a) $f(n) \in 2^n$, $g(n) \in 2^{2n}$

- For $f \in O(g)$: O notation gives us an upper bound for the complexity.

$f(n) \leq c \cdot g(n)$ → If there exist positive constants such that c and n_0 for $n \geq n_0$

$2^n \leq 4 \cdot 2^{2n}$ → let's give $c=4$, $n_0=1$

$2^1 \leq 4 \cdot 2^{2n}$

$2 \leq 16$ → $f \in O(g)$

for $f \in \Omega(g)$: Ω notation gives us a lower bound for the complexity.

$f(n) \geq c \cdot g(n)$ → If there exist positive constants such that c and n_0 for $n \geq n_0$

$2^n \geq c \cdot 2^{2n}$ → let's give $c=1$, $n_0=1$

$2 \geq 1 \cdot 2^{2 \cdot 1}$

$2 \geq 4$ $f \notin \Omega(g)$

for $f \in \Theta(g)$:

$c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$

$c_1 \cdot 2^{2n} \leq 2^n \leq c_2 \cdot 2^{2n}$ → let's give $c_1=1$, $n=1$, $c_2=2$

$1 \cdot 2^2 \leq 2^1 \leq 2 \cdot 2^{2 \cdot 1}$

$4 \leq 2 \leq 8$ → this is not true $f \notin \Theta(g)$

Limit :

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = \frac{2^n}{2^{2n}} = \lim_{n \to \infty} \frac{1}{2^n} = 0 \Rightarrow f(n) \in O(g)$$

b) $f(n) \in n^2$, $g(n) \in n^3$

for $f \in O(g)$:

$f(n) \leq c \cdot g(n)$ let's give $c=2$, $n_0 = 2$

$n^2 \leq c \cdot n^3$

$2^2 \leq 2 \cdot 2^3$

$4 \leq 16$ → this is true. $\boxed{f \in O(g)}$

for $f \in \Omega(g)$:

$f(n) \geq c \cdot g(n)$ Let's give $c=2$, $n_0 = 2$

$n^2 \geq c \cdot n^3$

$2^2 \geq 2 \cdot 2^3$

$4 \geq 16$ this is not true. $\boxed{f \notin \Omega(g)}$

for $f \in \Theta(g)$:

$c_1 \cdot gn \leq f(n) \leq c_2 \cdot gn$ let's give $c_1 = 1$, $n_0 = 2$, $c_2 = 2$

$c_1 \cdot n^3 \leq n^2 \leq c_2 \cdot n^3$

$2^3 \leq 2^2 \leq 2 \cdot 2^3$ → this is not true

$\boxed{f \notin \Theta(g)}$

Limit:

$$\lim_{n \to \infty} \frac{n^2}{n^3} = 1 \cdot \frac{1}{n} = \frac{1}{\infty} = 0$$

$f \in O(g)$

c) $f \in 3n+1$, $g \in 2n-5$

for $f \in O(g)$:

$f(n) \leq c.g(n)$ let's give $c=6$, $n_0 = 5$

$3n+1 \leq c.(2n-5)$

$3.5+1 \leq 6.(2.5-5)$

$16 \leq 6.5$

$16 \leq 30 \longrightarrow$ $\boxed{f \in O(g)}$

for $f \in \Omega(g)$:

$f(n) \geq c.g(n)$ let's give $c=3$, $n_0 = 3$

$3n+1 \geq c.(2n-5)$

$3.3+1 \geq 3.(2.3-5)$

$10 \geq 3$. this is true $\boxed{f \in \Omega(g)}$

for $f \in O(g)$:

$c_1.g(n) \leq f(n) \leq c_2.g(n)$ let's give $c_1 = 2$, $c_2 = 6$, $n_0 = 5$

$2.(2n-5) \leq 3n+1 \leq 6.(2n-5)$

$2.(2.5-5) \leq 3.5+1 \leq 6.(2.5-5)$

$10 \leq 16 \leq 30$ this is true $\boxed{f \in \Theta(g)}$

Limit:

$$\lim_{n\to\infty} \frac{3n+1}{2n-5} = \lim_{n\to\infty} \left(1 \left(\frac{3+\frac{1}{n}}{2-\frac{5}{n}}\right)\right) = \frac{3+\frac{1}{\infty}}{2-\frac{5}{\infty}} = 1.\frac{3}{2} = \frac{3}{2}$$

