SW Assignment 1

Ishan Rehal

Tejaswa Singh Mehra

Design Decisions:

The design decisions are as follows:

1. Sorting Gates:

- **Strategy 1:** Gates are sorted by their total size (width + height + width * height), so larger gates are packed first.
- **Strategy 2:** Gates are sorted by their largest dimension, focusing on the biggest side.
- **Strategy 3:** Gates are sorted by the total area (width + height), emphasizing the overall space they take up.

2. Packing Gates:

- **Setup:** The GatePacking class starts with a list of gates and a chosen sorting strategy. It keeps track of where each gate is placed and updates potential new positions for gates.
- **Placing Gates:** The pack_gates method tries to place each gate in the best available spot. It uses the find_position method to find a good place and updates the list of possible positions and the size of the container.

3. Finding Positions:

- **Valid Spots:** find_position looks for places where a gate can fit without overlapping with others. It has two lists of possible positions to pick the best spot.
    - The first list consists of positions where adding the rectangle wouldn't change the bounding area. If such points exist, we choose the ones with the minimum x*y coordinate product.
    - The second list is used if the first list is empty. The position where adding the rectangle would cause the minimum increase in bounding area is chosen.

4. Updating Candidate Points:

- **New Spots:** After placing a gate, new potential spots are added based on the gate's corners to help find where to place the next gate.

5. Measuring Efficiency:

- **Packing Efficiency:** The calculate_packing_efficiency method calculates how well the space is used by comparing the area of the placed gates to the size of the container.

6. File Operations:

- **Input/Output:** read_input reads gate sizes from a file, and write_output saves the results to a file. This helps in managing data and recording results.

7. Visualization:

- **Optional Visualization:** The main function can show a visual representation of the packing, making it easier to see how well the gates are arranged.

8. Choosing the Best Strategy:

- **Best Strategy:** The main function tests all three sorting strategies and picks the one that packs the most efficiently.

Overall Design

The design aims to pack gates efficiently by using different strategies and updating positions as gates are placed. It keeps track of how well the space is used and allows for visualization to see the results.

Time Complexity Analysis:

The time complexity of the algorithm can be analysed based on its key operations:

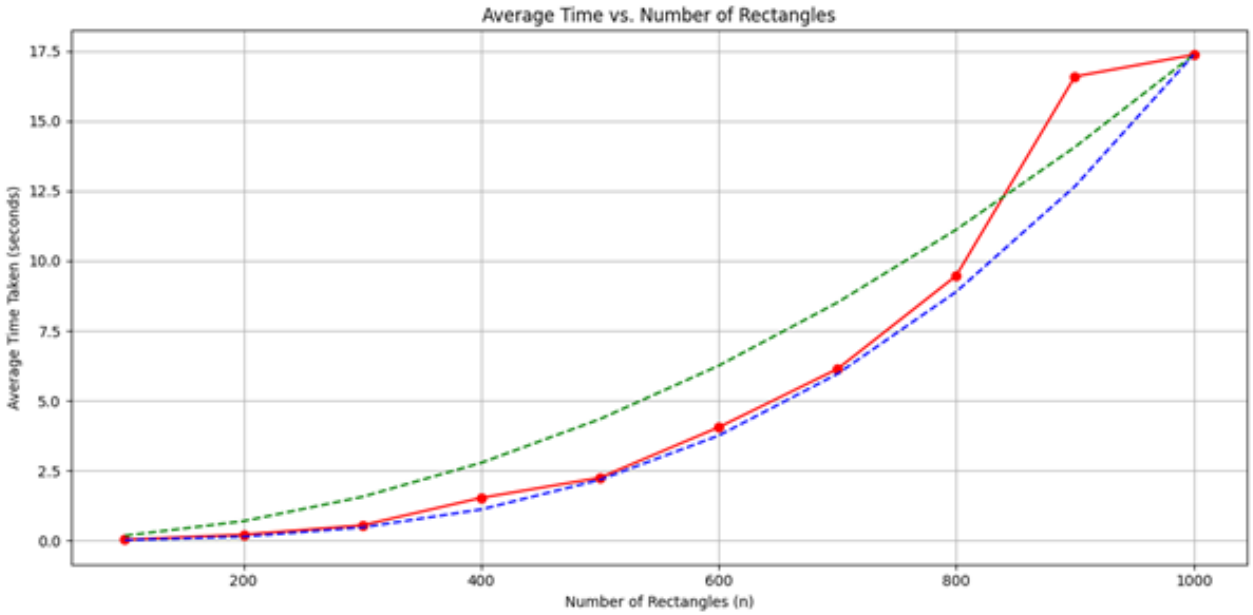
1. **Initialization (Sorting the Gates)**:
    - The gates are sorted based on the chosen strategy. If there are n gates, the sorting step takes O(n*log n) time.
2. **Packing Gates**:
    - **Outer Loop (pack_gates)**: This loop iterates over each gate, so it runs n times.
    - **Inner Loop (find_position)**: For each gate, the algorithm checks all candidate points to find a valid placement. If there are m candidate points, this step runs O(m) time. In the worst case, m can grow linearly with the number of gates, so m=O(n). Therefore, this step runs O(n) time for each gate.

