Q1)

1.1) n^{100}, 2^n, $n!$, n^n

$n^{100} = O(2^n)$

Because n^{100} is of polynomial complexity while 2^n is exponential.

$2^n = O(n!)$

Because $n! > 2^n$ for $n \geq 4$

$n! = O(n^n)$

Because $n^n = n \times n \times n \times n \ldots$ n times

while $n! = n(n-1)(n-2) \ldots (1)$

which makes $n! \leq n^n$

1.2) a) 82

b) Sum of elements which are even

1.3) Time complexity of $f(n) =$ $O(n^{2.5})$

Because $O(g(n)) = n^2$

helper recurses $\lceil \sqrt{n} \rceil$ times

$\therefore O(f_{helper}) = n^2 \times \sqrt{n} = n^{2.5}$

$O(f(n)) = O(O(f_{helper}) + O(hn) + 1)$

$= O(n^{2.5} + n + 1) = O(n^{2.5})$

Scanned with CamScanner

Q2)

2.1) Functions bar1 and bar2 both Takes a list as an input and return a list which is the reverse order of the list which was input.

for eg:- Let input list be = [1, 5, 2, 92, 53]

Output will be = [53, 92, 2, 5, 1]

2.2) bar2 is the tail recursive function

2.3) fun bar2 [] L = L

| bar2 (x :: xs) L = bar2 xs (x :: L)

(L_4 L_3 l_2 l_1)

Invariant:

At any point in iteration

(bar2 L_3 []) @ (x :: xs) = Original input

Base case: (bar2 [] L) [] = L_4 = original input

Induction step:

bar2 L_4 L_3 = bar2 l_2 l_1

To prove

(bar2 l_1 []) @ l_2 = Original Input

l_1 = x :: L_3

bar2 [L_3] @ [x] @ xs = Invariant satisfied

Hence proved

24) $T_n = T_{n-1} + (n-1) + 1$

$= T_{n-1} + n$

$T_n - T_{n-1} = n$

$T_{n-1} - T_{n-2} = n-1$

$T_{n-2} - T_{n-3} = n-2$

$\vdots$

$T_2 - T_1 = 2$

$T_1 - T_0 = 1$

$T_n - T_0 = n + (n-1) + (n-2) + \cdots + 1$

$T_n - T_0 = \frac{n(n+1)}{2}$

$T_0 = 0$

$T_n = \frac{n(n+1)}{2}$

$O(T_n) = n^2$

Scanned with CamScanner

Q3) 3.1)

```
fun smallest L =
  let
    fun helper [] i = raise Empty
      | helper [x] i = (x, i)
      | helper (x::xs) i =
          let val (y_val, y_ind) = helper xs (i+1)
          in
            if x < y_val
            then (x, i)
            else (y_val, y_ind)
          end
  in
    helper L 0
  end;
```

3.2)

```
fun del L ind =
  let
    fun helper (x::xs) i n =
        if i = n
        then xs
        else x::(helper xs (i+1) n)
      | helper [] i n = raise empty
  in
    helper L 0 ind
  end;
```

Scanned with CamScanner

3.3)

```
fun selectionSort [] = []
  | selectionSort L =
      let
        val (k, ind) = smallest L
      in
        k :: (selectionSort (del L ind))
      end;
```

Scanned with CamScanner

Q4) 4.i)

```
fun stickCut (n, p) =
  let
    f(n, p, value, space, i, cut) =
      if i > n
        then (value, cut)
      else if
        (space + i > n) then f(n, p, value, space, i+1, cut)
      else
        let
          val (x, L1) = f(n, p, value + p(i), space + i, i, i::cut);
          val (y, L2) = f(n, p, value, space, i+1, cut);
        in if x > y
          then (x, L1)
            else (y, L2)
      end
  in
    f(n, p, 0, 0, 1, [])
  end;
```

4.2)

Function f iterates from i = 1 to n
at each turn if first checks if
space + i > n , if its true
then we cannot make the cut of i length
more since the length of the stick does not
permit it

so we move to the next case

If it is permitted i.e. space + i ≤ n
then we take variable x and y
x : is one in which we make the cut of
length i and then iterate again with same i
so as to find if we can make one more cut.

y is one in which we don't make cut
of length i.

If x > y then function will return x
else y

where x and y are the values
for exit condition when i becomes greater
than n then f returns the value
(variable value) and the length of cuts that it has taken
to get that value.

Scanned with CamScanner