Since the result is a constant, it satisfies all notation

$\boxed{f \in O(g) \ , f \in \Omega(g) , f \in \Theta(g)}$

d) $f \in 4n^2$, $g \in n^2$

for $f \in O(g)$:

$f(n) \leq c_1 \cdot g(n)$

$4n^2 \leq c_1 \cdot n^2$ Let's give $c_1 = 4$, $n = 1$

$4 \leq 4$ this is true

$f \in O(g)$

for $f \in \Omega(g)$

$f(n) \geq c_1 \, g(n)$ Let's give $c_1 = 4$, $n = 1$

$4n^2 \geq c_1 \, n^2$

$4 \geq 4$ this is true $f \in \Omega(g)$

for $f \in \Theta(g)$

$c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$ Let's give $c_1 = 3$, $n = 1$, $c_2 = 4$

$c_1 \cdot n^2 \leq 4n^2 \leq c_2 \cdot n^2$

$3 \cdot 1^2 \leq 4 \cdot 1^2 \leq 4 \cdot 1^2$

$3 \leq 4 \leq 4$ this is true

$f \in \Theta(g)$

Limit:

$\lim_{n \to \infty} \frac{4n^2}{n^2} = 4$ Constant $\Rightarrow$ $f \in O(g)$, $f \in \Omega(g)$, $f \in \Theta(g)$

c) $f \in \log_2(n)$, $g \in \log_{10}(n)$

Since the calculation will be long, it is much simpler to solve this question with the limit.

$$\lim_{n\to\infty} \frac{\log_2(n)}{\log_{10}(n)} = \text{L'opital Rules.}$$

$$\lim_{n\to\infty} = \frac{\left(\frac{1}{n * \ln(2)}\right)}{\left(\frac{n}{n * \ln(10)}\right)} = \frac{1}{n * \ln(2)} \cdot \ln(10) = \frac{\ln(10)}{\ln(2)} \approx 3.3$$

Since the result is a constant, satisfied all notations.

$f \in O(g)$, $f \in \Omega(g)$, $f \in \Theta(g)$.

f) $f \in 2^n$, $g \in 3^n$

for $f \in O(g)$:

$f(n) \leq c_1 . g(n)$

$2^n \leq c_1 . 3^n$ Let's give $c_1 = 2$, $n_0 = 2$

$2^2 \leq 2 . 3^2$

$4 \leq 18$ this is true. $f \in O(g)$

for $f \in \Omega(g)$:

$f(n) \geq c_1 . gn$ Let's give $c_1 = 1$ $n = 1$

$2 \geq 3$ this is not true $f \notin \Omega(g)$

for $f \in \Theta(g)$:

Since $f \in O(g)$ and $f \notin \Omega(g)$, $f \notin \Theta(g)$

Limit:

$\lim_{n\to\infty} \frac{2^n}{3^n} = 0$ Since the result is 0, $f \in O(g)$

g) $f \in n^3$, $g \in 1000n^2$

For $f \in g(n)$:

$f(n) \leq c_1 . g(n)$ Let's give $c_1 = 1$, $n_0 = 2$

$n^3 \leq c_1 . 1000 . n^2$

$2^3 \leq 1 . 1000 . 2^2$

$8 \leq 4000$ this is true $f \in O(g)$

for $f \in \Omega g(n)$:

$f(n) \geq c_1 . g(n)$ Let's give $c_1 = 1$, $n_0 = 1000$

$n^3 \geq c_1 . 1000 . n^2$

$n^2 . n \geq c_1 . 1000 . n^2$

$n \geq c_1 . 1000$

$1000 \geq 1000$ this is true $f \in \Omega(g)$

for $f \in \Theta g(n)$:

$c_1 . g(n) \leq f(n) \leq c_2 . g(n)$ Let's give $c_1 = 1$ $c_2 = 2$ $n_0 = 3$

$1 . 1000 . 3^2 \leq 3^3 \leq 2 . 1000 . 3^2$

this is not true $f \notin \Theta(g)$

Limit:

$$\lim_{n \to \infty} \frac{n^3}{1000 n^2} = \frac{n}{1000} = \frac{\infty}{1000} = \infty$$

the result is ∞ , $f \in \Omega(g)$

h) $f \in 5n+4$, $g \in 2n+2$

Let's solve it with the limit.

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{5n+4}{2n+2} = \lim_{n\to\infty} \frac{5n}{2n} = \frac{5}{2}$$

Since the result is a constant, it satisfies all notation

$f \in O(g)$, $f \in \Omega(g)$, $f \in \Theta(g)$

i) $f \in \sqrt{n}$, $g \in \log_2 n$

$f(n) \leq c \cdot g(n)$: Let's give $c_1 = 2$, $n = 2$.

