Q2)

i)

```
fun maximumValue(n,v,w,W)= let fun f(i,n,weight,W,value)= if i<=n then
                    if (weight + w(i)) <= W then
                        let val y= f(i+1,n,weight+w(i),W,value + v(i)); val x=
                        f(i+1,n,weight,W,value); in
                        if y>x then y else x end
                        else  f(i+1,n,weight,W,value)
                    else value
                    in f(1,n,0,W,0) end;
```

Algorithm:

$$f(\text{i, weight, value}) = \begin{cases} max(f(i+1, weight\ +\ w(i), value + v(i)), f(i+1, weight\ , value)) & weight\ +\ w(i) <= W \quad i <= n \\ f(i+1, weight\ , value) & weight\ +\ w(i) > W, i <= n \\ value & i > n \end{cases}$$

Proof:

In variant : weight is always less than W

If we assume that the function remains correct till some i<=k where k<n then for i+1th iteration

Case 1: We'll check the max($f(i+1, weight\ +\ w(i), value + v(i)), f(i+1, weight\ , value)$) only if $weight\ +\ w(i)$ still remains less than W, so our invariant remains true, and therefore get the max of values of the 2 function which we required as an output

Case 2: if $weight\ +\ w(i)\ >\ W$ then invariant will no longer remain true, so in that case we don't increment weight and value in the function and just pass on to the next iteration, so that invariant still remains true while i gets incremented by 1

Case 3: when i becomes greater than n(i>n) i.e. all values have been checked, the function returns the final value, which is the maximum of all possible cases

ii)

Time Complexity = 2^n

Because for every i ≤n we check if it can be used in the maximum value or not

Space Complexity = n

Because n frames get stacked to check function from i=1 to n

Q3)

i)

```
fun inttoString(n)=     if n=1 then"1"
                        else if n= 2 then "2"
                         else if n=3 then "3"
                         else if n=4 then "4"
                         else if n=5 then "5"
                         else if n=6 then "6"
                         else if n=7 then "7"
                         else if n=9 then "9"
                         else "0"
fun toString(n)=        if n=0 then "0"
                        else let
                        fun f(n,a:string)= if n=0 then a
                                           else f(n div 10, inttoString(n mod 10)^a)
                         in f(n,"") end;
```

$$f(n,a) = \begin{cases} a, & n = 0 \\ \text{f(n div 10, inttoString(n mod 10)\^a)}, & n \geq 0 \end{cases}$$

```
fun convertUnitsRec(n,factor,name)= let
        fun f(n,i) = if n=0 then ""
        else f(n div factor(i), i+1)^toString(n mod factor(i))^name(i)
        in f(n,0) end;
```

$$f(n,i) = \begin{cases} "" & , \quad n = 0 \\ \text{f(n div factor(i), i + 1)\^toString(n mod factor(i))\^name(i)}, & n > 0 \end{cases}$$

Correctness Proof:

1) inttostring(n): this function takes a single digit integer as input and the same integer but in the string format
2) toString(n): it first checks if the input is 0 and returns “0” if its true but if the number is not zero then it recurses over the number of digits in the number and converts it into string as follows:

Correctness Proof:

Invariant: in f(n,a), toString(n)^a always will be equal to the input number in string format or toString(n) where n is the original input.

Base Case: if n=0(inside the let part) then it returns a

“”^a = toString(n)

Induction Step:

For n>0 f(n,a)= $f(n \text{ div } 10, \text{inttoString}(n \text{ mod } 10)\text{^a})$

Invariant still holds as:

toString(n div 10)^(inttoString(n mod 10)^a) =

Since, toString(n)=toString(n div 10)^inttostring(n mod 10).

3) convertUnitsRec(n,factor,name):In the function f(n,i):
Correctness Proof:
Base Case: If n=0 then 0
Induction hypothesis: Let f(k,i) be true for all k<n
Induction step:
f(n,0)= $f(n \text{ div } \text{factor}(0), 1)\text{^toString}(n \text{ mod } \text{factor}(0))\text{^name}(0)$
f(n div factor(0),1) is correct by induction hypothesis

so if our input was 86401 and our output was supposed to be
“1 day 0 hours 0 minutes 1 seconds” then f(n div factor(0),1) will return
“1 day 0 hours 0 minutes”

So f(n,0)= “1 day 0 hours 0 minutes”^toString(86401 mod 60)^ “Seconds”
Since, toString(86401 mod 60) = “1”
So f(n,0)= “1 day 0 hours 0 minutes”^ “1” ^“Seconds”
= “1 day 0 hours 0 minutes 1 seconds”

Hence Prooved

ii)

