Q1)

a) intToLgint(n) $= \begin{cases} [0] & n = 0 \\ g(n) & n \neq 0 \end{cases}$

$g(n) = \begin{cases} [] & n = 0 \\ (n \bmod 10) :: g(n \operatorname{div} 10) & n \neq 0 \end{cases}$

b) LgintToInt(l) $= \begin{cases} 1,000,000,000 & length(l) > 9 \\ g(l, 0, 1) & length \leq 9 \end{cases}$

$g(l, i, a) = \begin{cases} i + x * a & l = [x] \\ g(tl(l), i + hd(l) * a, a * 10) & l \neq [x] \end{cases}$

c) addLgint(x, y) = g(x, y, 0)

$g(x, y, i) \begin{cases} [] & i = 0, x = [], y = [] \\ i :: [] & i \neq 0, x = [], y = [] \\ (hd(y) + i) \bmod 10 :: g([], tl(y), (hd(y) + i) \operatorname{div} 10) & x = [] \\ (hd(x) + i) \bmod 10 :: g(tl(x), [], (hd(x) + i) \operatorname{div} 10) & y = [] \\ (hd(x) + hd(y) + i) \bmod 10 :: g(tl(x), tl(y), (hd(x) + hd(y) + i) \operatorname{div} 10) & \end{cases}$

d) LgLesseq (l_1, l_2) =

$$\begin{cases} \text{true} & l_1 = l_2 \\ \text{true} & \text{length}(l_1) < \text{length}(l_2) \\ \text{false} & \text{length}(l_1) > \text{length}(l_2) \\ \text{comp}(x, y) & \text{, where } x = \text{reverse}(l_1, []) \\ & \quad y = \text{reverse}(l_2, []) \end{cases}$$

$$\text{reverse}(x, l) = \begin{cases} \text{hd}(x) :: l & x = [x] \\ \text{reverse}(\text{tl}(x), \text{hd}(x) :: l) & x \neq [x] \end{cases}$$

$$\text{comp}(x :: xs, y :: ys) = \begin{cases} \text{false} & x > y \\ \text{true} & y > x \\ \text{comp}(xs, ys) & xs \neq [] \\ & ys \neq [] \end{cases}$$

Q2) for int To Lgint (l)

If n = 0 then [0]

else g(n)

Base case g(0) = []

Induction hypothesis :

Let g(k) be correct for all $k < n$

Induction step : (let n be a k digit no. :: $d_k d_{k-1} \cdots d_2 d_1$

g(n) = (n mod 10) :: g(n div 10)

↳ correct by IH

$= [d_1, d_2, d_3 \cdots d_{k-1}, d_k]$

$d_1 :: [d_2, d_3, \cdots, d_{k-1}, d_k]$

Time complexity = $\Theta(\log(n))$

Space complexity = $\Theta(\log(n))$

b) For ~~intToInt~~ LgintToInt

If length(l) > 9, i.e., number $> 10^9$, then (go to guess)

else g(l, 0, 1)

g(l, i, a)

Base case:

i + x * a when l = [x] (singleton list)

time complexity = Θ(length(l))

space complexity = Θ(length(l))

If we assume storing the input takes this space)

a variable 'a' stores the power of 10 as required for a particular digit's place

~~variable 'i' stores the value of g~~

Invariant:

i + g(l, 0, a) will always be equal to the desired output for ~~g~~ g(l, @l, 0, 1)

Let l, @l = [1,2,3] @ [1,2,6]

Let invariant be true at step where

i = 321 l = [1,2,6] a = 1000

next iteration will give

321 + g([2,6], 1000, 10000]

= 1321 + g([2,6], 0, 10000) invariant is satisfies

hence proved

c) for add Lg int $(x, y) = g(x, y, 0)$

$g(x, y, i)$:

'i' is used to store carry over at any stage

Base case :

$i :: []$ if $x = []$, $y = []$

Induction hypothesis

Let the function be correct till the k^{th} digit

Let i be the carry over during k^{th} digit

(1st digit) :: (2nd digit) :: (... :: (k^{th} digit) :: $((x_{k+1} + y_{k+1} + i) \bmod 10)) ::$

