CSE222 - Homework 2

1. To solve this question, the limit method will be implemented as seen below.
   PS: You cannot get full credit, if you didn't solve indeterminations like $\frac{\infty}{\infty}$.

$$\boxed{\begin{array}{l} f(n) = O(g(n)) \implies \lim\limits_{n\to\infty} \dfrac{f(n)}{g(n)} = 0 \\ f(n) = \Omega(g(n)) \implies \lim\limits_{n\to\infty} \dfrac{f(n)}{g(n)} = \infty \\ f(n) = \Theta(g(n)) \implies 0 < \lim\limits_{n\to\infty} \dfrac{f(n)}{g(n)} < \infty \end{array}}$$

   (a) $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{n^2+7n}{n^3+7} = \frac{\infty}{\infty} \xrightarrow{L'hospital} \lim_{n\to\infty} \frac{2n+7}{3n^2} = \frac{\infty}{\infty} = \xrightarrow{L'hospital}$
   $\lim_{n\to\infty} \frac{2}{6n} = \frac{0}{\infty} = 0$

   Thus, $f(n) = O(g(n))$.

   (b) $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{12n+log_2 n^2}{n^2+6n} = \frac{\infty}{\infty} \xrightarrow{L'hospital} \lim_{n\to\infty} \frac{12+\frac{2}{n*ln(2)}}{2n+6} = \lim_{n\to\infty} \frac{12+0}{2n+6} = \frac{0}{\infty} = 0$

   Thus, $f(n) = O(g(n))$.

   (c) $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{n*log_2 3n}{n+log_2(8n^3)} = \lim_{n\to\infty} \frac{n*log_2 3n}{n+3*log_2 2n} = \frac{\infty}{\infty} \xrightarrow{L'hospital} \lim_{n\to\infty} \frac{log_2 3n+\frac{1}{ln2}}{1+\frac{3}{n*ln(2)})} =$
   $\frac{\infty+\frac{1}{\infty}}{1+\frac{3}{\infty}} = \frac{\infty+0}{1+0} = \frac{\infty}{1} = \infty$

   Thus, $f(n) = \Omega(g(n))$.

   (d) $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{n^n+5n}{3*2^n} = \lim_{n\to\infty} \frac{n^n}{3*2^n} + \lim_{n\to\infty} \frac{5n}{3*2^n} = \frac{1}{3} * \lim_{n\to\infty} (\frac{n}{2})^n + \lim_{n\to\infty} \frac{5n}{3*2^n}$

   $= \frac{1}{3} * \infty + \lim_{n\to\infty} \frac{5n}{3*2^n} = \infty + \frac{\infty}{\infty} \xrightarrow{L'hospital} \infty + \lim_{n\to\infty} \frac{5}{3n*2^{n-1}} = \infty + \frac{5}{\infty} = \infty + 0 = \infty$

   Thus, $f(n) = \Omega(g(n))$.

   (e) $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{\sqrt[3]{2n}}{\sqrt{3n}} = \frac{\sqrt[3]{2}}{\sqrt{3}} * \lim_{n\to\infty} \frac{n^{\frac{1}{3}}}{n^{\frac{1}{2}}} = \frac{\sqrt[3]{2}}{\sqrt{3}} * \lim_{n\to\infty} n^{\frac{1}{3}-\frac{1}{2}} = \frac{\sqrt[3]{2}}{\sqrt{3}} * \lim_{n\to\infty} n^{-\frac{1}{6}} =$
   $\frac{\sqrt[3]{2}}{\sqrt{3}} * \lim_{n\to\infty} \frac{1}{n^{\frac{1}{6}}} = \frac{\sqrt[3]{2}}{\sqrt{3}} * \frac{1}{\infty} = \frac{\sqrt[3]{2}}{\sqrt{3}} * 0 = 0$

   Thus, $f(n) = O(g(n))$.

2. (a) Since *methodA* prints each element in an array and the length of the array is n, the complexity is $O(n)$.

   (b) Similarly,*methodB* has a loop with n iterations (n is the length of the array). But at each iteration, *methodB* is being called and as we saw, its complexity is $O(n)$. Therefore, the complexity is $O(n * n) = O(n^2)$.

   (c) *MethodC* doesn't terminate due to the lack of increment operation of the counter within the loop. Thus, we cannot talk about complexity here.

   (d) *MethodD* has a loop with uncertain iterations. According to the values of the elements of the array, the loop might not run at all, or it might run n times (since there are n elements in the array). However, we are asked to analyze the worst-case time complexity. So, the answer is $O(n)$.

   But, there is no condition for the array size. Therefore the program might raise an exception and stop running due to reaching an element with the index of n in an array with n elements ($0 \leq index \leq n - 1$). In such a case, we cannot talk about the complexity as in *methodC*.

3. In terms of time complexity, there is no difference between the two methods. A loop is not the necessity of $O(n)$ time complexity. If we are repeating an operation with $O(1)$ time complexity n times, it will end up with a time complexity of $n * O(1) = O(n)$. So, both of the methods have a time complexity of $O(n)$ (due to the n operations, n is the length of the array). But if we are to choose one, of course, loops are easier to code/read.

4. Quick answer: no. Because if we don't know whether the array is sorted or not, we might have to check every element in the array which ends up with a time complexity of $O(n)$.

   (Yes, it is possible to find it at once if you are lucky, but the question asks you to consider all possible inputs.)

5. In this question using nested loops causes a quadratic time complexity ($n*m$). To understand better, why this is not linear, assume $n = m$, then the complexity would be $O(n*m) = O(n*n) = O(n^2)$. To prevent this, we can calculate the minimum and maximum elements of both arrays by using two consecutive loops (maximum is also required because some of the array elements might be negative). After that, we can compare the multiplication of min/max elements of both arrays and return the minimum value. The pseudo-code of this solution is given on the next page. As seen, we don't use nested loops. The first loop has n iterations and the second one has m iterations. Thus, the larger one among these numbers dominates the algorithm and determines the complexity. In other terms, the complexity will be $O(n) + O(m) + k * O(1)$ (k is the number of constant time operations, in the pseudo-code given below, it is 13, including if conditions). As known, if $n > m$, we can ignore $O(m)$ and vice versa. Also, $k * O(1)$ will be ignored as well, since k is independent from the input size. In conclusion, the time complexity of this algorithm is $O(max(n, m))$.

Algorithm 1: Find the minimum value of $a_i * b_j$

```
Inputs: A, B
initialization;
minA ← A[0]
minB ← B[0]
maxA ← A[0]
maxB ← B[0]
result ← A[0]*B[0]
for each element of A do
    if element < minA then
        minA ← element
    end
    if element > maxA then
        maxA ← element
    end
end
for each element of B do
    if element < minB then
        minB ← element
    end
    if element > maxB then
        maxB ← element
    end
end
if minA*minB < result then
    result ← minA*minB
end
if minA*maxB < result then
    result ← minA*maxB
end
if maxA*minB < result then
    result ← maxA*minB
end
if maxA*maxB < result then
    result ← maxA*maxB
end
return result
```