Q. 1.

a. $f(n) = (n^n)^3$; $g(n) = n^{(n^3)}$

$f_n = n^{3n}.$ $g_n = n^{n^3}$

Taking log both sides.

$f(n) = 3n \log n.$ $g(n) = n^3 \log n.$

$\lim_{n\to 0} \frac{f_n}{g_n} = \frac{3n \log n}{n^3 \log n} = \frac{3}{n^2} = 0.$

Since $\lim_{n\to 0} \frac{f_n}{g_n} = 0$. (We can conclude)

Answer.

$$\boxed{① \; f(n) = o\, g(n) \qquad 2.\; f(n) = O\, g(n) \; \text{(Little \& Big Oh.)}}$$

b) $f(n) = n!$; $g(n) = (n+1)!$

$f(n) = 1\times 2\times 3 \ldots \times (n-1)n$; $g(n) = 1\times 2\times 3\times \ldots \times (n-1)\, n\, (n+1)$

$f(n) \approx n^n.$; $g(n) \approx n^{n+1}$

$\lim_{n\to 0} \frac{f(n)}{g(n)} = \frac{n^n}{n^{n+1}} = n^{(n-n-1)} = \frac{1}{n} = 0.$

We can conclude the following.

Answer.

$$\boxed{① \; f(n) = o\, g(n) \;\&\; ② \; f(n) = O\, g(n) \; \text{(Little \& big Oh.)}}$$

c. $f(n) = n^{0.99} + 15(\log_2 n)^{100}$; $g(n) = 4^{\log_{16} n^2}$

$f(n) = n^{0.99} + 15(\log_2 n)^{100}$; $g(n) = n^{2 \cdot \log_{16} 4} = n^{2 \times 1/2} = n$

$g(n) = n$

$\boxed{\text{Assymtotically equal}} = f(n)\ \Theta\ g(n)$

or

Alternatively we can take log both sides

$f(n) = 0.99\ \underbrace{\log n} + 1500 \log \log n$; $g(n) = \underbrace{\log n}$

We $\therefore$ Conclude that

$$\boxed{\begin{aligned} &① \ f(n) = \Theta\ g(n) \\ &② \ f(n) = O\ g(n) \\ &③ \ f(n) = \Omega\ g(n) \end{aligned}}$$ ← Answer.

4. $f(n) = n^{0.000001}$; $g(n) = (\log n)^{1000001}$

Taking logs both the sides

$f(n) = 0.000001 \underline{\log n}$; $g(n) = 1000001 \underline{\log \log n}$

$f(n) \geq g(n)$

We can conclude

$$\boxed{\begin{aligned} &① \ f(n) = \Omega\ g(n) \\ &② \ f(n) = \omega\ g(n) \end{aligned}}$$ ← Answer.

5. $f(n) = n^{5/\log_2 n}$; $g(n) = 10000$

Take logs both sides

$f(n)\ \frac{5}{\cancel{\log_2 n}} \times \cancel{\log n}$; $g(n) = \log 10000$

$f(n) = 5$; $g(n) = \log 10000$

$\therefore$ Assymtotically equal.

$$f(n) \approx g(n)$$

$$\boxed{\begin{array}{l} ① \; f(n) \;\; \Theta \;\; g(n) \\ ② \; f(n) \;\; O \;\; g(n) \\ ③ \; f(n) \;\; \Omega . g(n). \end{array}}$$ → Answer.

Problem 1-2.

1 $\dfrac{n^2+2}{1+n^3 . 2^{-n}} \approx \dfrac{n^2}{n^3 2^{-n}} \approx \dfrac{2^n}{n} \approx \Theta(2^n)$ ← Answer.

2. $\log(\sqrt{\log(\sqrt[3]{n})}) . \log.(37n^3+45) = \log_2 n^{1/3} \log_2 n^3$

$\approx \frac{1}{3} \times 3 \times \log n \times \log n = (\log n)^2 \approx \Theta(\log n)^2$ ← Answer.

3. $\log(n!) + 10^{205} n \approx \log.(n^n) + n \approx n \log n + n$

$\approx \Theta(n \log n)$. ← Answer.

4. $\dfrac{6^n - 1000}{2^n + 1} \approx \dfrac{6^n}{2^n} \approx 3^n \approx \Theta.(3^n)$ ← Answer.

5. $n^{2/\log n} + 1000 \approx n^{2 . 1/\log n} \approx \cancel{2}$

$\approx \cancel{\Theta(}\Theta)$ ← Answer.