$\sqrt{2} \leq 2 \cdot \log_2 2$ this is true. $f \in O(g)$

$f(n) \geq c \cdot g(n)$ let's give $c = 2$ $n = 2$

$\sqrt{n} \geq c \cdot \log_2 n$

$n \geq c \cdot (\log_2 n)^2$

$2 \geq 2 \cdot (\log_2 2)^2$ this is not true.

$f \notin \Omega(g)$

$c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$ let's give $c_1 = 1$ $n = 2$ $c_2 = 2$

$1 \cdot \log_2 n \leq \sqrt{n} \leq 2 \cdot \log_2 n$

$1 \log_2 n \leq \sqrt{2} \leq 2 \cdot \log_2 2$

$(\log_2 2)^2 \leq 2 \leq (2 \log_2 2)^2$ $\Rightarrow$ this is true

$f \notin \Theta(g)$

Limit:

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{\sqrt{n}}{\log_2 n} = \text{L'opital Rule.}$$

$$\lim_{n\to\infty} \frac{\frac{1}{2\sqrt{n}}}{\frac{1}{\ln(2)}} \qquad \lim_{n\to\infty} \frac{n^k \ln(2)}{2\sqrt{n}} = 0$$

$f \in O(g)$

j) $f(n) \in 2^n$ and $g(n) = 2^{n+1}$

$f(n) \leq c_1 . g(n)$

$2^n \leq c^* 2^n . 2^1$

$1 \leq 2c$ that is, constant c must b $\frac{1}{2}$ or greater.

because of this $\boxed{f \in O(g)}$ $\therefore \Rightarrow$

$f(n) \geq c_1 g(n)$

$2^n \geq c . 2^{n+1}$

$2^n \geq c . 2^n . 2$

$1 \geq c . 2$ this is not true because of this

$\boxed{f \notin \Theta(g)}$ $\boxed{f \notin \Omega(g)}$

2) I compared each function in turn using limits.

First,

$\lim_{n \to \infty} \frac{\frac{1}{2n}}{\log(n)} = \frac{0}{\infty} = 0$ $\frac{1}{2n} < \log(n)$

$\lim_{n \to \infty} \frac{\log(n)}{\sqrt{n+5}} = \frac{\infty}{\infty} = \text{lópital} \lim_{n \to \infty} = \frac{\frac{1}{n}}{1/2\sqrt{n+5}} = \lim_{n \to \infty} \frac{2}{\sqrt{5}} = 0$

$\frac{1}{2n} < \log n < \sqrt{n+5}$

$\lim_{n \to \infty} \frac{\sqrt{n+5}}{n+1} = \frac{\infty}{\infty}$ by doing rationalization

$\lim_{n \to \infty} \frac{\sqrt{n+5}}{n+1} \times \frac{1}{1} = \lim_{n \to \infty} \frac{\sqrt{1 + \frac{5}{n}}}{\sqrt{n}\left(1 + \frac{1}{n}\right)} = \lim_{n \to \infty} \frac{\sqrt{1 + \frac{5}{\infty}}}{\sqrt{\infty}\left(1 + \frac{1}{\infty}\right)}$

$\lim_{n \to \infty} \frac{\sqrt{n+5}}{n+1} = 0$

$\frac{1}{2n} < \log n < \sqrt{n+5} < n+1$

$$\lim_{n\to\infty} \frac{n+1}{10^n} = \frac{\infty}{\infty} = \text{L'hospital Rule}$$

$$\lim_{n\to\infty} = \frac{1}{10^n \log(10)} = \frac{10^{-n}}{\log 10} = \frac{1}{10^n \log(10)} = \frac{1}{10^\infty \log(10)} = \frac{1}{\infty} = 0$$

$$\frac{1}{2} < \log n < \sqrt{n+5} < n+1 < 10^n$$

$$\lim_{n\to\infty} \frac{10^n}{n^2 \log(n)} = \text{L'hospital rule}$$

$$\lim_{n\to\infty} \frac{10^n}{2n\log n + n} \quad \infty \qquad 2n\log n + n < 10^n$$

$$\lim_{n\to\infty} \frac{n^2 \log n}{n+1} = \frac{\infty}{\infty} = \text{L'hospital} = \frac{2n\log n + n^2 \cdot \frac{1}{n}}{1} = \infty$$

