1. Operations on very large numbers:

1- To perform the sum operation on very large numbers, the algorithm is as follows:
   a- First I convert integer format number to list by defining addLgint function in such a way that the least significant digit occupies the first position in the list and the most significant digit occupies the last digit.
   b- Then I defined LgintToInt function to convert a given integer in list format into its equivalent integer format.
   c- Then I have added the two numbers in list format and stored the answer in form of a list by defining addLgint function
   d- Then I have compared two numbers in the list format. To do this I have defined LgLesseq function.

2- For the function

power(x,n):

Base Case:

n = 0

We know that x^0 = 1 and the same is outputted by our algo, hence true.

Induction Hypothesis:

k<n power(x,k)= x^k: Assuming this is true.

Induction Step:

For n we have: power(x,n) = x*power(x,n-1) = x*x^{n-1} = x^n

Hence, proved.

Time Complexity: T(n) = 1+T(n-1) = O(n)

Space Complexity: S(n) = 1+S(n-1) = O(n)

intToLgint(x):

Base case:

n = 0 this outputs [0]

Induction Hypothesis:

Assume for k<n this outputs correct value.

Induction Step:

For n, we have : [n mod 10] :: intToLgint(n div 10)

We know that n div 10 is obviously less than n. Hence proved.

Time Complexity: T(n) = 1+t(n/10) = O(logn)

Space Complexity: S(n) = 1+S(n/10) = O(logn)

LgintToInt(xs):

Base case:

For singleton list this function outputs correctly as the element x in it.

Induction Hypothesis:

For n assume this outputs correctly.

Induction Step:

For n+1

We have: LgintToInt(xs) =(n+1)*power(10,n) :: LgintToInt(tl(xs))

Now LgintToInt(tl(xs)) is true as our assumption, therefore our hypothesis is correct and hence proved.

Time Complexity: T(n) = 1+t(n-1) = O(n)

Space Complexity: S(n) = 1+S(n-1) = O(n)

Where n is the length of the list.

addLgint(l1,l2):

Base Case:

When l1= [], l2= [] we have output as []

Induction Hypothesis:

For l1, l2 this is true

Induction Step:

For a::l1,b::l2 we have

(a + b):: addLgint(l1,l2)

Now there are two cases if a+ b is >9 or not.

For handling this I have defined another local function c which takes care of this part to carry 1 when we have a+ b is >9 else carry 0.

Hence we have an extra element to the list which is successfully added to the beginning/end.

Hence proved.

Time Complexity: T(n) = 2+t(n-1) =2n= O(n)

Space Complexity: S(n) = 1+S(n-1) = O(n)

LgLesseq:

This function first changes the integer in list format to equivalent form in integer format. The time complexity of this is O(n) and time for comparison is O(1). Hence the overall time complexity is O(n) (where n is the digit count).

Space complexity: O(n) (same as LgintToInt(xs))

3- The algorithm is implemented in SML.

2. Quarterly Performance

1- I have defined a function avg that is used to calculate the average of all the items in a given list. This gives a real output as expected.
I have also defined a function filter to filter a list according to some specific condition.
After this I have defined another function phi which is used to filter out 4 elements out of 5 in a given list.
Ex: (a,b,c,d,e) -> (a, b, c, d).
After this another function new is defined to filter out last element e out of the list and also give the value of e /100.
After this another function salary is defined to filter out last element e out of the list.
Then a complex function mult is defined which takes input two lists and gives a specified output as (100*hd(l1) + y*x) recursively called for all the elements.
After this the function qPerformance is defined which is core of the solution.
In this function I have specified the algorithm to calculate the incremented salary of the employees mentioned. For all the quarters, the algorithm specified to find the net increment is as follows:
{a(value of the performance of a particular individual) – average value of all the employees}/0.1 *(average value of all the employees).
This gives the increment percentage at a particular quarter of the year.
I have summed up the increments for all the quarters.
After finding the net increment, I have called mult function to find the net salary with increment in that year.
This gives output as the salary of all employees in the company stored in a list.
After this I have defined a function sum to calculate sum of any two numbers.
Then I have defined a function budgetRaise which is used to calculate the overall percentage raise in salary budget the start-up has to handle solely considering salary hike. This is computed using the foldl function which computes sum starting from left till the end. We find the sum of salary paid after increments (suppose a) and the salary supposed to be paid without incrementation(suppose b). Then to find the answer I have computed (a-b)/b.
This is the required answer.

2- <u>Time Complexity:</u> Time complexity of avg is O(n). Where n is size of the list.
This avg function is called 4n times in qPerformance

T(n) = 4n + T(n-1) =4n*n = $4n^2$ = O(n^2)

Space Complexity: Space complexity of avg is O(n). Where n is size of the list.

The avg function is called 4n times in qPerformance and qPerformance is called n times. Therefore stacking up 4n*n frames in total. Therefore space complexity = O(n)

3- The algorithm is successfully implemented in SML.

3. Lexicographic Permutations

1- To find the lexicographic permutations of a given list of distinct characters. I have designed an algorithm as follows.

I have used recursive loop in which I fix one of the character and permute the rest. To do this, firstly I have defined a sorting algorithm which sorts the input by the user and then starts the algorithm. To sort, I have designed an algorithm which compares two characters and sorts them accordingly.

Function insert is used to insert a character in a sequentially sorted way. Function sort is designed to insert the different elements in the same list in a sorted way.

Then I have defined a function nth which I have used to 'refer' to a specific element in the list. Ex: nth ([a, b, c], 2) here 2 is referred to 'b'

Then I have defined a function delete which deletes a specific character in the list.

Then I have defined a function append which is used to concatenate back the character which I had fixed initially (deleted). Append function here appends the specified character to every sub list in the list.

After this I have defined a function permute which is used to find all the lexicographic permutations of the given list. This recursively works by deleting first element and permuting rest ones. In this, I have defined a local variable which temporarily stores the possible permutations by fixing the first element.

I have then defined lexicographicPerm which calls the function permute to find the permutations.

Correctness:

Base Case:

When input is [a] this has just one output which is itself i.e. [a]. Hence the algorithm is true for base case.

Induction Hypothesis:

(PMI version 3)

For list length k in the list where k < n, we assume this is true.

Induction Step:

For list length n we have

lexicographicPerm(n,n,i) =

....

val z = lexicographicPerm(tl(n),tl(n),i)

now this is the recursion part and as our assumption says that the algorithm works correctly for any k<n. Hence value of z is found correctly and hence value of lexicographicPerm of n is also found correctly.

Hence proved.

2- Time Complexity:

T(n) = n^2*T(n-1) --> O(($n!)^2$) (n = number of elements in the list)

Space Complexity:

S(n) = n*S(n-1) ->O(n!) (n = number of elements in the list)

3- The algorithm has been implemented in SML.