Task 1:
Formulate the puzzle as a Constraint Satisfaction Problem (CSP) by identifying the variables, domains, and constraints:

For size 4:

Variables:
c1,1, c1,2, c1,3, c1,4, c2,1, c2,2, c2,3, c2,4, c3,1, c3,2, c3,3, c3,4, c4,1, c4,2, c4,3, c4,42.

Domains:

Domain for each variable: {1, 2, 3, 4}.

Constraints:
Row and Column Constraints:
Each digit must appear exactly once in each row and column. Thus, for every row and column, the values assigned to the variables must be distinct.
Row Constraints:
$c1,1 \neq c1,2 \neq c1,3 \neq c1,4$
$c2,1 \neq c2,2 \neq c2,3 \neq c2,4$
$c3,1 \neq c3,2 \neq c3,3 \neq c3,4$
$c4,1 \neq c4,2 \neq c4,3 \neq c4,4$
Column Constraints:
$c1,1 \neq c2,1 \neq c3,1 \neq c4,1$
$c1,2 \neq c2,2 \neq c3,2 \neq c4,2$
$c1,3 \neq c2,3 \neq c3,3 \neq c4,3$
$c1,4 \neq c2,4 \neq c3,4 \neq c4,4$

Group Constraints:
Group 1: {(c1,1), (c1,2), (c2,1)} | Operator: * | Result: 24
Group 2: {(c1,3), (c1,4)} | Operator: / | Result: 2
Group 3: {(c2,2), (c2,3)} | Operator: - | Result: 3
Group 4: {(c2,4), (c3,4)} | Operator: - | Result: 1
Group 5: {(c3,1), (c3,2)} | Operator: + | Result: 5
Group 6: {(c3,3), (c4,3), (c4,4)} | Operator: + | Result: 6
Group 7: {(c4,1), (c4,2)} | Operator: - | Result: 3

For **size** 6:
Variables:
c1,1, c1,2, c1,3, c1,4, c1,5, c1,6,
c2,1, c2,2, c2,3, c2,4, c2,5, c2,6,
c3,1, c3,2, c3,3, c3,4, c3,5, c3,6,
c4,1, c4,2, c4,3, c4,4, c4,5, c4,6,
c5,1, c5,2, c5,3, c5,4, c5,5, c5,6,
c6,1, c6,2, c6,3, c6,4, c6,5, c6,6.

Domains:
Domain for each variable: 1, 2, 3, 4, 5, 6}.
Constraints:

Row and Column Constraints:
Each digit must appear exactly once in each row and column. Thus, for every row and column, the values assigned to the variables must be distinct.
Row Constraints:
c1,1 ≠ c1,2 ≠ c1,3 ≠ c1,4 ≠ c1,5 ≠ c1,6
c2,1 ≠ c2,2 ≠ c2,3 ≠ c2,4 ≠ c2,5 ≠ c2,6
c3,1 ≠ c3,2 ≠ c3,3 ≠ c3,4 ≠ c3,5 ≠ c3,6
c4,1 ≠ c4,2 ≠ c4,3 ≠ c4,4 ≠ c4,5 ≠ c4,6
c5,1 ≠ c5,2 ≠ c5,3 ≠ c5,4 ≠ c5,5 ≠ c5,6
c6,1 ≠ c6,2 ≠ c6,3 ≠ c6,4 ≠ c6,5 ≠ c6,6
Column Constraints:
c1,1 ≠ c2,1 ≠ c3,1 ≠ c4,1 ≠ c5,1 ≠ c6,1
c1,2 ≠ c2,2 ≠ c3,2 ≠ c4,2 ≠ c5,2 ≠ c6,2
c1,3 ≠ c2,3 ≠ c3,3 ≠ c4,3 ≠ c5,3 ≠ c6,3
c1,4 ≠ c2,4 ≠ c3,4 ≠ c4,4 ≠ c5,4 ≠ c6,4
c1,5 ≠ c2,5 ≠ c3,5 ≠ c4,5 ≠ c5,5 ≠ c6,5
c1,6 ≠ c2,6 ≠ c3,6 ≠ c4,6 ≠ c5,6 ≠ c6,6
Group Constraints:
Group 1: {(c1,1), (c2,1)} | Operator: - | Result: 4
Group 2: {(c1,2), (c2,1)} | Operator: - | Result: 1
Group 3: {(c1,3), (c1,4)} | Operator: - | Result: 3
Group 4: {(c1,5), (c1,6)} | Operator: / | Result: 3
Group 5: {(c2,3), (c2,4)} | Operator: - | Result: 1
Group 6: {(c2,5), (c2,6), (c3,5)} | Operator: * | Result: 150
Group 7: {(c3,1), (c3,2), (c4,1)} | Operator: + | Result: 7
Group 8: {(c3,3), (c3,4)} | Operator: - | Result: 2
Group 9: {(c3,6), (c4,6)} | Operator: + | Result: 5
Group 10: {(c4,2), (c4,3)} | Operator: - | Result: 1
Group 11: {(c4,4), (c4,5)} | Operator: / | Result: 3
Group 12: {(c5,1)} | Operator: | Result: 3
Group 13: {(c5,2), (c5,3), (c6,3)} | Operator: * | Result: 60
Group 14: {(c5,4), (c6,4)} | Operator: - | Result: 4
Group 15: {(c5,5), (c5,6)} | Operator: - | Result: 1
Group 16: {(c5,6), (c6,6)} | Operator: - | Result: 3

