Question 1

(a)

**Base case** when $n = 2$:

$$ (2! = 2 \times 1 = 2) < (2^2 = 4) $$

**Inductive Hypothesis**: assume that $n! < n^n$ for some integer $n > 1$

Now, using the inductive hypothesis, we need to prove that the above holds for $n + 1$ i.e. that $(n + 1)! < (n + 1)^{n+1}$

**Proof**: We first reduce the equation in such a way that enables us to use the inductive hypothesis i.e.

$$ (n + 1)! < (n + 1)^{n+1} $$

$$ (n + 1)n! < (n + 1)(n + 1)^n $$

$$ n! < (n + 1)^n $$

The above equation holds because using the inductive hypothesis $n! < n^n$ and the fact that $n^n < (n + 1)^n$ we deduce that:

$$ n! < n^n < (n + 1)^n $$

and hence that $n! < n^n$ for all integers $n > 1$.

(b)

**Base case** when $n = 1$:

21 divides $4^{1+1} + 5^{2-1} = 16 + 5 = 21$
Inductive Hypothesis: assume that 21 divides $4^{n+1} + 5^{2n-1}$ for an integer $n > 0$

Now, using the inductive hypothesis, we need to prove that the above holds for $n + 1$ i.e. that 21 divides $4^{n+2} + 5^{2n+1}$

Proof: We first reduce the problem in such a way that enables us to use the inductive hypothesis. Here we add and subtract the term $4^{n+1} * 5^2$ and we obtain:

$$
(4^{n+2} + 5^{2n+1}) + 4^{n+1} * 5^2 - 4^{n+1} * 5^2
$$

$$
5^2(4^{n+1} + 5^{2n-1}) + 4^{n+2} - 4^{n+1} * 5^2
$$

$$
25(4^{n+1} + 5^{2n-1}) - 4^{n+1}(5^2 - 4)
$$

$$
25(4^{n+1} + 5^{2n-1}) - 4^{n+1}(21)
$$

The above expression is divisible by 21 because from the inductive hypothesis we know that the first term is divisible by 21 and we see that the second term is divisible by 21 as well. Therefore the whole expression is divisible by 21 and hence 21 divides $4^{n+1} + 5^{2n-1}$ for all integers $n > 0$.

**Question 2**

**R-3.20**

- The first assignment, int $a$, takes constant time $O(1)$.

- The comparison, $i < n$, will be repeated $n+1$ times within the for loop and hence is $O(n)$. 


• The counter, \(i\), will be initialized and incremented \(n\) times and this takes \(O(n)\).

• The assignment within the for loop, \(a = i\), will be repeated \(n\) times and hence is \(O(n)\).

Therefore the algorithm runs in \(O(1 + n + n + n)\) i.e. \(\in O(n)\).

**R-3.21**

• The for loop of this algorithm will repeat for \(n/2\) times instead of \(n\) times. Otherwise it's identical to the previous algorithm.

Therefore the algorithm runs in \(O(1 + n/2 + n/2 + n/2)\) i.e. \(\in O(n)\).

**R-3.22**

• The for loop of this algorithm will repeat for \(n^2\) times.

As before, the algorithm runs in \(O(1 + n^2 + n^2 + n^2)\) i.e. \(\in O(n^2)\).

**R-3.23**

• The outer for loop will be repeated \(n\) times. For each iteration of the outer loop, \(i\) operations will be performed where \(i\) varies between 1 and \(n\).

The algorithm runs in \(O(1 + 2 + 3 + ... + (n - 1) + n)\) i.e. \(O(\sum_{i=1}^{n} i) = O(\frac{n(n+1)}{2}) \in O(n^2)\).
R-3.24

- The outer for loop will be repeated $n^2$ times. For each iteration of the outer loop, $i$ operations will be performed where $i$ varies between 1 and $n^2$.

The algorithm runs in $O(1 + 2 + 3 + \ldots + (n^2 - 1) + n^2)$ i.e. $O(\sum_{i=1}^{n^2} i) = O\left(\frac{n^2(n^2+1)}{2}\right) \in O(n^4)$.

C-3.4

\[
\sum_{i=1}^{n} i^2 = 1^2 + 2^2 + 3^2 + \ldots + (n - 1)^2 + n^2
\]

\[
= \frac{n(n + 1)(2n + 1)}{6} = \frac{2n^3 + 3n^2 + n}{6} \in O(n^3)
\]

Question 3

func popN(List L, int n): returns List

begin;
    if(isempty(L) || (n < 1))
        return newlist();
    else
        SubList=popN(sub(L), n-1);
        push(top(L),SubList);
        return SubList;
    end if;
end func;
func merge(List A, List B);

begin;
if(isempty(B))
    return;
else if (isempty(A))
    A := B;
else if(top(B)<top(A))
    push(top(B),A);
    merge(sub(A),sub(B));
else
    merge(sub(A),B);
end if;
end func;

Analysis of LmergeSort

- length(L) is $O(n)$.

- popN(L,n) is $O(n)$.

- merge(A,B) is $O(n)$.

Hence the recurrence relation is:

$$T(n) = 2T(n/2) + O(n) + O(n) + O(n)$$

i.e.

$$T(n) = 2T(n/2) + kO(n)$$
where $k$ is a constant.

by iteratively substituting for $T(n/2)$ in the recurrence relation above we’ll obtain the following after $i$ times:

$$T(n) = 2^i T(n/2^i) + kin$$

assuming that $n = 2^h$ for some integer $h$, the above process will terminate after $h$ times. $h = \log(n)$ and hence we obtain the following:

$$T(n) = 2^{\log(n)}(O(1)) + kn\log(n)$$

$$= nO(1) + kn\log(n)$$

This means that the Big-Oh running time for LmergeSort is $O(n\log n)$. 