1. Stone Game
2. For this problem, it would have the recursive aspect of constantly calling the method itself in order to find the greater of the two, either the beginning pile or the end. This will result in either Alex or Lee getting the pile with the greatest possible result at that instance. Considering that we only want to return true if and only if Alex wins, we would have to end the recursive call when the amount of piles hits None.
3. The data structure I will be using is a simple array as we know the number of players in the game (Alex and Lee) each one of the index’s belonging to each one of the players. This “count” will be storing the score and keep incrementing as the game is played and the piles are collected. This will allow the current score to be stored (the sub problems) and be able to solve the final problem collecting the information needed.
4. I\_ dentify

* The problem at hand is fairly simple, the task was to do a game where a row of stones contains a different amount of stones per each, the winner of the game is the one with the most. This can be seen as a recursive method simply by the fact that you are constantly trying to find the greatest value at the given instance.

1. D\_efine

* The problem as stated before is going to be a recursive problem, there are certain aspects that we can see a pattern and one specific one in particular which will be discussed later.

1. E\_xplore

* Most possibilities are fairly simple as the problem will just be adding the amount in the pile to the current count. After doing multiple examples, I noticed a pattern. This pattern being that with the assumption that alex will begin the game all the time puts him at the advantage of constantly taking the largest amount and allowing him to win all the time (at least with the examples I made).

1. A\_ct

* The act of ideal will not apply to much besides creating the actual code and implementing the recursive and Dynamic programming aspect of it

1. L\_earn

* This program/ problem allowed me to see how Dynamic programming and recursive methods tie together.

1. Minimum Falling Path Sums
2. For this problem we will have to compare values contained within a 3 x 3 matrix (for this instance) and see how we can get to the final row containing the least possible number. Information inside of the matrix will be just integers. the recursive aspect of this problem will be the fact that it will be constantly checking the values of the top row compared the one below and adding them with the lowest possible summation, this is to be done with some restrictions.
3. The data structure I will be using is going to be another 3 x 3 matrix that will allow me to replicate the original matrix and be able to manipulate the values with the new ones. This will allow me to store the previous answer and compare them to the next best possible solution (Creating the subproblems and finally solve the main problem).
4. I\_ dentify

* The problem was simple to understand by just iterating through the matrix row by row and allowing us to use the lowest number to find the “shortest path” down the matrix.

1. D\_ efine

* The problem had some restrictions that consisted of only allowing the summation to be made from the upper level to the one below by either doing it in three paths, straight down, left or right depending on the position being compared at that instance.

1. E\_xplore

* All possibilities would depend on the type of matrix passed in and what those numbers consist of. One pattern I saw was that no matter what, the topmost row would always stay the same regardless of the integers because no operation would be done to those. Another case of integers would be the fact that negative numbers could be involved resulting in a “subtraction” to the previous amount.

1. A\_ ct

* Acting once again would consists of implementing the recursive and dynamic programming aspect of the problem into the code.

1. L\_earn

* The learning outcome allowed me to understand how matrix could be used and what operations you could use on them to solve different problems. ­­­­

1. Arithmetic Slices
2. For this problem it will consist of having an “array” passed in and seeing if each element in there the same difference between each has, therefore it would have to traverse and compare, for example, the first and second, the second and third. This will give us a difference amount between each but will see if it will always be the same, if not it is not arithmetic slices. This allows us to implement a recursive call to keep comparing the elements.
3. The data structure used in this problem will be a simple array because the problem at hand will not require much more. I will be using two arrays one containing the integers themselves and the other one containing the difference between the pairs. If all the differences are the same then this is a arithmetic sequence.
4. I\_dentify

* The problem is to simply see if all the numbers in a list have the same difference as all the rest. This is checked within any two consecutive elements.

1. D\_efine

* With this in mind this allows us to simply store the resulting differences into a separate array and see if all elements in the new one once populated are the same, if so it is correct.

1. E\_xplore

* Some of the extreme cases would be where the array is null or if it contains other data types but for the most part this problem was straight forward.

1. A\_ct

* Acting out the learning outcome would be implementing all necessities in the code itself.

1. L\_earn

* This helped me understand how to optimize even arrays to perform operations in the best time complexity possible.

1. Maximum length of pair chain
2. For this problem we will receive n amount of integers and the goal is to place them into pairs. After doing so we are to see if we can create a chain, if so what is the longest chain we can create. A chain consists of the following condition, if the structure is (a,b) (c,d) b must be less than c in order for the “link” to happen. Therefore the iteration of this problem can be done with recursion.
3. For the storing of the data, I realized that using a linked list will be the best solution to this as it will represent the idea of a “chain” as well as making the insertion of elements fairly simple.
4. I\_dentify

* The problem at hand was simple and could be done using basic comparison of lesser and greater values as well as simply implementing a linked list method to have it do the chain for us.

1. D\_efine

* Before doing the linked list implementation, we could be using an array to store the information and compare and once so we can see how we can link them up in the linked list.

1. E\_xplore

* All possible outcomes will cause issues and will require examples to be tested. I was unable to find any possible outcomes in this problem.

1. A\_ct

* The implementation of the linked list and recursion will allow us to have the most optimal outcome.

1. L\_earn

* Allowed me to get a grasp of the linked list data structure and have recursion implemented as well.

1. Perfect Square
2. The recursive aspect of this problem is to simply find the least number of perfect square numbers in order to add those and achieve the desired value (n).
3. A list would be used for this problem to be able to store the previous solutions to the subproblems and have this work be referred to for the main problem.
4. I\_dentify

* As stated before, the task is to find the perfect numbers needed to add together in order to achieve the desired value.

1. D\_efine

* If the number is from a perfect square we would see what the minimum one is and store that into the list to know what can be used for that instance.

1. E\_xplore

* The test results showed that most numbers would have the result as long as it was the even or odd value allowing most numbers to be used for this implementation.

1. A\_ct

* Once again, the act will consist of implementing the list and recursive calls.

1. L\_earn

* Allowed the understanding of how the list and recursion aspect can be used to solve this problem and others like it, this one showed the best on how Dynamic programming works.