For size 9:
Variables:
c1,1, c1,2, c1,3, c1,4, c1,5, c1,6, c1,7, c1,8, c1,9,
c2,1, c2,2, c2,3, c2,4, c2,5, c2,6, c2,7, c2,8, c2,9,
c3,1, c3,2, c3,3, c3,4, c3,5, c3,6, c3,7, c3,8, c3,9,
c4,1, c4,2, c4,3, c4,4, c4,5, c4,6, c4,7, c4,8, c4,9,
c5,1, c5,2, c5,3, c5,4, c5,5, c5,6, c5,7, c5,8, c5,9,
c6,1, c6,2, c6,3, c6,4, c6,5, c6,6, c6,7, c6,8, c6,9,
c7,1, c7,2, c7,3, c7,4, c7,5, c7,6, c7,7, c7,8, c7,9,
c8,1, c8,2, c8,3, c8,4, c8,5, c8,6, c8,7, c8,8, c8,9,
c9,1, c9,2, c9,3, c9,4, c9,5, c9,6, c9,7, c9,8, c9,9.
Domains:
Domain for each variable: {1, 2, 3, 4, 5, 6, 7, 8, 9}.
Constraints:

Row and Column Constraints:
Each digit must appear exactly once in each row and column. Thus, for every row and column, the values assigned to the variables must be distinct.
Row and Column Constraints:
Each digit must appear exactly once in each row and column. Thus, for every row and column, the values assigned to the variables must be distinct.
Row Constraints:
$c1,1 \neq c1,2 \neq c1,3 \neq c1,4 \neq c1,5 \neq c1,6 \neq c1,7 \neq c1,8 \neq c1,9$
$c2,1 \neq c2,2 \neq c2,3 \neq c2,4 \neq c2,5 \neq c2,6 \neq c2,7 \neq c2,8 \neq c2,9$
$c3,1 \neq c3,2 \neq c3,3 \neq c3,4 \neq c3,5 \neq c3,6 \neq c3,7 \neq c3,8 \neq c3,9$
$c4,1 \neq c4,2 \neq c4,3 \neq c4,4 \neq c4,5 \neq c4,6 \neq c4,7 \neq c4,8 \neq c4,9$
$c5,1 \neq c5,2 \neq c5,3 \neq c5,4 \neq c5,5 \neq c5,6 \neq c5,7 \neq c5,8 \neq c5,9$
$c6,1 \neq c6,2 \neq c6,3 \neq c6,4 \neq c6,5 \neq c6,6 \neq c6,7 \neq c6,8 \neq c6,9$
$c7,1 \neq c7,2 \neq c7,3 \neq c7,4 \neq c7,5 \neq c7,6 \neq c7,7 \neq c7,8 \neq c7,9$
$c8,1 \neq c8,2 \neq c8,3 \neq c8,4 \neq c8,5 \neq c8,6 \neq c8,7 \neq c8,8 \neq c8,9$
$c9,1 \neq c9,2 \neq c9,3 \neq c9,4 \neq c9,5 \neq c9,6 \neq c9,7 \neq c9,8 \neq c9,9$
Column Constraints:
$c1,1 \neq c2,1 \neq c3,1 \neq c4,1 \neq c5,1 \neq c6,1 \neq c7,1 \neq c8,1 \neq c9,1$
$c1,2 \neq c2,2 \neq c3,2 \neq c4,2 \neq c5,2 \neq c6,2 \neq c7,2 \neq c8,2 \neq c9,2$
$c1,3 \neq c2,3 \neq c3,3 \neq c4,3 \neq c5,3 \neq c6,3 \neq c7,3 \neq c8,3 \neq c9,3$
$c1,4 \neq c2,4 \neq c3,4 \neq c4,4 \neq c5,4 \neq c6,4 \neq c7,4 \neq c8,4 \neq c9,4$
$c1,5 \neq c2,5 \neq c3,5 \neq c4,5 \neq c5,5 \neq c6,5 \neq c7,5 \neq c8,5 \neq c9,5$
$c1,6 \neq c2,6 \neq c3,6 \neq c4,6 \neq c5,6 \neq c6,6 \neq c7,6 \neq c8,6 \neq c9,6$
$c1,7 \neq c2,7 \neq c3,7 \neq c4,7 \neq c5,7 \neq c6,7 \neq c7,7 \neq c8,7 \neq c9,7$
$c1,8 \neq c2,8 \neq c3,8 \neq c4,8 \neq c5,8 \neq c6,8 \neq c7,8 \neq c8,8 \neq c9,8$
$c1,9 \neq c2,9 \neq c3,9 \neq c4,9 \neq c5,9 \neq c6,9 \neq c7,9 \neq c8,9 \neq c9,9$

