1- Algorithm:

a- I have defined a function that checks if a string is an integer

b- The function is defined which takes a string s and an integer i, and returns a pair consisting of the number starting at the ith character of the string and the index after the last character of the number

c- The function is defined that evaluates a single parenthesized sub expression.

d- Then I have evaluated the result of the input string by calculating the result.

PROOF:

PMI-3

Base Case:

For input having only one operation, and two strings to compute the operation on, the algorithm first makes the input parenthesised and then calls for the function evalParen(s, i) to evaluate the answer inside the parenthesis. I have used recursion to successively compute the values of parameters inside nested parenthesis.

For input as “a opr b”

This evaluates to c where c = a opr b and c is float answer.

Hence its true for base case having 1 operator to solve and 0 parenthesis.

Induction Hypothesis:

For k parenthesis nested inside each other we assume that our algorithm gives the correct result.

$k<n$ and k, n belong to set N

Induction Step:

For the input having n parenthesis, we have to prove that our algorithm computes correct answer. Firstly we remove the outermost parenthesis and store the value inside a variable and then we use recursion to solve for the

value inside the 2nd parenthesis. We store the solved value as another variable and then we apply the operator to these two variables and compute the final answer.

Ans = (a opr1 (b opr2(c opr3(d opr4(.....))))) {n parenthesis and n operators}

Var1 = a

Var b= (b opr2(c opr3(d opr4(.....))))) { n-1 parenthesis and n-1 operators}

Var b is computed correctly as we assumed in our hypothesis this algorithm correctly gives answer for k <n

Ans = (var1) opr1 (var2). This is as expected.

Hence, proved.

2-

1- The function sumSequence(n) has been defined which returns the first n numbers of the sequence.

2- ALGORITHM:

a- Two numbers have been initialized as 1 and 2 in the array.

b- A variable is defined to keep the count of the ways in which a number can be added from two other numbers.

c- There are three nested loops running in parallel. The innermost loop checks if a number can expressed as a sum of two numbers in only one way.

d- The innermost loop breaks if we have count of ways >1.

e- If the count of ways after exiting the two inner loops is 1 it appends the value, else increments the value.

f- We return the required array.

PROOF:

PMI-2

Base Case:

When n = 3, we have the output as [1, 2, 3] which is as expected, because 3 can be expressed as sum of 1 and 2 only, and the list is of size 3

Induction Hypothesis:

When size of list = k, where k < n for all k, n belonging to set of natural numbers. We assume that the required list of length k can be returned by our algorithm.

Induction Step:

When the size of the list to be returned is n, we have:
Suppose the last element of the list containing n-1 elements is 'm'.
Now our algorithm has correctly appended up to n-1 elements and coming to nth element, we have two cases, either to append m+1 or increment m+1 by 1.
It is appended if and only if, it can be expressed as a sum of two numbers in only one way. Our algorithm checks this by equating all possible pairs and increasing the counting by 1 as and when we find a possible combination.
It also breaks the loop if count >1. If count is one or m+1 can be expressed as a sum of two numbers in only one way, then we append it, else increment it and repeat the same procedure for m+2 and so on carry till we find an element to be appended.
Hence, this algorithm works correctly to give the required list.
Hence, correctness is proved.

3- TIME COMPLEXITY:

We increase value of k by adding one each time, so in the worst case we iterate this m times (where m is the last element).
Then we have two loops nested each looping till length of the array.
Which is O($\boldsymbol{n^2}$).
Overall we have time complexity as **O ($\boldsymbol{mn^2}$).**

3-

1- The function minLength such that minLength(A,n) returns the length of the shortest contiguous sublist of A whose sum strictly exceeds n and returns -1 if no such sublist exists, has been written in python

2- Algorithm:

a- Two nested loops are used to compute the solution of minimum length of subarray.

b- A variable is defined to store the length of minimum array and is constantly updated if we find a shorter subarray.

c- In the outer loop we fix the first element of the array and loop over for the other elements and increment the sum. If we find a shorter subarray, we update the minimum length to length of shorter one.