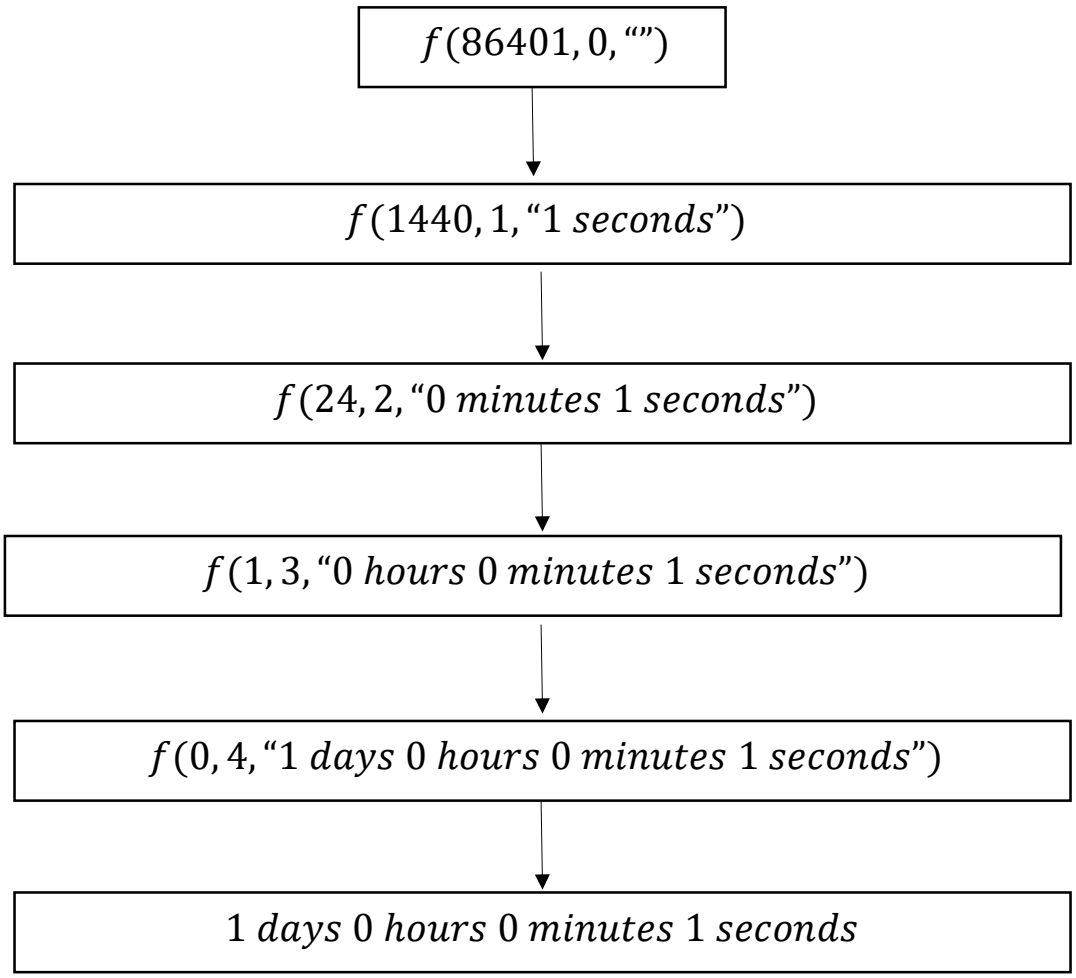
```
fun convertUnitsIter(n,factor,name)=
    let fun f(n,i,os:string) =
            if n=0 then os
            else f(n div factor(i), i+1, toString(n mod factor(i))^name(i)^os)
            in f(n,0,"") end;
```

$$f(\text{n, i, os: string}) = \begin{cases} os, & n = 0 \\ \text{f(n div factor(i), i + 1, toString(n mod factor(i))\^name(i)\^os)}, & n > 0 \end{cases}$$

The above written algorithm is of Space Complexity of O(1) since it does not leave behind any calculations to be done before moving to the next iteration.

For eg: If our input was 86401 and our output was supposed to be
"1day 0 hours 0 minutes 1 seconds"

Then, f(86401, 0, "") will proceed as



iii) Time complexity will be of $O((\log n)^2)$ for both the algorithms since there will be log n terms which will be added some constant*(logn) times which will give $(\log n)^2$

4)

```
fun g(n,a)= if n=0 then a else g(n div 4, a*4);

fun f(n,i,q,p) = if p=1 then
                    if (2*i +1)*(2*i + 1)<=n then 2*i + 1
                    else 2*i
                else f(n,if (2*i +1)*(2*i + 1)<=q then 2*i +1 else 2*i, n div(p div 4), p div 4);

fun intSqrt(n)= f(n,0,n div g(n div 4,1),g(n div 4,1));
```

Correctness Analysis:

g(n,a): it gives the lowest power of 4 which is greater than n.

f(n,i,q,p): it gives the intsqrt of n,

Invariant: at any moment i is the intsqrt of q*4 (where q is : gif($n/4^p$)

Lets assume invariant to be true for some value

For the next iteration q becomes gif($n/4^{p-1}$) therefore

$(2i)^2 <= q < (2i + 2)^2$

So intsqrt q is either 2i or 2i + 1, which we check and get

ii)

the function iterates till p becomes 1, where p is the highest power of 4 less than or equal to n

at each iteration value of p becomes gif(p/4)

therefore function iterates for gif($\log_4 p$) times

hence time complexity is of order log(n)

since the function is iterative, and it finishes all calculations before moving to the next iteration, the frames do not stack up and hence space complexity is of O(1)

iii) since time complexity of O(log n) big numbers like 400,000,000 will 14.28….