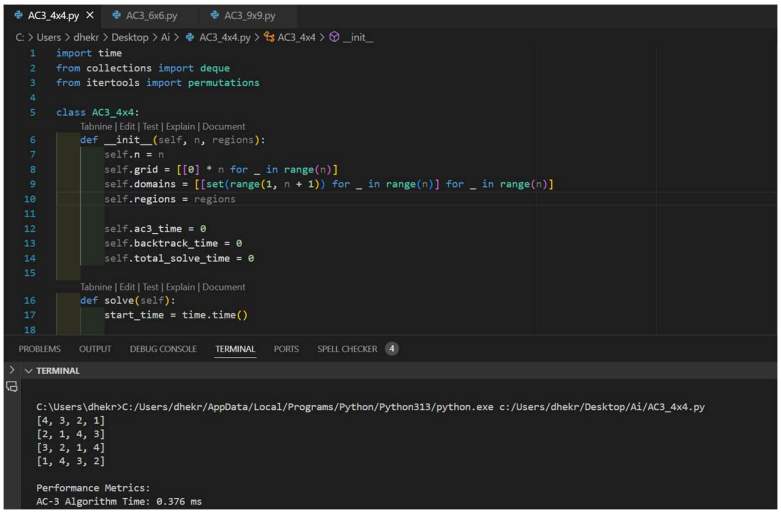
Group Constraints:
Group 1: {(c1,1), (c1,2), (c1,3), (c2,1), (c2,2), (c2,3)} | Operator: + | Result: 24
Group 2: {(c3,1), (c3,2), (c3,3), (c4,1), (c4,2), (c4,3)} | Operator: - | Result: 5
Group 3: {(c1,4), (c1,5), (c1,6), (c2,4), (c2,5), (c2,6)} | Operator: * | Result: 120
Group 4: {(c3,4), (c3,5), (c3,6), (c4,4), (c4,5), (c4,6)} | Operator: + | Result: 18
Group 5: {(c5,1), (c5,2), (c5,3), (c6,1), (c6,2), (c6,3)} | Operator: + | Result: 15
Group 6: {(c7,1), (c7,2), (c7,3), (c8,1), (c8,2), (c8,3)} | Operator: - | Result: 10
Group 7: {(c5,4), (c5,5), (c5,6), (c6,4), (c6,5), (c6,6)} | Operator: / | Result: 2
Group 8: {(c7,4), (c7,5), (c7,6), (c8,4), (c8,5), (c8,6)} | Operator: + | Result: 12
Group 9: {(c9,1), (c9,2), (c9,3)} | Operator: * | Result: 72
Group 10: {(c7,7), (c7,8), (c7,9)} | Operator: - | Result: 3
Group 11: {(c8,7), (c8,8), (c8,9)} | Operator: / | Result: 3
Group 12: {(c9,4), (c9,5), (c9,6)} | Operator: + | Result: 10
Group 13: {(c7,4), (c8,4), (c9,4)} | Operator: + | Result: 20
Group 14: {(c7,1), (c7,2)} | Operator: * | Result: 20
Group 15: {(c8,6), (c9,6)} | Operator: - | Result: 4
Group 16: {(c6,7), (c6,8), (c6,9)} | Operator: * | Result: 56

Student Name (in English): _____ **Student ID:** _____

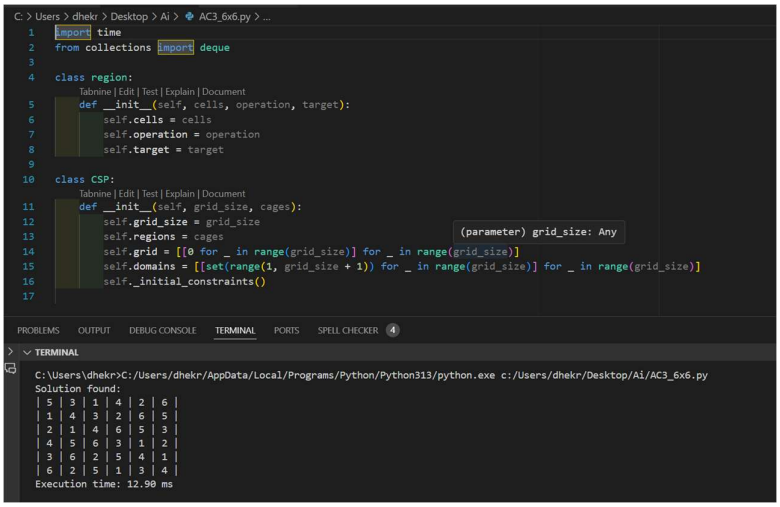
Task 2 with task 6 :

