1.

a) $f(n)=2^n,\ g(n)=2^{2n}=4^n \quad \lim_{n\to\infty} \frac{f(n)}{g(n)} = \frac{2^n}{4^n} = \frac{2^n}{2^n . 2^n} = \frac{1}{2^n} = 0$

therefore; $f(n) \in o(g(n))$

b) $f(n)=n^2,\ g(n)=n^3 \quad \lim_{n\to\infty} \frac{f(n)}{g(n)} = \frac{n^2}{n^3} = \frac{1}{n} = 0$

therefore; $f(n) \in o(g(n))$

c) $f(n)=3n+1,\ g(n)=2n-5 \quad \lim_{n\to\infty} \frac{f(n)}{g(n)} = \frac{3n+1}{2n-5} = \frac{3}{2}$

therefore; $f(n) \in \Theta(g(n))$

d) $f(n)=4n^2,\ g(n)=n^2 \quad \lim_{n\to\infty} \frac{f(n)}{g(n)} = \frac{4n^2}{n^2} = 4$

therefore; $f(n) \in \Theta(g(n))$

e) $f(n)=\log2(n),\ g(n)=\log10(n) \quad \lim_{n\to\infty} \frac{f(n)}{g(n)} = \frac{\log2(n)}{\log10(n)} = \frac{\frac{\log10(n)}{\log10(2)}}{\log10(n)} = \frac{1}{\log10(2)}$

therefore; $f(n) \in \Theta(g(n))$

f) $f(n)=2^n,\ g(n)=3^n \quad \lim_{n\to\infty} \frac{f(n)}{g(n)} = \frac{2^n}{3^n} = \frac{2^n}{2^n . \frac{3}{2}^n} = \frac{1}{\frac{3}{2}^n} = 0$

therefore; $f(n) \in o(g(n))$

g) $f(n)=n^3,\ g(n)=1000n^2 \quad \lim_{n\to\infty} \frac{f(n)}{g(n)} = \frac{n^3}{1000n^2} = \frac{n}{1000} = \infty$

therefore; $f(n) \in \omega(g(n))$

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

therefore; $f(n) \in \Theta(g(n))$

i) $f(n)=\sqrt{n},\ g(n)=\log2(n) \quad \lim_{n\to\infty} \frac{f(n)}{g(n)} = \frac{f'(n)}{g'(n)} = \frac{\frac{1}{2} . n^{-\frac{1}{2}}}{\frac{1}{n\ln 2}} = \frac{1}{2} . \sqrt{n} . \ln 2$

therefore; $f(n) \in \omega(g(n))$

j) $f(n)=2^n,\ g(n)=2^{n+1}=2.2^n$

$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \frac{2^n}{2.2^n} = \frac{1}{2}$

therefore; $f(n) \in \Theta(g(n))$

2.

Functions in the order: $\frac{1}{2n} < \log(n) < \sqrt{n+15} < n+1 < n^2\log(n) < 2^n < 10^n < n! < n^{2^n}$

I can proof my claim by performing limit analysis for every consecutive functions.

$\lim_{n\to\infty} \frac{\frac{1}{2n}}{\log(n)} = \frac{1}{2n.\log n} = 0$ | $\lim_{n\to\infty} \frac{\log n}{\sqrt{n+5}} = \frac{\frac{1}{n\ln 10}}{\frac{1}{2.\sqrt{n+5}}} = \frac{2.n^{\frac{1}{2}}}{n.\ln 10} = \frac{2}{\sqrt{n}.\ln 10} = 0$

$\lim_{n\to\infty} \frac{\sqrt{n+5}}{n+1} = \frac{1}{\sqrt{n}} = 0$ | $\lim_{n\to\infty} \frac{n+1}{n^2\log n} = \frac{1}{n\log n} = 0$

When comparing $n^2\log(n)$ with 2^n, I can use n instead of $\log(n)$ (n is bigger than $\log(n)$ and show that even n^3 is smaller than 2^n.

n^3, 2^n we can use L'hospital Rule multiple times.