6. $2^{n+\log n} \approx 2^n . 2^{\log n} \approx 2^n . n^{\log_2 2}$

$\cancel{2^n}$ $\approx 2^n \times n \approx \Theta . n 2^n$ ← Answer.

7. $2^n + 10(\log n)^{\log n} \simeq 2^n = \theta(2^n) \leftarrow$ Answer.

8. $2^{3n} + 4^n \simeq 2^{3n} + 2^{(n+1)} \simeq 2^{3n} + 2^n$

$\simeq \theta(2^{3n}) \leftarrow$ Answer.

b) Functions for asymptotic growth. (least to maximum.)

~~$e, (\log n)^2, n \log n, 2^n, 2^n, n2^n, 3^n,$~~

$e, (\log n)^2, n \log n, 2^n, 2^n, n2^n, 2^{3n}, 3^n.$

Problem 1-3

A.

```
bubbleSort(A,n)
for j=0 to n-1
     checkSwap=false
     for j = 0 to n-1
       if A[j] > A[j+1] then
         swap( A[j], A[j+1] )
         checkSwap = true
       end if
     end for
   if(not checkSwap) then
       break
     end if
  end for
end bubbleSort return A
```

B.

Proof of Correctness Loop Invariant

After each iteration of the loop greatest element of the array is always placed at right most position. So, at the end of i iteration right most i elements are sorted and in place. After the N-1 iterations the right most N-1 items are sorted and this implies that all the N items are sorted.

Worst Case Time Complexity: O(n*n)
Worst case occurs when array is reverse sorted.Example:9,8,7,6,5,4

Best Case Time Complexity: O(n).
Best case occurs when array is already sorted.Example:4,5,6,7,8,9

c.

D.

```
void bubbleSort(int arr[], int n)
{
  int i, j;
  bool swapped;
  for (i = 0; i < n-1; i++)
  {
    swapped = false;
    for (j = 0; j < n-i-1; j++)
    {
      if (arr[j] > arr[j+1])
      {
        swap(&arr[j], &arr[j+1]);
```

```
        swapped = true;
      }
    }
     if (swapped == false)
      break;
  }
}
```

In this example, we need to take the best case time complexity.
Best Case Time Complexity: O(n)

E.

```
void insertionSort(int arr[], int n)
{
  int i, key, j;
  for (i = 1; i < n; i++)
  {
     key = arr[i];
     j = i-1;
     while (j >= 0 && arr[j] > key)
     {
        arr[j+1] = arr[j];
        j = j-1;
     }
     arr[j+1] = key;
  }
}
```

The while loop executes only if i > j and arr[i] < arr[j]. Therefore total number of while loop iterations (For all values of i) is same as number of inversions.
Therefore overall time complexity of the insertion sort is O(n + f(n)) where f(n) is inversion count.
If the inversion count is O(n), then the time complexity of insertion sort is O(n).

F.
An array example for which bubble sort runs for O(N) and insertion sort runs in O(N^2) :{1,2,5,3,4}

In bubble sort:
A={1,2,5,3,4}

Comparing 1 and 2:Since 2>1 no swapping will happen.Array stays the same {1,2,5,3,4}
Comparing 2 and 5:Since 5>2 no swapping will happen.Array stays the same {1,2,5,3,4}
Comparing 5 and 3:5>3 it will swap 5 and 3.Array becomes {1,2,3,5,4}
Comparing 5 and 4:5>4 it will swap 5 and 4.Array becomes {1,2,3,4,5}

Since the array has become sorted in one iteration,time complexity remains as O(N).

In insertion sort:

A={1,2,5,3,4}

Starting j from the second element i.e 2

Key element=2,A[j-1]=1

Comparing 2(key element) with A[j-1]=1 ,since 1<2 so won't check for rest and will move on to the next element.

Array stays the same {1,2,5,3,4}

Key element=5,A[j-1]=2

Comparing 5(key element) with A[j-1]=2 ,since 2<5 so won't check for rest and will move on to the next element.

Array stays the same {1,2,5,3,4}

Key element=3,A[j-1]=5

Comparing 3(key element) with A[j-1]=5 ,now 5>3 so it will start checking for all prior elements until 3 is placed in its proper position(which will again take O(N)time complexity).

Array becomes {1,2,3,5,4}

Key element=4,A[j-1]=5

Comparing 4(key element) with A[j-1]=5 ,now 5>4 so it will start checking for all prior elements until 4 is placed in its proper position(which will again take O(N)time complexity).

Array becomes {1,2,3,4,5}

So,insertion sort becomes O(N) in this case.