AC3:

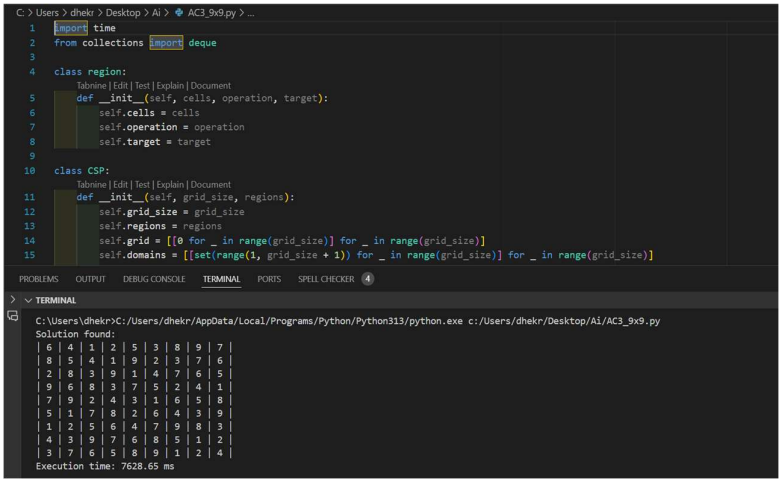
4x4



6x6



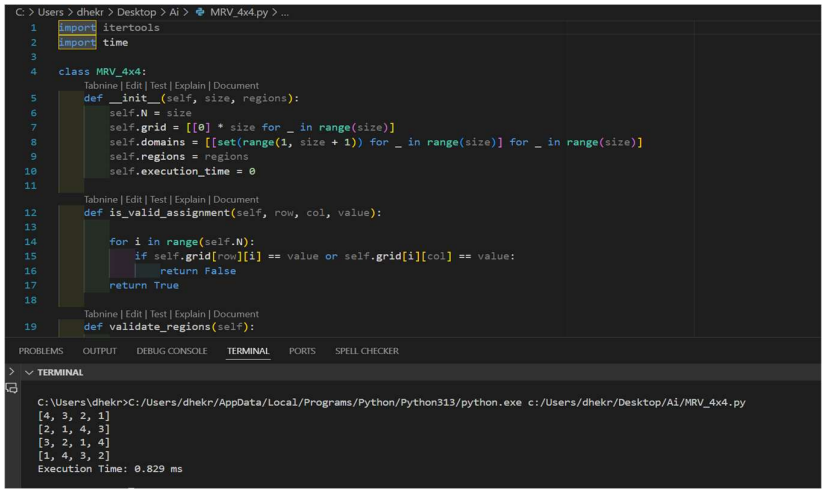
9x9



Student Name (in English): _____ **Student ID:** _____

Task 3 with task 6 : Back tracking with MRV

4x4



```
C: > Users > dhekr > Desktop > Ai > MRV_4x4.py > ...
import itertools
import time

class MRV_4x4:
    Tabnine | Edit | Test | Explain | Document
    def __init__(self, size, regions):
        self.N = size
        self.grid = [[0] * size for _ in range(size)]
        self.domains = [[set(range(1, size + 1)) for _ in range(size)] for _ in range(size)]
        self.regions = regions
        self.execution_time = 0

    Tabnine | Edit | Test | Explain | Document
    def is_valid_assignment(self, row, col, value):

        for i in range(self.N):
            if self.grid[row][i] == value or self.grid[i][col] == value:
                return False
        return True

    Tabnine | Edit | Test | Explain | Document
    def validate_regions(self):
```

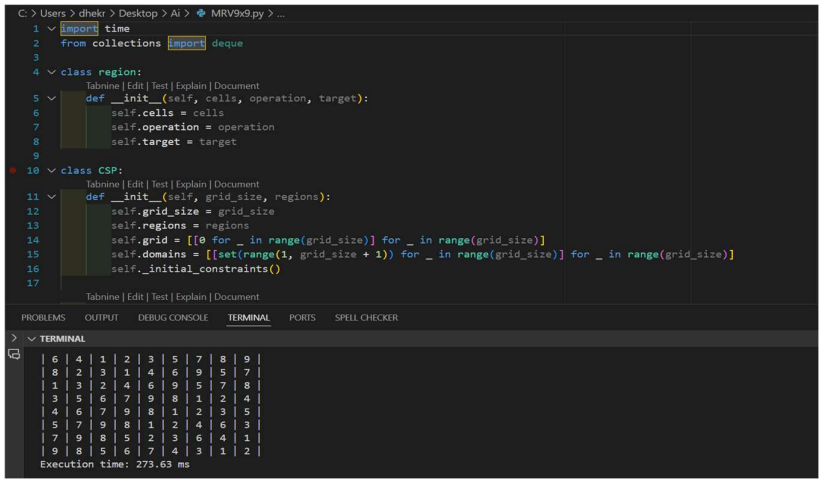
```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  SPELL CHECKER
TERMINAL

C:\Users\dhekr>C:/Users/dhekr/AppData/Local/Programs/Python/Python313/python.exe c:/Users/dhekr/Desktop/Ai/MRV_4x4.py
[4, 3, 2, 1]
[2, 1, 4, 3]
[3, 2, 1, 4]
[1, 4, 3, 2]
Execution Time: 0.829 ms
```

