**Problem 1:**

**I** – Identifying the problem

Alex and Lee play a game with piles of stones.  There are an even number of piles **arranged in a row**, and each pile has a positive integer number of stones piles[i].

The objective of the game is to end with the most stones.  The total number of stones is odd, so there are no ties.

Alex and Lee take turns, with Alex starting first.  Each turn, a player takes the entire pile of stones from either c beginning or the end of the row.  This continues until there are no more piles left, at which point the person with the most stones wins.

**D** – Define the cause  
We want to know if Alex wins, he has to take the best number of stones in order to win, if so we have to return True if and only if Alex wins the game.

**E** – Explore possible strategies

Step 1: Work some *small* instances by hand

I started to finding the best way of traversing a list twice finding the best path that checks if Alex wins; I started thinking that the best option was to create a 2D array by having two indexes that will make me find a right solution.

**A** – Act

Step 2: Write down what you did and

Since the problem is challenging for me, I decide to look for hints in order of completing the stone game, first we got to know that We will have to return True ONLY if Alex wins and False ONLY if Lee wins, in better chances We might see the values as coins.

Step 3: Find Patterns

We can start to see some patterns here that might help us finding the optimistic solution; firstly, checking the first and the third index of the array then if they are equal, we store the value if not we have to move to the next indexes comparing its maximum minus its minimum. (explained better below)

I started with test clarifying questions, like what if our list was empty, as we know we cannot return either a True or false, because if we do that the algorithm answer would be that one of the two guys are playing which is not the case, so my first base case was this:



Then we know that they are playing, however I came with a better idea, which was that if you have an odd number of numbers there will be no chances that Lee could win, so I improve my if function with this one:



The third step was to create a 2D arraylist which will contain 0 in its values, in the range of length of the original list:

Step 4: Check by hand and Step 5: Translate to Code

Graphically will do this if you have a length of 4 in your list:

|  |  |  |  |
| --- | --- | --- | --- |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |

This is the loop I made and improve it by searching into the web:



This is the best option so we can have a better understanding in how we can have all the possible situations in which Alex takes the first or the last so I created two nested loops in order of traversing the same list with two indexes, 2D array will be like:

|  |  |  |  |
| --- | --- | --- | --- |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | x |



Alex has to think for Lee’s worst option, once he has done with his moves, Lee will think in which coin is better, picking a greater or best value than Alex, leaving him with a lower coin options, then Alex picks again.

There are two options, either choosing the second coin or the last one, because Alex took the first one, so we want to find the greatest coin value at the second or the last coin.

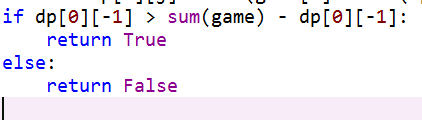
If we do not take either the second or the last one, that means that Alex took the last coin, we have to choose between the first or the third one, then we have to move and make a decision in which coin will give him the minor coin values; in order of completing what I mentioned before we have to take advantage of the 2D array. we want to store the greater coin value in position i (the first one) or the coin before in position j, reducing Alex to pick a lower coin.



There is a huge relation in creating sub-problems, because the indexes i and j are calculated twice

This is because we want to find the best result by checking if Alex wins, but not always will be possible so we have to make a right decision in what should be the answer.

After the loop completes we know that we traverse in all the j results so we can make an if statement checking if the largest sum will be taken by Alex or Lee, also with an if statement that values if the stored the possition [0][-1] (which is the possition of alex) is greater than the sum of al the list minus his possition If so we conclude that he won, else he lost.



Step 6: Run Test Cases Step & Step 7: Debug Failed Test Cases

Firstly, I was seeing that my if statement was wrong because I was using the greater than operator (>), however it was hard for me to see it because python sends me False in almost all the test cases. After rebooting my computer I saw that my base case returned me false even if the array was empty so I created an implementation which check if the array is empty and if it is even.

**L** – Look and Learn

This was a challenging problem and I saw that I need to practice more with my math solutions and also with my 2D array knowledge which I felt there were no good enough.

Source Code:

**Problem 2:**

**I** – Identifying the problem

Given a 2D array of integers, we want to find the minimum sum of a path through the list.

**D** – Define the cause **E** – Explore possible strategies

We know that the falling path starts at any element in the first row, and chooses one element from each row.  The next row's choice must be in a column that is different from the previous row's column by at most one.

Step 1: Work some *small* instances by hand Step 2: Write down what you did and Step 3: Find Patterns

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

We need to find the best strategy in order of finding any sub-problems that can be used to solve the main idea of our problem. As We know we need to start in the first row looking for the smallest number, the best way of solving this situation is by adding all the possible given scenarios in which we can store, then after moving at most one in the column or row, we have an idea of how the 2D array will be.