PROOF:

Base Case:

For an empty list [], we get an output -1 for all n because we don't have any element in our list.

Induction Hypothesis:

Suppose the list has k elements and we want to find the shortest sublist, we assume our algorithm gives out the correct result. Where $k<n$ and k, n belong to the subset N.

Induction Step:

When the elements in the list = n, then we have to options: either to include the element in the minimum length or not include.

The inclusion or exclusion depends upon the value of the sum of elements, and therefore in our loop, we decide to take it or not depending on the relative magnitude of difference with n, compared with the other elements.

Hence, our algorithm gives the correct answer of minimum length as required.

3- TIME COMPLEXITY:

n steps for inner loop fixing the outer loop value to a particular i

T (n) = n +T (n-1)

T(n) = n + n-1 + T(n-2) ➔ n(n-1)/2 ➔ O($\boldsymbol{n^2}$).

4-

1- The function 'mergeContacts' has been created in python and it performs this operation of merging contact lists and returns a new list with the merged contacts.
2- The algorithm that I have used to merge contacts is as described below.
   a- First change a tupled list to a list of list [(), ()]->[[],[]]
   b- Extract all elements at the first position of the list of list and form a new list without duplicates.
   c- Extract all elements at the second position of the list of list and form a new list (xs) which is a list of list containing all the A[j][1] elements corresponding to the same value for A[i][0] together. This is done by:

An outer while loop that makes sure the length of A is not equal to zero and i < len(A)

d- All the sublists to be appended to xs are appended individually by initially taking an empty list (ys) and appending all the A[j][1] elements corresponding to the same value for A[i][0].
e- Within the loop, the elements which have same A[i][0] and A[j][0] after appending their second elements, the element A[j] is deleted. This makes sure no repetitions in xs takes place.
f- Then the two lists ls and xs have been merged to give the final answer in a tuple of list format.

(To do the append operation in ys first I append A[i][1] within the while loop of 'i' then I have started another loop which goes till len(A) and appends element A[j][1] if the first elements of ith index and jth index are equal.)

HELPER FUNCTION CORRECTNESS PROOF:

tupleToList: changes tupled list to list of list.

Base Case:

Elements in list = 1

The loop in the function appends each tuple in the list to form a new list of list and for base case 1 the output is a tupled list, hence true.

Induction Hypothesis:

Let elements in list = n-1; n belonging to N

The loop goes from 0 to n-2 and appends consecutively. Assuming the loop appends till element n-1.

Induction Step:

For elements in list = n, we have:

The elements till n-2 have been already appended as in induction hypothesis.

Now i = n-1 and we have to append the nth element, and this element is successfully appended in the ans list, giving us final answer containing all elements upto the last one.

Hence, proved.

ALGORITHM CORRECTNESS PROOF

PMI - 3

Base Case:

Array =[]

output in this case is as expected []

Induction Hypothesis:

Array of length n-1 assumed to give the right answer.

Induction Step:

In our algorithm firstly we created a new list by the above function which is list of list.

Then we have the answer appended with previous n-1 elements appended correctly, coming to nth index:

After appending n-1 elements, I just have one element left in the array (nth) to append. We check if the first element of that element matches

any other first elements, if it does match, I appended it to the list of that element and delete this element, then I appended this list to the final list and mapped the two lists: containing first elements, and other containing proper indexes with list of list of second elements corresponding to index in first list together in separate sublists. Then I merged these two as a list of tuples and returned this as the answer.

Hence this algorithm works correctly.

3- Time Complexity:

We can see that there are roughly two loops running inside each other so on a big side we can see its O(n^2). Let's prove it:

Fixing the outer element and looping inside we need to enumerate n cases

T (n) = n +T(n-1)

Similarly for 2nd element we have n-1 cases and so on..

=n + n-1 + T (n-2)

= n (n-1)/2 ➔ O($\boldsymbol{n^2}$).