Q1) For the given problem I have formulated the problem state as the following:

```
#define BLOCK_COUNT 27

struct Sum_Pair
{
	int* positions;
	int len, max, curr;
	int filled;
};

struct State
{
	char numbers[9][9];
	unsigned short domain_left[9][9];
	Sum_Pair sum_blocks[BLOCK_COUNT];
};
```

The number matrix “numbers” in struct state represents the squares in the sudoku board. The domain_left variable represents domains left for each square as a bitmap. the sum_blocks structure represents the states of the blocks which are supposed to be equal to a sum.

Each sum pair holds positions of the squares which they have in them amount of squares inside them , amount of currently filled squares, the final sum and the current sum.

Legality of a state was calculated by checking other squares in the vertical-horizontal lines from numbers and other numbers in the 3x3 which encloses the position of each variable. Additionally also the sum_pair struct was checked to see if the following rules were abided or not:

1) Current sum should be less than or equal to required sum.
2) If there are less squares filled then total squares the sum can’t be equal to the final sum.
3) If all squares in the block are filled then the current sum cant be less than expected sum (the case where it is more is covered in case one)
4) If only one square is left in the block the difference between current sum and required sum can’t be more than 9

For this problem I have chose to use backtracking with addition of Minimum remaining value heuristic and least constraining value heuristic. I have limited the domains for numbers whenever I have inserted a new value by removing those values from domains of the vertical line horizontal line and the sum cube. For the sum block I did the following:

1) If only a single vacant square is left on the block the remove all numbers other than the difference between expected sum and current sum from the domain of that vacant square.
2) If after subtracting inserted numbers value from the the difference between the expected sum and current sum got smaller than 9 remove all the domains bigger than the difference inside the domain.

For the minimum remaining value heuristic I have scanned through all 81 squares and returned the square with least amount of elements inside its domain.

For the least constraining value heuristic I have calculated the amount of domain members the positioning of that value in the specified position only for the vertical lines horizontal lines and 3x3 cubes. For least constraining value heuristic I have ignored the sum blocks.

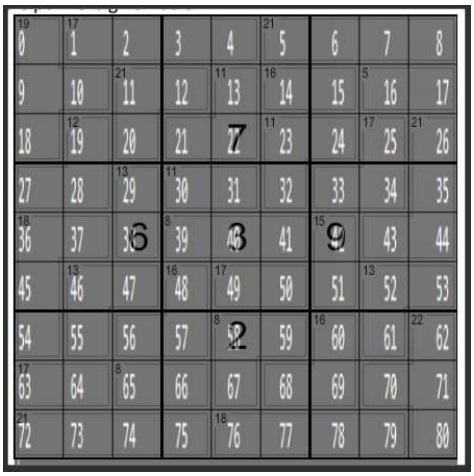
To give a specific input to the program the input can be given in the form provided inside the “test_input.txt” file which is in a c syntax form in case the computer being used isn’t using linux the input file can be named input.h and be placed in the same directory as the sudoku.cpp file and be compiled as specified below. The given solve.sh file gives the input file to the main c++ program as a header and compiles it to extract the input. The input is includes the following:

Total number of sum blocks is defined by BLOCK_COUNT variable.

sum blocks are in the following format in the blocks variable: {SUM, SQUARE COUNT, squares …}

The starting board is formatted as a 9x9 matrix of integers in the variable named board.

Squares are encoded in the following format:



To run the program you have to enter (has to be ran while in the same directory with sudoku.cpp): ./solve.sh test_input.txt

This shellscript does the following:

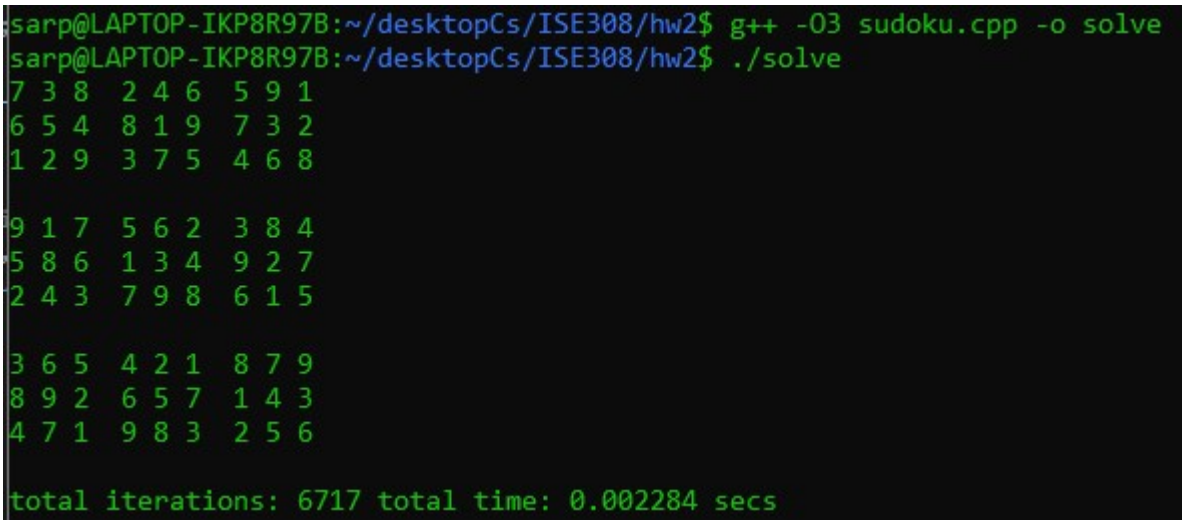
Converts the sample_input into input.h using: cp $1 input.h

compiles the code with this command: g++ -O3 sudoku.cpp -o solve

runs the code with the following command: ./solve

removes intermediate files using: rm input.h solve

On my computer with a 9750h I got the following results on Windows Subshell for Linux:



In text form the answer was this:

```
7 3 8  2 4 6  5 9 1
6 5 4  8 1 9  7 3 2
1 2 9  3 7 5  4 6 8

9 1 7  5 6 2  3 8 4
5 8 6  1 3 4  9 2 7
2 4 3  7 9 8  6 1 5

3 6 5  4 2 1  8 7 9
8 9 2  6 5 7  1 4 3
4 7 1  9 8 3  2 5 6
```

The program have tried 6717 values in 0.002 secs in order to reach this result. I have also tested this code on the itu ssh servers and it finds the correct answers but because of the difference in CLOCKS_PER_SEC of the itu ssh server the runtime have always returned 0 seconds.

When I tried this alghoritm without the lcs alghoritm it have resulted in the same answer with:

```
9234 total time: 0.003398 secs
```

Which is a downgrade of around 50% from the original both in terms of time and trials.

When I tried the alghoritm without lcs it have resulted with:

```
total iterations: 14398255 total time: 4.23347 secs
```

with neither lcs or mrv the problem have resulted with:

```
total iterations: 22044357 total time: 6.25429 secs
```

Those results also had domain modification code still as a part of them so better implementations may exist for them. Further speedups could be gained by adding additional features to the code yet the final form of the code already took 0.002 seconds to solve the problem.