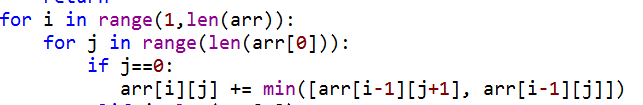
**A** – Act

Step 4: Check by hand and Step 5: Translate to Code

After creating the first if statement that checks if the array is not empty, we can start finding the solution



We will store only the minimum values from the first and second column, after looking to the all possible combinations we have to move into the next column of the second row, and store its minimum value.



In the table we will sum the minimum giving you this:

|  |  |  |
| --- | --- | --- |
| 1 | 2 | 3 |
| 5 |  |  |
|  |  |  |

After moving into the next column of the row we will add either the first, second or third value of the first column giving you different cases where we store either one or the other.

|  |  |  |
| --- | --- | --- |
| 1 | 2 | 3 |
| 5 | 6 |  |
|  |  |  |



Finally, we know that we are almost done, because we are in the last row of our 2D array, what we need to do is checking only the minimum of the second and the third value of the past column which will give you this:

|  |  |  |
| --- | --- | --- |
| 1 | 2 | 3 |
| 5 | 6 | 8 |
|  |  |  |



We can find some patterns here, after we are done with the row we have to move to the new column, this will help in order of trying to solve subproblems in order to solve the minimum sum of all cases, after the algorithm completes this will be the result

|  |  |  |
| --- | --- | --- |
| 1 | 2 | 3 |
| **5** | 6 | 8 |
| 12 | 13 | 15 |

Since we know we are done the last step to do is finding the minimum value of the last column and return it.



**L** – Look and Learn

Step 6: Run Test Cases Step & Step 7: Debug Failed Test Cases

After running several test cases I found that the position is really important at the time, we are adding the value so I was really cautious with the moves I have to do.

**Problem 4:**

**I** – Identifying the problem

Step 1: Work some *small* instances by hand Step 2: Write down what you did and

& Step 3: Find Patterns

In this problem we need to define what an arithmetic sequence of number is, which consists of at least three elements which its difference between any two consecutive elements are the same.

For example, these are arithmetic sequence:

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 2 | 3 | 4 |

With an output of 3 because we take all the elements then 1, 2, 3 and finally is 2, 3, 4

**D** – Define the cause **E** – Explore possible strategies

After knowing this we need to create a list in which we will insert if the subsequent number of a list and the past one has the same difference, adding one into our result because we are only checking 3 numbers.

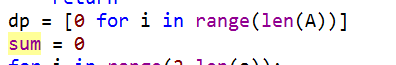
What we will do first is check if the list is empty or its equals to 1:



**A** – Act

Step 4: Check by hand and Step 5: Translate to Code

After this we need to have our counter which will give us all the arithmetic numbers we have (I personally created with 0s because we are assuming that we will use positive and negative numbers).



We need to traverse into the list, however we will start by taking the index 2 where we can check the difference between its previous and next number (like this we can check the arithmetic number, which needs at least three different numbers) this method is recursive because even if you are moving into the next index you are still checking its past number forgeting the idea of traversing all the list back again.





knowing the diffrences we might see this

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 2 | 3 | 4 |
| 0 | 0 | 1 | 0 |

The sum in this case will be equals 1 because 3 - 2 and 2 – 1 have the same difference

Then we came back with the loop and after looking into this we might see that now stores 2 because its first number was checked before:

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 2 | 3 | 4 |
| 0 | 0 | 1 | 2 |

This is because 4-3 and 3-2 have the same difference.

And after traversing the list our sum will be 3, because we were adding it into the list, finally we just return our result.

**L** – Look and Learn

Step 6: Run Test Cases Step & Step 7: Debug Failed Test Cases

Giving you that explanation we conclude that the next example is not arithmetic:

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 3 | 4 | 7 |
| 0 | 0 | 0 | 0 |

This is because even if we were using our dp list it could not sum because the only numbers with differences are 3 and 4 but you need at least 3.

**Problem 6:**

**I** – Identifying the problem

We are given n pairs of numbers. We must know that in every pair, the first number will always smaller than the second number. The tricky part of this problem is that a pair (c, d) can follow another pair (a, b) if and only if b < c. Chain of pairs can be formed in this fashion.

**D** – Define the cause **E** – Explore possible strategies

Step 1: Work some *small* instances by hand Step 2: Write down what you did and

Knowing that we need to find the longest chain which can be formed.