$\log(n^3) = 3\log n$ or; $\lim_{n\to\infty} \frac{n^3}{2^n} \Rightarrow \frac{3n^2}{\ln 2.2^n} \rightarrow \frac{6n}{(\ln 2)^2 2^n} \rightarrow \frac{6}{(\ln 2)^3.2^n} = 0$

$\log(2^n) = n\log 2$

$\lim_{n\to\infty} \frac{3\log n}{n\log 2} = 0$

$\lim_{n\to\infty} \frac{2^n}{10^n} = \frac{\cancel{2^n}}{5^n.\cancel{2^n}} = 0$ | $\lim_{n\to\infty} \frac{10^n}{n!}$ we can use Striling's formula here;

$n! \cong \sqrt{2\pi n} . \left(\frac{n}{e}\right)^n$ therefore $\lim_{n\to\infty} \frac{10^n}{\sqrt{2\pi n}\left(\frac{n}{e}\right)^n} = \frac{\left(\frac{10.e}{n}\right)^n}{\sqrt{2.\pi n}} = 0$

$\lim_{n\to\infty} \frac{n!}{n^{2^n}} = \frac{\sqrt{2\pi n}.\left(\frac{n}{e}\right)^n}{n^n.n^n} = \frac{\sqrt{2\pi n}}{n^n.n^n.\left(\frac{e}{n}\right)^n} = \frac{\sqrt{2\pi n}}{n^n.e^n} = 0$

I compared every consecutive functions in the order to prove my answer.

3. If the height of the tree is n, we can have at least n nodes and at most 2^n nodes.
Time complexities of the following questions are calculated according to this condition.

a) Merging with anothes BST of height n.
Put the keys in the first BST to a list by using In-order traversal so that you get a sorted list.
Put the keys in the second BST to a list using in order traversal so that you get another sorted list.
Merge the lists
Repeat the rest of the processes recursively until the beginning of the list is not smaller than the end of the list
Divide the list from the middle part.
Insert the middle element of the list into the tree.
Call the function again for the left part of the list.
Call the function again for the right part of the list.

Algorithm Analysis:
Best Case → $\Theta(n)$ (tree looks like a linked list)
Worst Case → $\Theta(2^n)$ (tree is a balanced tree)
Average Case → $O(2^n)$

b) Finding the kth smallest element in the BST.
If the tree is empty, return null.
Get the size by size of the left sub-tree +1 (go until the leaf node recursively)
Compare size with k:
  If the size is equal to k, return the key.
  If the size is smaller than k, get the right sub-tree and call the function again.
  If the size is bigger than k, get the left sub-tree and call the function again.

Algorithm Analysis:

Best Case → $\Theta(n)$ (the first key we look for is the kth smallest element in BST)

Wors Case → $\Theta(n^2)$ (the last key we look for is the kth smalles elemen in BST) Getting size is n time

Average Case → $O(n^2)$

c) Balancing the BST

Put the keys in the BST to a list using in-order traversal so that you get a sorted list.

Repeat the rest of the processes recursively until the beginning of the list is not smaller than the end of the list.

Divide the list from the middle part.

Insert the middle element of the list into the tree.

Call the function again for the left part of the list.

Call the function again for the right part of the list.

Algorithm Analysis:

Best Case → $\Theta(n)$ (tree looks like a linked list)

Worst Case → $\Theta(2^n)$. (tree is a balanced tree)

Average Case → $O(2^n)$

d) Finding the elements within a specified value range.

Start from the root node.

If the key is bigger than the min value, call the helper function for the left node.

If the key is smalle than the max value, call the helper function for the right node.

If the key is between the min and the max value, inclusive, add the key to the result array.

Return result array.

(Helper function is used in the if statements, and these if statements are seperate. In one flow of a program, multiple if statement can be executed.

Algorithm Analysis: Best Case → $\Theta(1)$ Worst Case → $\Theta(2^n)$ Average Case → $O(2^n)$

asked value is the root node's key

we need to get every key.

4.

```
i=2
while i<=n:
    if i%2 !=0:
        i= i-1
    else:
        i= i*i
        i= i+1
    print(i)
```

Algorithm Analysis:

It takes $\Theta(\log n)$ because:

When i is even, the loops takes the square of i and increases it by one so that it becomes odd. And when i is odd, the loop decreases it by one so that it becomes even. Basically it takes the square of i, increases and decreases by one every time. Which takes 3* logn time and it makes the time complexity $\Theta(\log n)$

5. Algorithm:

Go over the loop of the given array:

For every number, check if it's mod 2==0

if so, return the number.

Algorithm Analysis:

Best Case → $\Theta(1)$ (we find the first even number in the first index of the array)

Worst case → $\Theta(n)$ (we find the first even number after checking all the odd numbers Which takes 4n/5

Average Case → $\Theta(1)$ (on average, we find the first even number after on average five comparisons acording to the given probability distribution)