6x6

```
class region:
    Tabnine | Edit | Test | Explain | Document
    def __init__(self, cells, operation, target):
        self.cells = cells
        self.operation = operation
        self.target = target

class MRV_6x6:
    Tabnine | Edit | Test | Explain | Document
    def __init__(self, grid_size, regions):
        self.grid_size = grid_size
        self.regions = regions
        self.grid = [[0 for _ in range(grid_size)] for _ in range(grid_size)]
        self.domains = [[set(range(1, grid_size + 1)) for _ in range(grid_size)] for _ in range(grid_size)]
        self._initial_constraints()

    Tabnine | Edit | Test | Explain | Document
    def _initial_constraints(self):
        for i in range(self.grid_size):
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  SPELL CHECKER
TERMINAL

| 5 | 3 | 1 | 4 | 2 | 6 |
| 1 | 4 | 3 | 2 | 6 | 5 |
| 2 | 1 | 4 | 6 | 5 | 3 |
| 4 | 5 | 6 | 3 | 1 | 2 |
| 3 | 6 | 2 | 5 | 4 | 1 |
| 6 | 2 | 5 | 1 | 3 | 4 |
Execution time: 17.77 ms
```

9x9



```
C: > Users > dhekr > Desktop > Ai > MRV9x9.py > ...
import time
from collections import deque

class region:
    Tabnine | Edit | Test | Explain | Document
    def __init__(self, cells, operation, target):
        self.cells = cells
        self.operation = operation
        self.target = target

class CSP:
    Tabnine | Edit | Test | Explain | Document
    def __init__(self, grid_size, regions):
        self.grid_size = grid_size
        self.regions = regions
        self.grid = [[0 for _ in range(grid_size)] for _ in range(grid_size)]
        self.domains = [[set(range(1, grid_size + 1)) for _ in range(grid_size)] for _ in range(grid_size)]
        self._initial_constraints()

    Tabnine | Edit | Test | Explain | Document
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  SPELL CHECKER
TERMINAL

| 6 | 4 | 1 | 2 | 3 | 5 | 7 | 8 | 9 |
| 8 | 2 | 3 | 1 | 4 | 6 | 9 | 5 | 7 |
| 1 | 3 | 2 | 4 | 6 | 9 | 5 | 7 | 8 |
| 3 | 5 | 6 | 7 | 9 | 8 | 1 | 2 | 4 |
| 4 | 6 | 7 | 9 | 8 | 1 | 2 | 3 | 5 |
| 5 | 7 | 9 | 8 | 1 | 2 | 4 | 6 | 3 |
| 7 | 9 | 8 | 5 | 2 | 3 | 6 | 4 | 1 |
| 9 | 8 | 5 | 6 | 7 | 4 | 3 | 1 | 2 |
Execution time: 273.63 ms
```

Student Name (in English): _____ **Student ID:** _____

Task 4 with 6 **: Forward Checking:-**

4x4

```python
import time

class forwardchecking_4x4:
    N = 4

    Tabnine | Edit | Test | Explain | Document
    def __init__(self):
        self.grid = [[0] * self.N for _ in range(self.N)]
        self.domains = [[set(range(1, self.N + 1)) for _ in range(self.N)] for _ in range(self.N)]
        self.forward_checking_time = 0  # Variable to store forward checking time
        self.regions = self.initialize_regions()

    Tabnine | Edit | Test | Explain | Document
    def initialize_regions(self):
        return [
            {'cells': [(0, 0), (0, 1), (1, 0)], 'operation': '*', 'target': 24},
            {'cells': [(0, 2), (0, 3)], 'operation': '/', 'target': 2},
            {'cells': [(1, 1), (1, 2)], 'operation': '-', 'target': 3},
            {'cells': [(1, 3), (2, 3)], 'operation': '-', 'target': 1},
            {'cells': [(2, 0), (2, 1)], 'operation': '+', 'target': 5},
            {'cells': [(2, 2), (3, 2), (3, 3)], 'operation': '+', 'target': 6},
            {'cells': [(3, 0), (3, 1)], 'operation': '-', 'target': 3},
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  SPELL CHECKER 6

TERMINAL

C:\Users\dhekr>C:/Users/dhekr/AppData/Local/Programs/Python/Python313/python.exe c:/Users/dhekr/Desktop/Ai/forwardchecking_4x4.py
[4, 3, 2, 1]
[2, 1, 4, 3]
[3, 2, 1, 4]
[1, 4, 3, 2]
Forward Checking Time: 0.421047 ms
```