- **Verification of Each Point(can_place_gate)**: For each point, the algorithm checks with all rectangles to ensure that a rectangle placed can be placed. This runs n times.
- **Updating Candidate Points**: Each placement can add at most two new candidate points, so this step is O(1).

3. **Overall Time Complexity**:
    - The dominant cost is the combination of sorting and the nested loops in the packing process.
    - The sorting step takes O(n*log n),
    - The packing process involves an outer loop running n times and an inner loop that can run up to n times with a verification that runs exactly n times each, leading to O(n^3).

Thus, the overall time complexity of the algorithm is O(n^3).

In order to verify this, we can also plot normalised graphs of n^2 and n^3 time complexities in comparison in the same graph. The curve that fits better will tell us the experimental time complexity:



The graph closely corresponds to the cubic polynomial curve, thus verifying our calculations.

1.3 Test Cases

First, we will test our code on the given test cases:

Sample test case 1:

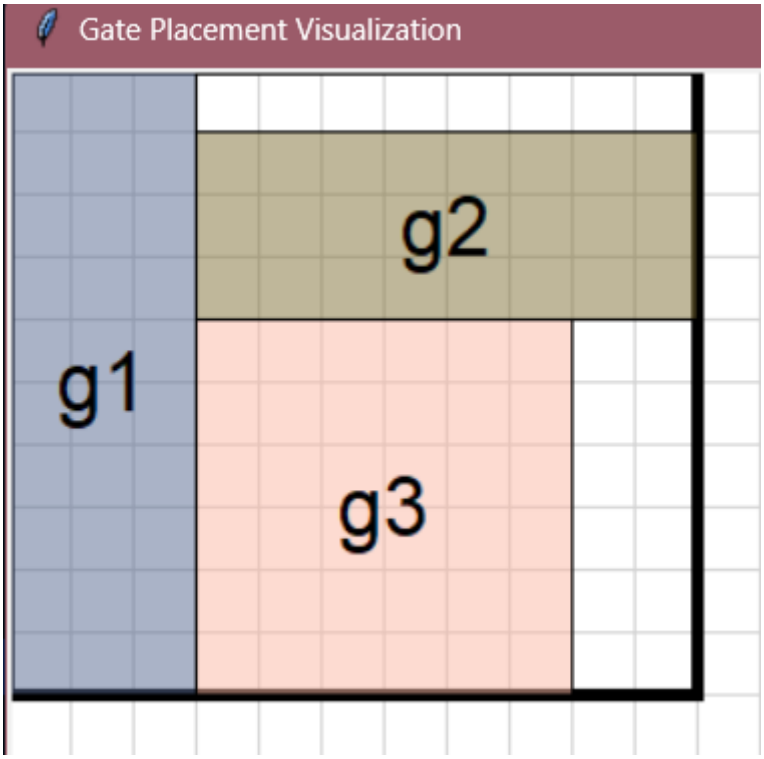
input.txt

```
1   g1 3 10
2   g2 8 3
3   g3 6 6
4
```

output.txt

```
1   bounding_box 11 10
2   g3 3 0
3   g1 0 0
4   g2 3 6
5
```

```
The packing percentage is 81.81818181818183%.
The time taken is 1.4529956000624225 seconds.
```