We already assume that in every chain the first number will always be smaller than its subsequent

|  |  |  |
| --- | --- | --- |
| 2, 3 | 1, 2 | 3, 4 |

**A** – Act

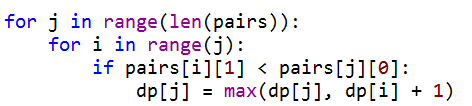
Step 3: Find Patterns, Step 4: Check by hand and Step 5: Translate to Code

Firstly, we have to sort our 2D array, because if we do not do this, our algorithm will not work good for manually traversing, then we will create our list full of ones because each chain is one.



|  |  |  |
| --- | --- | --- |
| 1, 2 | 2, 3 | 3, 4 |
| 1 | 1 | 1 |

After we are done with our list, we have to check our first number, with all the last connecting chains, we also are checking if the last number in the first column is greater than its past number from the same column, this is because we do not want to break our problem deduction; by finding this we have to move into our second column and also checking if the last numbers of each chain are smaller than the first number of each chain, if they are we have to store them into our list.



|  |  |
| --- | --- |
| 1 | 2 |
| 2 | 3 |
| 3 | 4 |

With that being said we stating seeing a path into all the subsequent columns where finally traverse into all the list, by checking if 2<1, 3<1 and 2<1, because it is not the case then we have to move into the next row column where will check if 2<2, 3,<2 and, 4<2

|  |  |
| --- | --- |
| 1 | 2 |
| 3 | 4 |
| 4 | 5 |

This is also not a connecting chain, so we have to move into our last column row where we can find that we are handling the subproblem in order of solving our main goal which is to find the longest connection chain

When we find our connection chain, we will be sorted into the list by adding just 1 to our existing result, where b is less than c.

|  |  |
| --- | --- |
| 1 | 2 |
| 2 | 3 |
| 3 | 4 |

Where 3<3 4<3 and 2<3

Then we stored the maximum number into our list



|  |  |  |
| --- | --- | --- |
| 1,2 | 2,3 | 3,4 |
| 1 | 1 | 2 |

Finally, we will only have to return our maximum number of our dp list.

**L** – Look and Learn

Step 6: Run Test Cases Step & Step 7: Debug Failed Test Cases

Giving you that explanation we conclude that the next example is not a chain connection:

|  |  |  |
| --- | --- | --- |
| 7,12 | 8,11 | 9,10 |
| 1 | 1 | 1 |

Because none of each last number is less than any of its first number.

Problem 3:

**I** – Identifying the problem

Step 1: Work some *small* instances by hand Step 2: Write down what you did and

& Step 3: Find Patterns

You are given a string, and we need to count how many palindromic substrings we have into our main string. The substrings with different start indexes or end indexes are counted as different substrings even they consist of same characters.

**D** – Define the cause **E** – Explore possible strategies

As google says “a palindrome is a word, phrase, or sequence that reads the same backward as forward, e.g., madam or nurses run”.

**A** – Act

Step 4: Check by hand and Step 5: Translate to Code

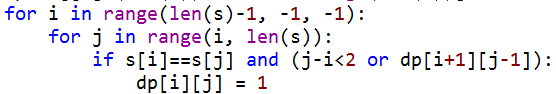
After knowing this we might find a good string that might give us more than 4 palindromes which is “aba”

Then we need to store our result and create a 2D array in order of accomplishing this task



After we did this, we might have something like this

|  |  |  |
| --- | --- | --- |
| 0 | 0 | 0 |
| 0 | 0 | 0 |
| 0 | 0 | 0 |

Then we have to check if every letter is a palindrome, that’s is why we are using a 2D list, recursively we will check each palindrome with 2 nested loops where we will start from the last row and the last column.

With that being said we will check if the letter is the same so it can be read backwards, then we will store it into our last row and column

|  |  |  |
| --- | --- | --- |
| 0 | 0 | 0 |
| 0 | 0 | 0 |
| 0 | 0 | 1 |

then will check either a is similar to b which is not, then we will have to check back again if b is similar to the first a and the last a (this is because our size is decreasing).

|  |  |  |
| --- | --- | --- |
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 0 | 0 | 1 |

After we are done with our last column we have to go and check a to itself (which it is) then check if a is equals to b and finally if a is equals to a. after this point we can start seeing a recursive way of checking all the palindromes because if the one in the middle is one, this means that we can check the palindromes on the inside.

|  |  |  |
| --- | --- | --- |
| 1 | 0 | 1 |
| 0 | 1 | 0 |
| 0 | 0 | 1 |

After that we are done traversing the list and we only have to return its result

**L** – Look and Learn

Step 6: Run Test Cases Step & Step 7: Debug Failed Test Cases

Since most of the test cases will always be a palindrome a great scenario, we will be using abc where its size is 3 however even if you have one in the middle the first and the last one are not the same

|  |  |  |
| --- | --- | --- |
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 0 | 0 | 1 |