6x6

```
C: > Users > dhekr > Desktop > Ai > forwardchecking_6x6.py > ...
```

```python
import time

class region:
    Tabnine | Edit | Test | Explain | Document
    def __init__(self, cells, operation, target):
        self.cells = cells
        self.operation = operation
        self.target = target

    Tabnine | Edit | Test | Explain | Document
def kenken_solver(grid_size, regions):
    def is_valid_assignment(solution):

        for i in range(grid_size):
            row_values = solution[i]
            col_values = [solution[j][i] for j in range(grid_size)]
            if len(set(row_values)) != grid_size or len(set(col_values)) != grid_size:
                return False

        for region in regions:
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  SPELL CHECKER 6

TERMINAL

Solution found:
| 5 | 3 | 1 | 4 | 2 | 6 |
| 1 | 4 | 3 | 2 | 6 | 5 |
| 2 | 1 | 4 | 6 | 5 | 3 |
| 4 | 5 | 6 | 3 | 1 | 2 |
| 3 | 6 | 2 | 5 | 4 | 1 |
| 6 | 2 | 5 | 1 | 3 | 4 |
Execution time: 11.978626ms
```

9x9

```
C: > Users > dhekr > Desktop > Ai > forwadchecking_9x9.py > ...
```

```python
    region([(8, 2), (8, 3)], '-', 1),
    region([(8, 6), (8, 7)], '/', 2),
    region([(8, 8)], ' ', 4),
]

start_time = time.time()
solution = kenken_solver(grid_size, regions)

if solution:
    print("Solution found:")
    for row in solution:
        print("| " + " | ".join(map(str, row)) + " |")
else:
    print("No solution found.")

end_time = time.time()
print(f"Execution time: {(end_time - start_time) * 1000:.2f} ms")
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  SPELL CHECKER 6

TERMINAL

| 6 | 4 | 1 | 2 | 5 | 3 | 8 | 9 | 7 |
| 8 | 5 | 4 | 1 | 9 | 2 | 3 | 7 | 6 |
| 2 | 8 | 3 | 9 | 1 | 4 | 7 | 6 | 5 |
| 9 | 6 | 8 | 3 | 7 | 5 | 2 | 4 | 1 |
| 7 | 9 | 2 | 4 | 3 | 1 | 6 | 5 | 8 |
| 5 | 1 | 7 | 8 | 2 | 6 | 4 | 3 | 9 |
| 1 | 2 | 5 | 6 | 4 | 7 | 9 | 8 | 3 |
| 4 | 3 | 9 | 7 | 6 | 8 | 5 | 1 | 2 |
| 3 | 7 | 6 | 5 | 8 | 9 | 1 | 2 | 4 |
Execution time: 4627.345562 ms
```

Student Name (in English): _____ Student ID: _____

Task 7 :
Analysis the result
AC3 result :
Size 4:

1.164 ms	0.421 ms	0.530 ms	0.385 ms	0.379 ms	0.388 ms	0.407 ms

Best time	0.379 ms
Average time	0.525 ms
Worst time	1.164 ms

Size 6:

88.78 ms	83.10 ms	110.53 ms	70.44 ms	13.65 ms	16.80 ms	16.07 ms

Best time	13.65 ms
Average time	57.05 ms
Worst time	110.53 ms

Size 9:

7662.13 ms	7931.60 ms	6240.11 ms	6513.70 ms	7179.22 ms	6051.60 ms	6279.28 ms

Best time	6051.60 ms
Average time	6836.81 ms
Worst time	7931.60 ms

Execution Time (ms) :

Problem Size	Best Time (ms)	Average Time (ms)	Worst Time (ms)
Size 4	0.379	0.525	1.164
Size 6	13.65	57.05	110.53
Size 9	6051.60	6836.81	7931.60

Analyses:

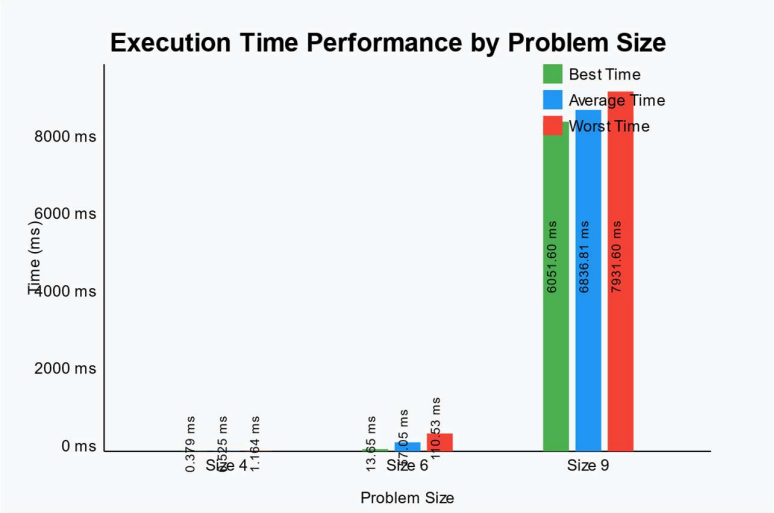
1.Problem Size and Complexity: As the problem size increases, execution time grows exponentially, especially in algorithms like AC3, where the number of constraints and variables increases significantly.