Sample test case 2:

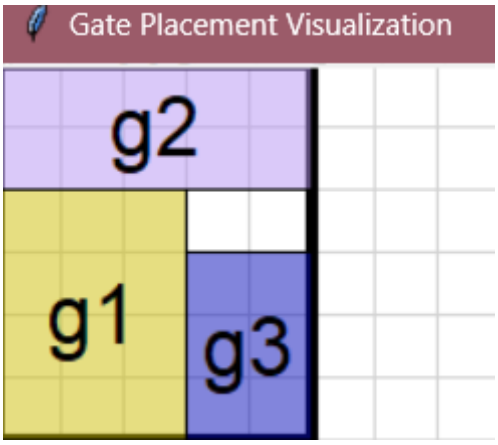
input.txt

```
1   g1 3 4
2   g2 5 2
3   g3 2 3
4
```

output.txt

```
1   bounding_box 5 6
2   g1 0 0
3   g2 0 4
4   g3 3 0
5
```

```
The packing percentage is 93.33333333333333%.
The time taken is 1.3115200999891385 seconds.
```



Sample test case 3:

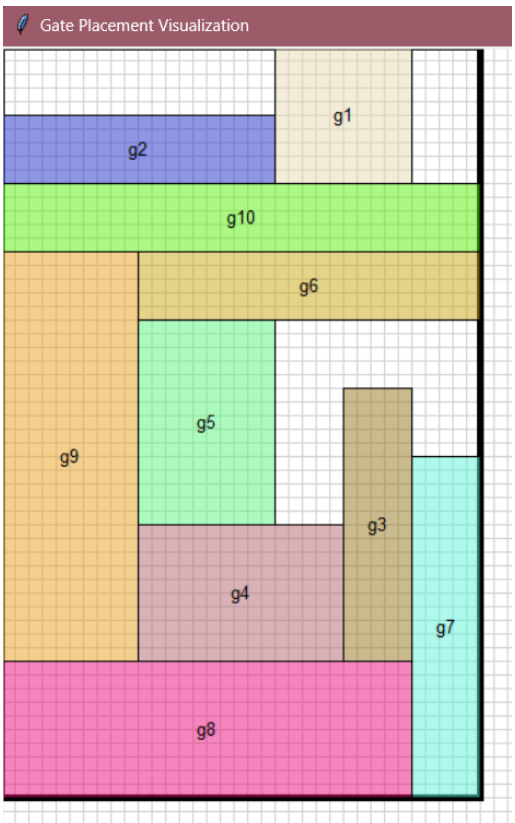
input.txt

```
g1 10 10
g2 20 5
g3 5 20
g4 15 10
g5 10 15
g6 25 5
g7 5 25
g8 30 10
g9 10 30
g10 35 5
```

output.txt

```
bounding_box 35 55
g8 0 0
g9 0 10
g4 10 10
g5 10 20
g2 0 45
g3 25 10
g10 0 40
g6 10 35
g7 30 0
g1 20 45
```

```
The packing percentage is 84.4155844155844%.
The time taken is 1.0751034999266267 seconds.
```



Sample test case 4:

