Problem 1 (10 points):

a) (5 points) Briefly explain, describe and illustrate your method of representing KenKen puzzles, including the board and constraints. Which aspects are explicitly represented and which, if any, are implicitly represented?

Ans: I have implemented the kenken puzzle solver in html, javascript and used the jquery library to display the CSS (adding & removing classes, etc). In my implementation, I use cages and cells as different objects.

The cell object consists of the following:
- Row – the row in which cell is present.
- Col – the column in which cell is present.
- Index – the index of the cell, the top left cell is index 0 and traverses according to rows.
- Cage – specifies that the cell belongs to which cage
- Cell_border_top & cell_border_right – Specifies whether the top or right border is present or not according the puzzle.
- Possible choices to store the possible choices for that cell
- The cageValue ‘div’ tags to display the “goal” values for the first member in the cage.

The cage object consists of the following:
- Goal – the target value of the cage
- Index – starting index for the cells
- Operation – Mathematical operation for that cage
- Description – this is used to describe the target value in the div tag
- Members – list of cells it consists of
- cageLength – The number of cells it has

We define the input for the puzzle explicitly by manually entering each of the object details in the following format:

```javascript
var cell = cells[0];
if (cell.cage === null) {
    cell.cell_border_right=true;
    cell.cell_border_top=true;
    var cage1= new cage_object();
    cage1.description=cell;
    cage1.add(cell);
    cage1.goal=1;
    cage1.index=0;
    cage1.operation="+";
    cagelist.push(cage1);
}
```

This is for the first cell in the puzzle, we need to define this for each of the cells in the puzzle.
b) (5 points) Briefly explain why a naïve search approach such as those represented by the tree-search algorithms in Chapter 3 would be, at best, inefficient for solving a KenKen puzzle (what property of CSPs is being ignored?). Why might a backtracking algorithm provide a better solution?

Ans: Naïve search approaches explore all the possible states in the problem space without considering the constraints. KenKen has constraints on rows, columns and the cages as well, thus to improve the performance of the puzzle, backtracking would be a better choice since this will eliminate possible choices from the solution set if either one of them is satisfied. We start with the cage constraints, if we find a value for the cage or a cell, we can remove that value from the rows and columns. Naïve search approaches would not consider this and still explore those values as possible candidates which make it inefficient both in time and space. CSPs have the commutative property which means that the order does not affect the solution. By using backtracking, we use the commutative property of CSPs and also eliminate the other possible values in row or columns if we have the solution to one cell, which makes the program faster.

Problem 2 (20 points):

a) (5 points) Use your algorithm to solve the following simple KenKen puzzle:

\[
\begin{array}{cc}
1 & 2 \\
3+ & \\
\end{array}
\]

Ans:

\[
\begin{array}{cc}
1 & 2 \\
2 & 1 \\
3+ & \\
\end{array}
\]

b) (5 points) How many solutions are there to this puzzle?

Ans: Just one unique solution as shown above.

c) (5 points) Show a complete trace puzzle boards that were considered by the backtracking procedure, from the root node.

Ans: Consider the following trace diagram.
d) (5 points) Point out the steps where a branch of the search tree was abandoned because it was found to contain no possible solutions (i.e., the locations where backtracking occurred).

Ans: Backtracking occurs at the places marked as “X” in the diagram in 2c. The number 1 is repeated in the row, which violates the constraint rule.
Problem 3 (30 points):

a) (15 points) Use your algorithm to solve the following KenKen puzzle:

```
  2   5   18+  16+   3
 20*  2       7+   
  1       180*  
  3   20+   16*   
  9+   
```

Ans:

```
  2+  5+  18+  16+  3+  
  2   5   4   1   6   3 
 20* 2+  5   2   1   4   3 6 
  4+  1   6   3   7+  2  5 
 4*  180* 4+  180*  
  3+  20+  3   6   5   2 
  3   6   5   2   1   4 
  9+  6   3   2   5   4  1 
```

b) (5 points) How many solutions are there to this puzzle?

**Ans:** There are EIGHT solutions to this puzzle.
c) (10 points) What is the size of the complete search space (all possible boards) for this puzzle? How many boards are considered by backtracking?

Ans: The complete search space for this puzzle is \( n^{n^2} = 6^{36} \) starting from the initial board and without considering any of the constraints. 2758 boards are considered by backtracking.

Problem 4 (30 points):

a) (15 points) Use your algorithm to solve the following KenKen puzzle:

Answer: The solution is as follows:
b) (5 points) How many solutions are there to this puzzle?

   Ans: There is just one unique solution Possible.

c) (10 points) What is the size of the complete search space (all possible boards) for this puzzle? How many boards are considered by backtracking?

   Ans: The complete search space for this puzzle is \( n^{(n^2)} = (6^{36}) \) starting from the initial board and without considering any of the constraints. 379 boards are considered by backtracking.

Problem 5 (10 points):

The book describes two domain-independent heuristics for CSPs. One is called the "most constrained variable" (or MRV = Minimum Remaining Values) heuristic and the other is the "least constraining value" heuristic. Explain whether these heuristics can be useful for KenKen puzzles and why or why not.

   Ans: Yes, the “most constrained variable (or MRV)” is definitely useful for the KenKen puzzles. If we consider the amount of constraints that are present in the whole puzzle, such as the cage_constraints, row_constraints and column_constraints, and we also consider the number of cells each cage has. We can start putting the values to the cells with the least members, i.e. the cage which has only one cell. The goal for that cell is equal to the value (solution) of that cell. This would remove a lot of values from the search space considering the fact that there can be no repetitions in the row or column. This is the most constrained variable and by using such a heuristic, we can remove a large portion of the search tree, efficiently enhancing the performance of the program.

Extra Credit Questions (20 points):

   a) Can you think of any domain-specific heuristics that might further limit the number of puzzles considered by backtracking?

   Ans: Some of the domain specific heuristics that might further limit the number of puzzles is by removing the value that we already assigned to a cell from the other variables. This means that if we find “2” in question 3, we can remove the possible choice of 2 from other variables in that domain so that the branching factor is decreased. This kind of look-ahead check will reduce the number of puzzles. Also, as suggested in problem 5, we can sort the cages according to their member lengths, which mean that the cages with only one member are filled first, removing that value from other variables.

   b) Implement a CSP heuristic from the book in your algorithm, for instance one suggested in Problem 5 or Extra Credit Question 1, and compare the performance on several puzzles vs. non-heuristic search.

   Ans: Not implemented.