2.Performance Variance : For smaller problem sizes (e.g., Size 4), the algorithm shows consistent performance with little variance. However, for larger sizes (e.g., Size 6 and Size 9), the variance increases, showing that performance is highly dependent on the specific constraints and structure of the problem.

3.Scalability: The AC3 algorithm is efficient for smaller problems but becomes computationally expensive as the size grows, a common issue for backtracking-based algorithms in constraint satisfaction.

4.Practical Implications: AC3 is a good choice for small to medium-sized problems due to its efficiency. For larger problems, optimizations or alternative algorithms (e.g., forward checking, heuristics) might be necessary to reduce execution time.

The results shows that the AC3 algorithm's execution time is heavily influenced by the problem size. While it performs well for small inputs, its performance degrades significantly as the problem size grows. This highlights the importance of considering problem size and complexity when choosing or optimizing constraint satisfaction algorithms.



MRV:
Size 4:

1.298 ms	0.865 ms	0.844 ms	0.939 ms	0.843 ms	0.824 ms	0.818 ms

Best time	**0.818 ms**
Average time	**0.91871 ms**
Worst time	**1.298 ms**

Size 6 :

16.40 ms	15.84 ms	19.80 ms	15.87 ms	16.53 ms	16.10 ms	15.84 ms

Best time	**15.84 ms**
Average time	**16.62571ms**
Worst time	**19.80 ms**

Size 9 :

228.15 ms	252.58 ms	228.11 ms	227.34 ms	226.25 ms	225.42 ms	224.13 ms

Best time	**252.58 ms**
Average time	**230.28285ms**
Worst time	**228.15 ms**

Execution Time (ms)

Problem Size	Best Time (ms)	Average Time (ms)	Worst Time (ms)
Size 4	0.818	0.91871	1.298
Size 6	15.84	16.62571	19.80
Size 9	224.13	230.28285	252.58

Analyses:

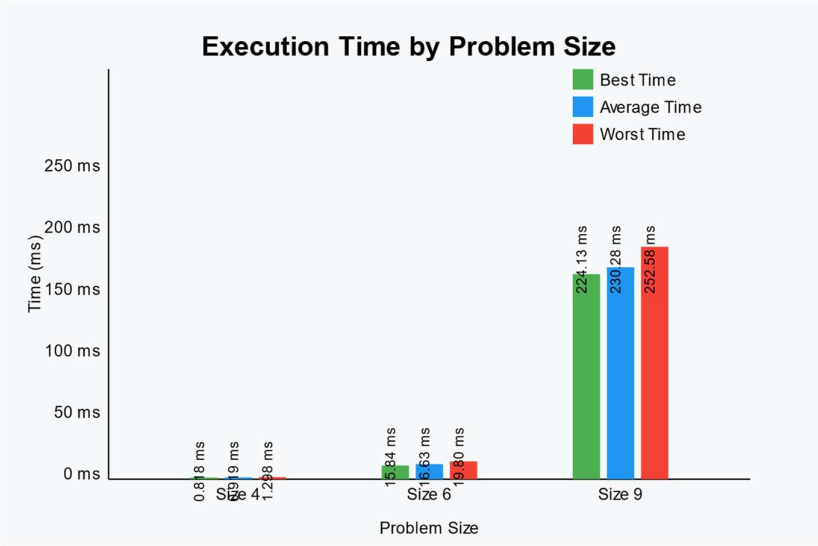
1.Problem Size and Complexity:
- As the problem size increases, the execution time grows, but the growth is not exponential because MRV heuristic reduces the search space.

2. Performance Variance:
- For smaller sizes (e.g:4), the algorithm performs with little variance between best and worst times.
- For larger sizes (e.g., Size 6 and Size 9), the variance increases slightly, but the MRV heuristic keeps the performance stable.

3. Scalability:
- The algorithm scales well with problem size, thanks to the MRV heuristic, but maybe another algorithms will work better

Comparison with AC3 Algorithm:
- The Backtracking with MRV algorithm performs significantly better than the AC3 algorithm for larger problem sizes:
- Size 9 Backtracking with MRV takes: 230 m on average, while AC3 takes: 6836 ms on average.
- This illustrates how the MRV heuristic reduces the search space and improves performance.