so it will take only 15 iterations to get the output

1)

```
fun g(n,a)= if n=0 then a else g(n div 4, a*4);


fun f(n,i,q,p) = if p=1 then
                    if (2*i +1)*(2*i + 1)<=n then 2*i + 1
                    else 2*i
          else f(n,if (2*i +1)*(2*i + 1)<=q then 2*i +1 else 2*i, n div(p div 4), p div 4);


fun intSqrt(n)= f(n,0,n div g(n div 4,1),g(n div 4,1));


fun isPrime(a,b,c)=  if a=2 then true
                          else if a mod b=0 orelse a<=1 then false
                          else if b>c then true
                          else isPrime(a,b+1,c);


fun f(a,n,l)=if a>n then l else if isPrime(a,2,intSqrt(a)) then f(a+1,n,a::l) else f(a+1,n,l);


fun help(a,[])=false | help(a,[x])= if a=x then true else false | help(a,x::l)= if a=x then true else help(a,l);


fun helper(n,a,b,l)=if a>n then (0,0,0)
      else if b>a orelse b>(n-a) then helper(n,a+1,2,l)
      else if help(a,l) andalso help(b,l) then
                if isPrime(n-a-b,2,intSqrt(n-a-b)) then (a,b,n-a-b) else helper(n,a,b+1,l)
          else helper(n,a,b+1,l);


fun  findPrimes(n)= let val l=f(2,n,[])
 in helper(n,2,2,l) end;
```

i) Correctness analysis:

```
fun g(n,a)= if n=0 then a else g(n div 4, a*4);


fun f(n,i,q,p) = if p=1 then
                    if (2*i +1)*(2*i + 1)<=n then 2*i + 1
                    else 2*i
                else f(n,if (2*i +1)*(2*i + 1)<=q then 2*i +1 else 2*i, n div(p div 4), p div 4);


fun intSqrt(n)= f(n,0,n div g(n div 4,1),g(n div 4,1));
```

These functions are used in calculating the integer square root, correctness proof is done above for question number 4.

```
fun isPrime(a,b,c)=  if a=2 then true
                        else if a mod b=0 orelse a<=1 then false
                        else if b>c then true
                        else isPrime(a,b+1,c);
```

This function is used to find whether a number is prime, it iterates from 2 to c. Here b is the counter variable and c is the integer sqare root of a, if there is any number between 2 to c which divides a then a is not prime and the function returns false. Else if the count variable exceeds c, then it means the number a is prime and the function returns true.

```
fun f(a,n,l)=if a>n then l
                else if isPrime(a,2,intSqrt(a)) then f(a+1,n,a::l)
                else f(a+1,n,l);
```

This function takes an input n and a count variable a which is initialised at 2 during the function call. It checks whether and number is a prime between a to n and if they are then it is added to the list l. When a exceeds n, then it returns the list l, which contains all prime numbers between a to n.

```
fun help(a,[])=false |
 help(a,[x])= if a=x then true else false |
```

```
help(a,x::l)= if a=x then true else help(a,l);
```

This function is used to check whether the input element a is the part of the list l. if the list is empty then it returns false, else if it contains only one element then it checks if that number is a, else if it has more than 1 element then it checks if the first element of this list is a. if it is then it returns true, else it checks for the rest of the list by reiterating the same process for input a and list which is the tail of the original list.

```
fun helper(n,[],q,l) =(0,0,0) |
helper(n,x::p, [],l) = helper(n,p,l,l) |
helper(n,x::p,y::q,l)= if (n-x-y) <0 then helper(n,x::p,q,l)
                       else if  isPrime(n-x-y,2, intSqrt(n-x-y)) then (x,y,(n-x-y))
                       else helper(n,x::p,q,l);
```

- This function takes 2 elements x and y from the input list l and checks whether n-x-y is a prime or not. If not, then it moves on to the next element of the list q keeping the list p constant.(3^{rd} condition of algorithm)
- When the list p becomes empty(i.e. no such n-x-y is found which is prime) then it returns 0,0,0 as the output
- When the list q becomes empty I, then the function reinitialises list q to l while the first list is replaces by the tail of itself so that it checks for the next element of this list.

```
fun  findPrimes(n)= let val l=f(2,n,[])
 in helper(n,l,l,l) end;
```

this function just stores the value of f(2,n,l) in variable l and uses it to call the helper function.

ii) Finding the list l by function f(a,n,l) takes order $n^{1.5}$ but it is just stored in a variable only once. Helper(n,p,q,l) function takes order $n^{2.5}$ as it checks 2 variables x and y within the list containing maximum n elements takes order n^2 and for each such checks whether the number is prime or not, which takes order $n^{0.5}$ space, which gives total time complexity of order $n^{2.5}$

Space complexity of this function is O(1) since it does not leave any calculations to be done before moving on to the next iteration.