($[x_{k+2} \cdots x_n]$, $[y_{k+2} \cdots y_m]$ (($x_{k+1} + y_{k+1} + i$) div 10))

non carry over pair

carry over

Hence proved the function will also be correct for $k+1^{th}$ digit

---

time complexity = $O(\max(\text{length}(x), \text{length}(y)))$

space complexity = $O(m)$

where $m = (\text{length}(x) + \text{length}(y))$

d) for lyke sseq (l_1, l_2) =

$l_1 = l_2 \longrightarrow$ true

length $(l_1) <$ length $(l_2) \Rightarrow$

log (l_1 (numerical form)) < log (l_2 (numerical form))

$\therefore \quad l_1 < l_2 \longrightarrow$ true

and vice versa for $l_2 > l_1 \longrightarrow$ false

If length of (l_1) = length of (l_2)
then the number of digits in both l_1 and l_2 is same.

The last element of each list will have the highest priority
Therefore we first reverse the list and then check element 1 by 1, whenever we find 2 elements that are not equal,
if element of $(l_1) <$ element of $(l_2) \Rightarrow$ true
else false. (We already have checked the case for equality $l_1 = l_2$)

Time Complexity = $O(\max(\text{length}(l_1), \text{length}(l_2)))$

Space Complexity = $O(\text{length}(l_1) + \text{length}(l_2))$

Bonus:

multiplyLgint (x::xs, y::ys)

$$= \begin{cases} g(x::xs,\ y,\ 0) & \text{if } ys = 0 \\ \\ addLgint(g(x::xs, y, 0),\ 0::multiplyLgint(x::xs, ys & \end{cases}$$

$g(x, y, i) =$

$$\begin{cases} [] & i=0,\ x=[] \\ i::[] & i\neq 0,\ x=[] \\ \\ ((hd(x) * y + i) \bmod 10) :: g(xs, y, ((hd(x) * y + i) \operatorname{div} 10)) & \end{cases}$$

```
fun qPerformance(l) =
  let
        val len = length(l);
        fun q1(a,b,c,d,e) = Real.fromInt(a);
        val q1list = map q1 l;
        fun q2(a,b,c,d,e) = Real.fromInt(b);
        val q2list = map q2 l;
        fun q3(a,b,c,d,e) = Real.fromInt(c);
        val q3list = map q3 l;
        fun q4(a,b,c,d,e) = Real.fromInt(d);
        val q4list = map q4 l;
        fun sum(x:real,y:real)=x+y;
        val q1=((foldr sum 0.0 q1list))/Real.fromInt(len);
        val q2=((foldr sum 0.0 q2list))/Real.fromInt(len);
        val q3=((foldr sum 0.0 q3list))/Real.fromInt(len);
        val q4=((foldr sum 0.0 q4list))/Real.fromInt(len);
        fun profit(a)= if a>q1 then floor((a-q1)/(0.1*q1)) else 0;
        val p1= map profit q1list;
        fun profit(a)= if a>q2 then floor((a-q2)/(0.1*q2)) else 0;
        val p2= map profit q2list;
        fun profit(a)= if a>q3 then floor((a-q3)/(0.1*q3)) else 0;
        val p3= map profit q3list;
        fun profit(a)= if a>q4 then floor((a-q4)/(0.1*q4)) else 0;
        val p4= map profit q4list;
        fun final(x::xs,y::ys,z::zs,w::ws,(a,b,c,d,e)::es)=
        floor((Real.fromInt(e)*(1.0+0.01*(Real.fromInt(x+y+z+w))))::final(xs,ys,zs,ws,es)|
        final([],[],[],[],[])= [];
in
  final(p1,p2,p3,p4,l)
end;
```

```
fun budgetRaise(l) =
let
  val l1 = qPerformance(l);
  fun sum(x,y:real)=Real.fromInt(x)+y;

  fun q1(a,b,c,d,e) = (e);
  val q1list = map q1 l;
  val totalbefore= foldr sum 0.0 q1list;

  val budget= foldr sum 0.0 l1;
in
  (budget-totalbefore)/totalbefore
end;
```

Correctness Proof:

For qPerformance(l):

Variable len stores the number of employees.

Q_i list stores the i^{th} quantity of tupples in the form of list (performance of the i^{th} quarter)

Variable q_i then stores the average performance of the i^{th} quarter

Variable p_i stores a list which contain the percentage increase in of the employees' salary due to the i^{th} quarter.

fun final takes the p_i lists and map the final salary of each employee taking his initial salary and percentage increase in salary due to all quarters.

