1) The limit approach is a method to determine the relationship between two functions in terms of their growth rates. The limit of their ratio is taken as n approaches infinity.

* Big O notation describes an upper bound on the growth rate of a function. If f(n) grows slower than g(n), then $f(n) = O(g(n))$. In addition to that, if the limit of the ratio of f(n) and g(n) as n approaches infinity is 0, it means that f(n) grows slower than g(n), which also proves $f(n) = O(g(n))$. (Explanation 1)

* Big Omega notation describes a lower bound on the growth rate of a function. If f(n) grows faster than g(n), then $f(n) = \Omega(g(n))$. In addition to that, if the limit of the ratio of f(n) and g(n) as n approaches infinity is infinity, it means that f(n) grows faster than g(n), which also proves $f(n) = \Omega(g(n))$. (Exp. 2)

* Big Theta notation describes a tight bound on the growth rate of a function. If f(n) and g(n) grow at the same rate, then $f(n) = \Theta(g(n))$. In addition to that, if the limit of the ratio of f(n) and g(n) as n approaches infinity is a constant c where $0 < c < \infty$, it means that f(n) and g(n) grow at the same rate, which also proves $f(n) = \Theta(g(n))$. (Explanation 3)

a) $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{n^2+7n}{n^3+7}$. Since $\frac{n^2+7n}{n^3+7}$ is equal to ∞/∞ as n approaches infinity, L'Hopital's rule will be used. Since $\frac{2n+7}{3n^2}$ is equal to ∞/∞ as n approaches infinity, L'Hopital's rule will be applied again. $\lim_{n\to\infty} \frac{2}{6n} = 0$

Therefore, $\boxed{f(n) = O(g(n))}$. Explanation 1 explains in a detailed way.

b) $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{12n + \log_2(n^2)}{n^2+6n}$. Since f(n)/g(n) is equal to ∞/∞ as n approaches infinity, L'Hopital's rule will be applied. $\lim_{n\to\infty} \frac{12 + \frac{2}{\ln(2).n}}{2n+6} = 0$

Therefore, $\boxed{f(n) = O(g(n))}$. Explanation 1 explains in a detailed way.

c) $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{n.\log_2 3n}{n + \log_2(8n^3)}$. Since f(n)/g(n) is equal to ∞/∞ as n approaches infinity, L'Hopital's rule will be applied. $\lim_{n\to\infty} \frac{\log_2 3n + \frac{1}{\ln 2}}{1 + \frac{3}{n.\ln 2}} = \infty$

Therefore, $\boxed{f(n) = \Omega(g(n))}$. Explanation 2 explains in a detailed way.

d) $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{n^n + 5n}{3.2^n}$. Since n^n grows faster than 2^n as n approaches infinity, $\lim_{n\to\infty} \frac{n^n+5n}{3.2^n} = \infty$. Therefore, $\boxed{f(n) = \Omega(g(n))}$.

Explanation 2 explains in a detailed way.

e) $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{\sqrt[3]{2n}}{\sqrt{3n}}$. $\frac{f(n)}{g(n)} = \frac{2^{1/3} \cdot n^{1/3}}{3^{1/2} \cdot n^{1/2}} = \frac{2^{1/3} \cdot n^{-1/6}}{3^{1/2}}$

$\lim_{n\to\infty} \frac{2^{1/3} \cdot n^{-1/6}}{3^{1/2}} = 0$. Therefore, $\boxed{f(n) = O(g(n))}$. Explanation 1 explains in a detailed way.

---

2) a) The overall time complexity of methodA is determined by the number of iterations of the loop, which equals to n because there is one for loop which will execute System.out.println(names[i]); statement n times. Therefore, the worst-case time complexity of methodA is O(n).

b) The overall time complexity of methodB is determined by the number of iterations of loops, which equals to 3n because the length of myArray is 3. methodA is going to be called 3 times. methodA has a for loop iterating n times. The worst-case time complexity of methodA is O(n). Therefore, The worst-case time complexity of methodB is O(3n) = O(n).

c) The condition i < numbers.length will always be true. Therefore, the loop is infinite and will never terminate. Therefore, the worst-case time complexity of methodC is undefined.

d) The overall time complexity of methodD is determined by the number of iterations of the loop. In the worst case, all elements of the array named numbers are less than 4. Therefore, the loop will iterate n times. System.out.println(numbers[i++]); statement will be executed n times. The worst-case time complexity of methodD is O(n). However, in the worst case, the loop will terminate and ArrayIndexOutOfBoundsException will be thrown after the execution of System.out.println(numbers[i++]); statement n times. If the error is properly handled, then the worst-case time complexity of the method might be still acceptable. However, if the error is not handled correctly and causes the program to crash, the worst-case time complexity becomes irrelevant since the program will not produce any result or output.

3) If we assume that myArray.length is equal to n, both methods execute a statement n times. Therefore, the time complexity of both methods is the same and equals to $O(n)$. There is no differance between the time complexities of withoutLoop and witLoop methods. The more advantageous method varies depending on the context. For example, The witLoop method is more concise, flexible, easier to read, understand and maintain, especially for the arrays that are large. In addition to that, it is easier to handle errors in witLoop method, compared to withoutLoop method. The witLoop method can handle arrays of any length. When it comes to large arrays, using witLoop method is preferable to using withoutLoop method because of the advantages written above. Therefore, witLoop method is generally more advantageous. However, if you need to perform a specific task for a small number of elements in the array, using a loop might be unnecessary and therefore, using withoutLoop method might be more advantageous.

4) This problem cannot be solved in constant time. Since it is possible that the specific integer might be at any position in the array, in the worst case scenario, all elements of the array need to be examined to determine if the specific integer is present in the array or not. Therefore, considering the worst case scenario, the time complexity of the algorithm is $O(n)$, where n is the number of integers in the array, which proves that the problem cannot be solved in constant time $O(1)$.

5) We need to find the minimum value in array A and array B in order to find the minimum value of $a_i \cdot b_j$ where $0 \leq i < n$ and $0 \leq j < m$. Briefly, my linear time algorithm will find the minimum value in array A and array B and return the product of those minimum values. Here is the pseudo-code:

```
function findMinimumProduct(A,B):
minArrA = A[0]
minArrB = B[0]
for i in range [0,n) do
    if A[i] < minArrA then
        minArrA = A[i]
    end if
end for
for j in range [0,m) do
    if B[j] < minArrB then
        minArrB = B[j]
    end if
end for
return (minArrA * minArrB)
end function
```

The minimum value in array A and array B are found to find the minimum value of $a_i \cdot b_j$ and the product of the minimum values is returned from the function.

In the worst case scenario, all elements of the two arrays need to be examined, which means that the first for loop will iterate n times, the second for loop will iterate m times. Therefore, the worst-case time complexity of the algorithm is $O(n+m)$. It is a linear time algorithm.