$$\frac{1}{2} < \log n < \sqrt{n+5} < n+1 < n^2 \log n < 10^n$$

$$\lim_{n\to\infty} \frac{2^n}{10^n} = 0 \qquad 2^n < 10^n$$

$$\lim_{n\to\infty} \frac{2^n}{n^2 \log n} = \infty \qquad n^2 \log n < 2^n$$

$$\frac{1}{2} < \log n < \sqrt{n+5} < n+1 < n^2 \log n < 2^n < 10^n$$

$$\lim_{n\to\infty} \frac{10^n}{n!} = \text{L'hospital Rule}$$

$$\lim_{n\to\infty} \frac{10^n \cdot \ln(10)}{n! \cdot \ln(n)} = 0 \qquad 10^n < n!$$

$$\frac{1}{2} < \log n < \sqrt{n+5} < n+1 < n^2 \log n < 2^n < 10^n < n!$$

$$\lim_{n\to\infty} \frac{n!}{n^{2n}} = \lim_{n\to\infty} \frac{n!}{n^n} = 0 \quad (n^n \text{ will grow Faster } n!)$$

$$n! < n^{2n}$$

$$\frac{1}{2} < \log(n) < \sqrt{n+5} < n+1 < n^2 \log(n) < 2^n < 10^n < n! < n^{2n}$$

3-)

a)

```
mergeBST (tree1 , tree2):

  if tree2 is not empty:
      mergeBST (tree1 , tree2.left)
      tree1.insert(tree2.root)
      mergeBST (tree1 , tree2.right)
```

time Complexity (in terms of big O Notation):

- the in-order traversal of the second BST has a time complexity of O(n), where n is the height of the second BST.

- Inserting each element into the first BST takes O(logn) time on average for a balanced BST.
If the first BST is also of height n, the overall time complexity is O(n * log n).

b)

To find the kth smallest element BST, it perform an in-order traversal of the tree while keeping track of the count of nodes visited. When the count reaches k, it found the kth smallest element.

```
FindK (node, k, count):

  if node is not null:
      FindK (node.left , k, count)
      count = count + 1
      if count == k:
          return node.value
      FindK (node.right , k, count)
```

the in-order traversal visits all nodes in ascending order, so it has a time complexity of O(n) where n is the number of nodes in the tree.

c) Common algorithms like AVL or Red-Black tree balancing have an average time complexity of O(n log n) for balancing the tree. The worst-case time complexity can be $O(n^2)$ in certain scenarios.

d-) To find elements within a specified value range [min, max] in a BST, it's checked whether the value of each node is within this range.

```
FindRange (node, min, max, result):
  if node is not null:
    if node.value is greater than min:
      FindRange(node.left, min, max, result)
    if node.value is between min and max:
      result.append (node.value)
    if node.value is less than max:
      FindRange (node.right, min, max, result)
```

The in-order traversal visits all nodes that might fall within the range [min-max]. The number of nodes visited depends on the depth of the subtree containing elements within the range. In the worst-case, it can be O(n) if the entire tree is within the range, but in the average case, it's O(log n) for balanced BST

4-) Here's the breakdown of the code's behavior:

1. IF i is odd (i % 2 != 0), substract 1 from i
2. if i is even, square i and then add 1
3. The loop will run as long as i <= n

Let's see how the value of i changes:

1. i = 2 (even) i becomes 2*2 = 4 i becames 5
2. i = 5 (odd) i becames i became 4
3. i = 4 (even) i becames i become 4*4 = 16 ⇒ i became 17
4. i = 17 (add) i becames 16

.
.
. and it keep going.

From the behavior, we can see that once i is even, it will grow very quickly since it gets squared. This rapid growth will quickly make i much larger than n, ending the loop.

Given this behavior even for large values of n, i will surpass "n" after only a few squarings. Hence, the number of iterations is logarithmic with respect to the size of n.
So time complexity is O(log log n).
This is because the value of i grows exponentially, and the number of iterations is the algorithm of the exponential growth rate.

$$T(n) = O(\log \log n)$$

5-) The algorithm checks each element one by one from start.
It checks every element to see if it's even or odd.
If it's odd, it goes on to the next element to check that one.
If it's even, it's the element we are looking for.
If the algorithm reaches the last element and doesn't find an even element, it should print a message, saying there is no even element in the array.

For average case, %50 of the array is even and %50 of it is odd.

This gives us a constant number for the average case where we definitely run into an even number.

Since the complexity is constant, the time complexity should be O(1).