Hence we get the final salaries of each employee

Time Complexity = O(length(l)) (Map and foldr both use O(length(l) in this program)

Space Complexity = O(length(l)) (At any instant we store c*(length(l)) elements)

For budgetRaise(l):

First variable l1 stores the list of final salaries of each employee by calling qPerformance(l)

Q1list stores the initial salaries of each employee

totalBefore stores the total salary of all the employees initially

budget stores the sum of final salaries of the employees

then we calculate the factor change in the budget by:

(budget – total before)/totalbefore

Time Complexity = O(length(l)) (Map and foldr both use O(length(l) in this program)

Space Complexity = O(length(l)) (At any instant we store c*(length(l)) elements)

Q3)

ith element (L, i) = fn (L, 1, i)

$$n(L, 1, i) = \begin{cases} \text{raise empty} & L = [\,] \\ hd(L) & j = i \\ n(tl(L), j+1, i) & \end{cases}$$

This function gives the i^{th} element of a list

- It checks whether $j = i$ and if true returns that elem, else iterates either till list becomes empty (i.e. length(L) < i), in which case it raises empty / or else till j becomes equal to i (it goes on removing head of the list with each iteration)

- $$\text{Append}(a, n, ls) = \begin{cases} [\,] & ls = [\,] \\ [[a] @ hd(ls)] @ \text{Append}(a, n, tl(ls)) & \end{cases}$$

It adds an element to each list inside the list ls. It first adds the element a to the first list inside ls then appends it with the rest of the lists till no list is left.

fun del L ind = helper L 1 ind

helper (L, i, n) { raise Empty L = []
                 { tail(xs)   i = n
                 { hd(x) :: helper (tl(L), i+1, n)

It deletes the i^{th} element of the List L.
If $i \neq n$ then it just moves to the next iteration on the tail of L, when $i = n$ then it removes the head of L and returns the tail so that the hd(L) is removed from the resultant list.

fun Permutation(ls) =

{ [ls] if ls = [x]
{ g(length(ls), 1, ls)

g(n, i, L) = { [] i > n
             { Append(ithElement(L, i), length(L) - 1, Permutation(del L i)) @ g(n, i+1, L)

This function returns the permutations of the elements of the list and returns a list of lists like where the ~~n elems~~ k elements in the list.

[a, b, c, . . . , k]

It takes out the elements from the list one by one and then adds it to the permutation of rest of the list and then does the same with another element till all the elements are permutated. The thing to note is that for a sorted list it will return a lexicographic permutation since it starts from the 1st element of the list

---

insort (l) : Already proved in notes

lexicographicPerm (l) :

It first sorts the list and then Permutes it.

Function	Time	Space
i^th element	$O(i)$ (to iterate till i^{th} element)	$O(\text{length}(l))$ (to store the list initially)
Append	$O((\text{length}(l))^2)$ (assuming length(l) @ length(m) is of order length(l))	$O(\text{length}(l) \times \text{length}(\text{hd}(l)))$ (to store list of list)
del	$O(ind)$ (to iterate till ind^th element)	$O(\text{length}(l))$ (to store the list initially)
Permutation	$O((\text{length}(l)+1)!)$ $T_n = n(T_{n-1}) + 2n!$ guess $n! \Rightarrow (n+1)!$	$O(\text{length}(l)!)$
lexicographic perm	$O(\text{Permutation}) + O(\text{length}(l)^2)$ ↑ selection sort $= O((\text{length}(l)+1)!)$	$O($ Permutation $) + O(\text{length}(l))$ $= O(\text{length}(l)!)$

Bonus :

lexicographicPermDup(l) = f(lexicographicPerm(l))

$$f(l) = \begin{cases} [] \\ hd(l) :: [] \\ hd(l) :: f(tl(l)) \\ \quad f(tl(l)) \end{cases}$$

$l = []$
$tl(l) = []$
$g(l) = true$
$g(l) = false$

$$g(l) = \begin{cases} raise\ Empty \\ true \\ false \\ g(hd(l) :: (hd(tl(l)))) \end{cases}$$

$l = []$
$tl(l) = []$
$hd(l) = hd(tl(l))$