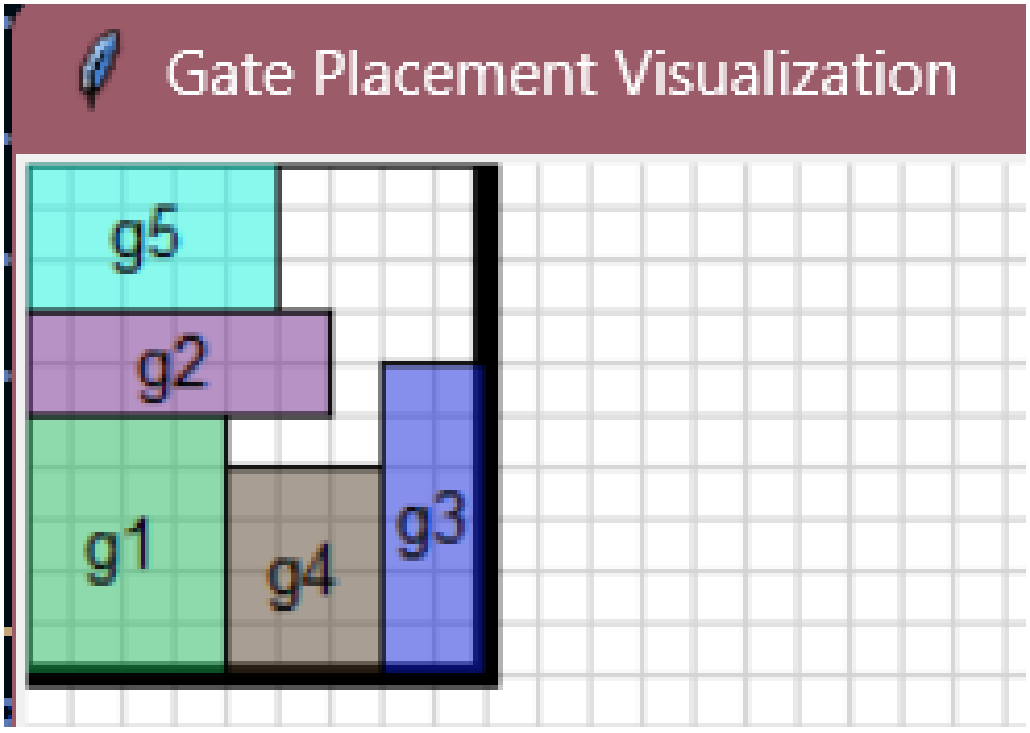
input.txt

```
1    g1 4 5
2    g2 6 2
3    g3 2 6
4    g4 3 4
5    g5 5 3
6
```

output.txt

```
1    bounding_box 9 10
2    g1 0 0
3    g2 0 5
4    g3 7 0
5    g5 0 7
6    g4 4 0
7
```

```
The packing percentage is 78.88888888888889%.
The time taken is 1.6573432999430224 seconds.
```



Sample test case 5:

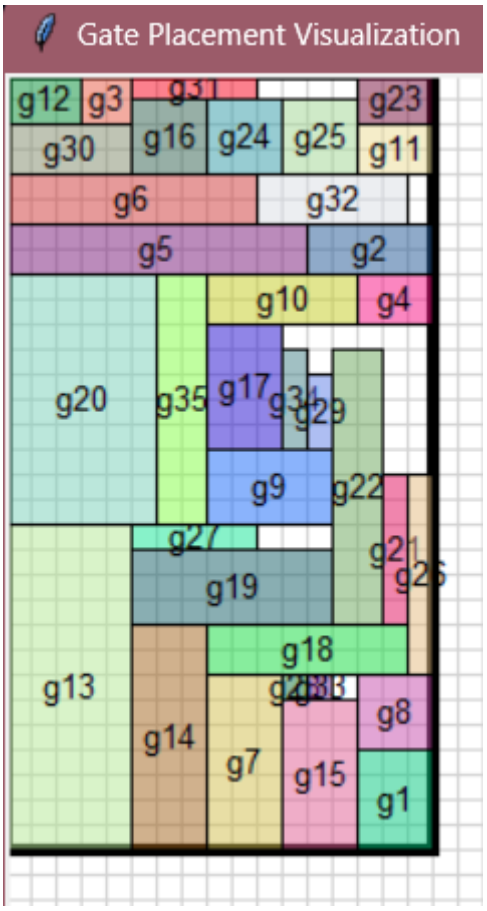
input.txt

```
g1 3 4
g2 5 2
g3 2 2
g4 3 2
g5 12 2
g6 10 2
g7 3 7
g8 3 3
g9 5 3
g10 6 2
g11 3 2
g12 3 2
g13 5 13
g14 3 9
g15 3 6
g16 3 3
g17 3 5
g18 8 2
g19 8 3
g20 6 10
g21 1 6
g22 2 11
g23 3 2
g24 3 3
g25 3 3
g26 1 8
g27 5 1
g28 1 1
g29 1 3
g30 5 2
g31 5 1
g32 6 2
g33 1 1
g34 1 4
g35 2 10
```

