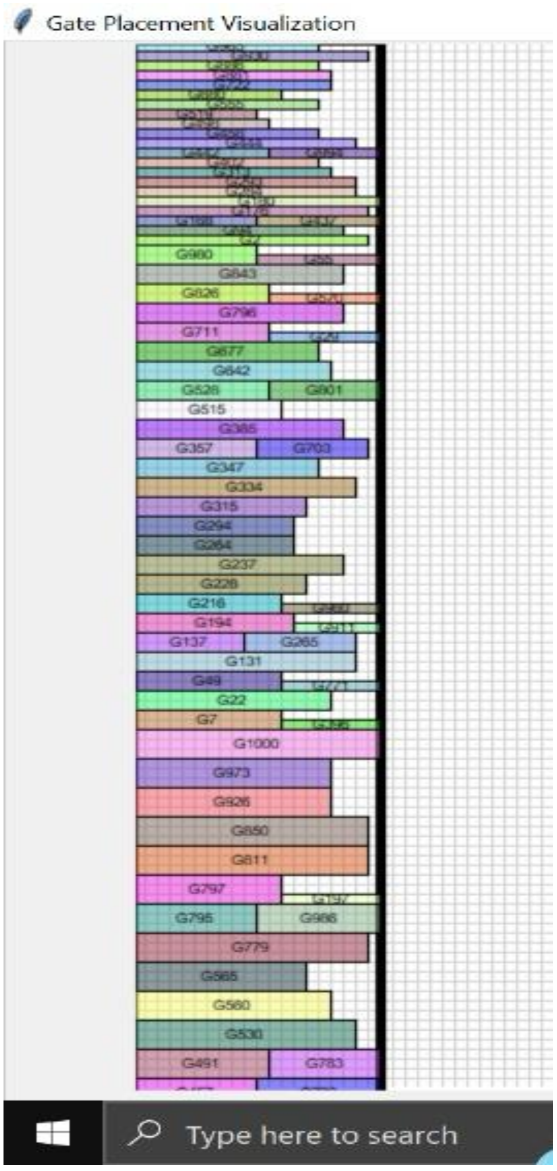
Output:



Test Case 3: We tested for a large input size(1000 gates) using python file {file is attached in zip folder}

Reasoning: This case is example for large number of gates and checks effectiveness of code. It is effective and gives quick output.

Output:



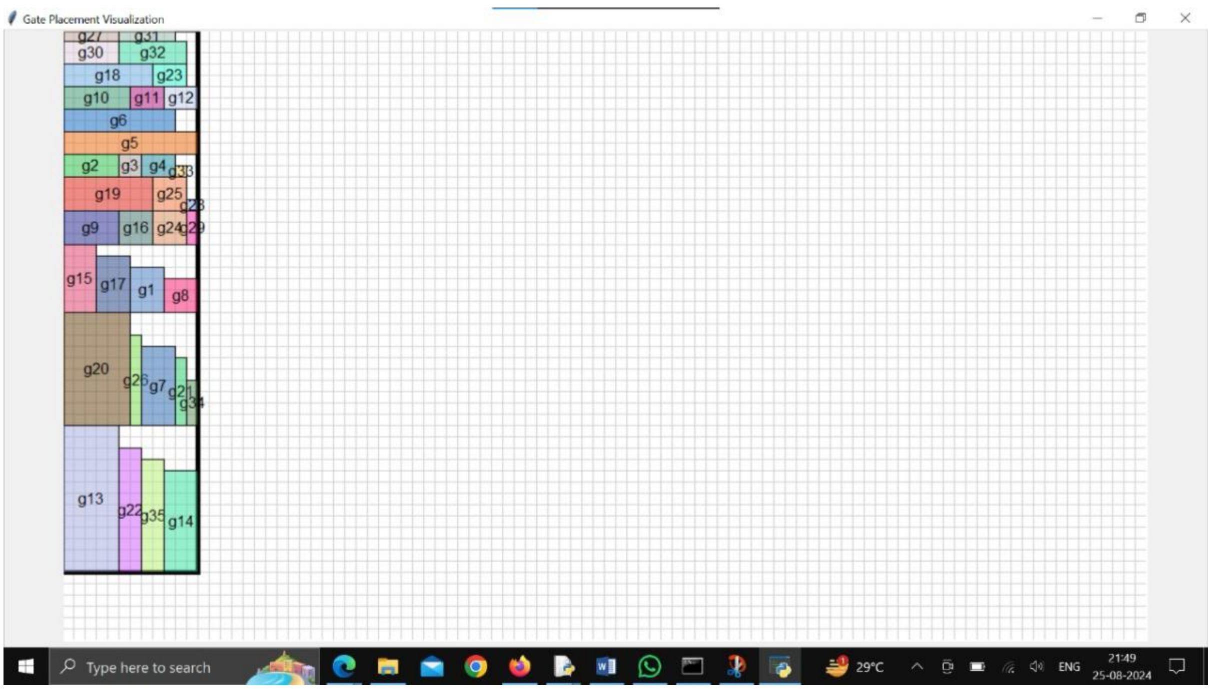
Entire screenshot cant be attached as there were 6000 grids.

Test Case 4(From moodle):

Given Test case of 35 rectangles:

Input file from moodle

Output:


Gate Placement Visualization
g27
g31
g30
g32
g18
g23
g10
g11
g12
g6
g5
g2
g3
g4
g33
g19
g25
g28
g9
g16
g24
g29
g15
g17
g1
g8
g20
g26
g7
g21
g34
g13
g22
g35
g14
Type here to search
29°C
ENG
21:49
25-08-2024