The results shows : Backtracking with MRV algorithm more efficient for CSP problem , even if the size is go larger. heuristic in MRV reduces the search space, keeping execution times affordable



Forward checking:

Size 4:

0.3407ms	0.365730ms	0.365257ms	0.328541ms	0.328541ms	0.327826ms	0.339031ms

Best time	0.328541ms
Average time	0.34223ms
Worst time	0.365730ms

Size 6:

9.175301ms	9.915829ms	11.949062ms	9.129763ms	9.322166ms	9.695530ms	9.954691ms

Best time	9.129763ms
Average time	9.87747ms
Worst time	11.949062ms

Size 9:

3888.18ms	3905.13ms	3869.76ms	3911.93ms	3967.02ms	3845.93ms	3965.87ms

Best time	3845.93ms
Average time	3907.68857
Worst time	3967.02ms

Execution Time (ms)

Problem Size	Best Time (ms)	Average Time (ms)	Worst Time (ms)
Size 4	0.328541	0.34223	0.365730
Size 6	9.129763	9.87747	11.949062
Size 9	3845.93	3907.68857	3967.02

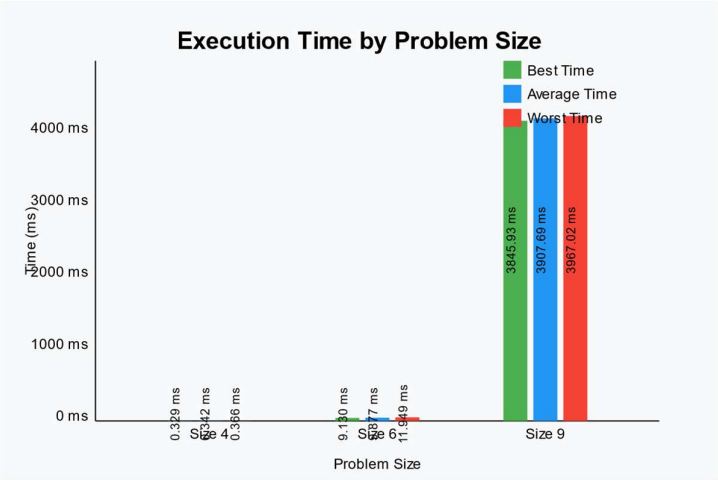
1. Problem Size and Complexity:
- Forward checking helps reduce the search space by pruning invalid branches early, but the exponential nature of the problem still leads to high execution times for larger sizes.

2.Performance Variance:
- For smaller sizes (e.g:4), the algorithm performs with little variance between best and worst times.
- The variance increases slightly for larger sizes (e.g: 6 and 9), but forward-checking keeps the performance relatively stable.

3. Scalability:
- The algorithm scales well for small and medium-sized problems but struggles with larger problems due to the increased size of the search space, leading to higher execution times.

comparison with Backtracking + MRV:
Backtracking with Forward Checking is slower than Backtracking with MRV for larger problems. In Size 9, Backtracking with Forward Checking takes :3907 ms on average, while Backtracking with MRV takes: 230 ms.
shows that MRV more effectively reduces the search space in larger problems.

The result : Forward checking helps reduce the search space as seeing in size 6 but it doesn't with large size as in size 9 , if we combine the forwarding with additional heuristics (e.g., MRV) or another algorithm we may make the execution time more efficient for larger problems size .



Conclusion):

summary of Execution Times

Algorithm	Size 4 (ms)	Size 6 (ms)	Size 9 (ms)
AC3			
- Best Time	**0.379**	**13.65**	**6051.60**
- Average Time	**0.525**	**57.05**	**6836.81**
- Worst Time	**1.164**	**110.53**	**7931.60**
Backtracking with MRV			
- Best Time	**0.818**	**15.84**	**224.13**
- Average Time	**0.91871**	**16.62571**	**230.28285**
- Worst Time	**1.298**	**19.80**	**252.58**
Backtracking with FC			
- Best Time	**0.328541**	**9.129763**	**3845.93**
- Average Time	**0.34223**	**9.87747**	**3907.68857**
- Worst Time	**0.365730**	**11.949062**	**3967.02**

After comparing : AC3 , MRV , and Forward Checking (FC) across different grid sizes (4x4, 6x6, and 9x9),
This what we found:

After analyzing
AC3, MRV, and FC across grid sizes (4x4, 6x6, 9x9):

1.Small Grids (4x4):
- FC is the fastest, with the best time of "0.328541 ms", making it ideal for small problems.

2. Medium Grids (6x6):
- "FC" still leads with a best time of "9.129763 ms", but "MRV" is competitive, showing its ability to handle growing complexity.

3. Large Grids (9x9):
- MRV dominates, with a best time of "224.13 ms", significantly outperforming FC (3845.93 ms) and AC3 (6051.60 ms).

NOTE :
- FC prunes invalid choices early, fast for small grids but less scalable.
- MRV reduces the search space by focusing on the most constrained variables, efficient in larger grids.
- AC3 enforces arc consistency, which is computationally expensive for larger grids.

- what we can do after knowing these:
- Using FC for small grids (4x4) .
- Use MRV for medium (6x6) and large grids (9x9).
- Avoid AC3 unless arc consistency is required.