output.txt

```
bounding_box 17 31
g13 0 0
g20 0 13
g14 5 0
g5 0 23
g19 5 9
g22 13 9
g6 0 25
g35 6 13
g7 8 0
g15 11 0
g18 8 7
g9 8 13
g17 8 16
g10 8 21
g32 10 25
g30 0 27
g16 5 27
g24 8 27
g25 11 27
g21 15 9
g27 5 12
g34 11 16
g29 12 16
g28 11 6
g33 12 6
g1 14 0
g2 12 23
g26 16 7
g8 14 4
g4 14 21
g11 14 27
g12 0 29
g23 14 29
g31 5 30
g3 3 29
```

```
The packing percentage is 94.87666034155598%.
The time taken is 0.9313460000557825 seconds.
```



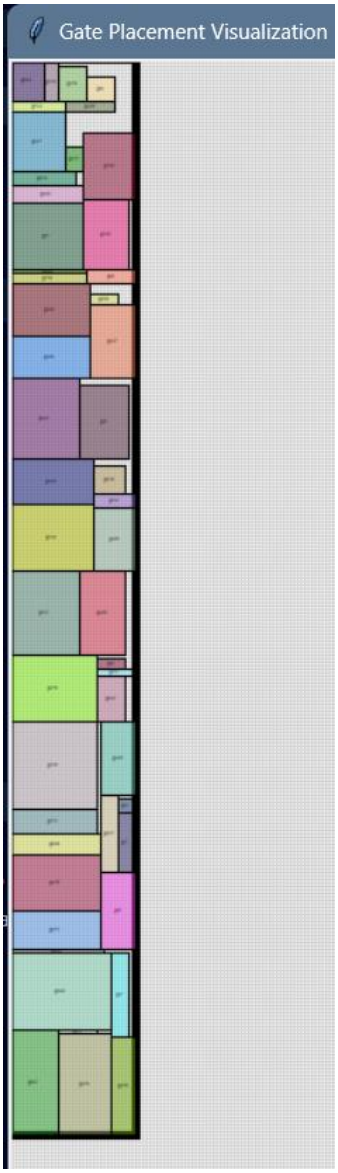
Now we will depict the performance of our code on our self-made test cases

PS: For the manual test cases, input and output file has been provided along with the report

Manual Test Case 1:

Input: Width and height of gates are Random integers from 1-30

No' of gates = 50




```
The packing percentage is 93.0691277596815%.
The time taken is 0.019701199955306947 seconds.
```

Manual Test Case 2:

Input: Width and height of gates are Random integers from 1-30

No’ of gates = 30


```
The packing percentage is 91.13876319758673%.
The time taken is 0.0020961000118404627 seconds.
```

Manual test case 3:

Input: Width and height of gates are all equal to 50

No’ of gates = 50

```
The packing percentage is 100.0%.
The time taken is 0.0179954000050202 seconds.
```

Manual Test Case 4:

Input: Width and height of gates are Random integers from 1-100

No’ of gates = 500

```
The packing percentage is 92.48090602728803%.
The time taken is 5.1344732999568805 seconds.
```

Manual Test Case 5:

Input: Width and height of gates are Random integers from 1-100

No’ of gates = 1000