Exercises: Algorithm Efficiency and Big O Notation - Solutions

Exercise 1

Order the following by their $\mathcal{O}(\cdot)$ ranking - here from slowest to fastest

n^n
$n!$
2^n
n^2
$\dfrac{n^2}{\log n}$
$n\sqrt{n}$
$n\log n$
n
$\log n$
$\log\log n$

Exercise 2

Which of the following are true?

a. $4n^2 = \mathcal{O}\left(n^2\right) \longrightarrow T$

b. $4n^2 + 18n\log n = \mathcal{O}\left(n^2\right) \longrightarrow T$

c. $4n^2 + 18n\log n = \mathcal{O}\left(n\right) \longrightarrow F$

d. $4n^2 + 18n\log n = \mathcal{O}\left(n\log n\right) \longrightarrow F$

e. $4n + 18n\log n = \mathcal{O}\left(n\log n\right) \longrightarrow T$

f. $4n + 18n\log n = \mathcal{O}\left(n^2\right) \longrightarrow T$

Exercise 3

The following code snippet calculates x^n. We are given two integers, and the algorithm returns an integer. Find the time complexity of the algorithm.

```
// Algorithm1 - Iterative
public static long power1(int x, int n)
{
    // initialize result by 1
    long pow = 1L;

    // multiply x exactly n times
    for (int i = 0; i < n; i++) {
        pow = pow * x;
    }
    return pow;
}
```

The time complexity of Algorithm1 is $\mathcal{O}(n)$.

Exercise 4

Below you see some Python code to do some operation:

```python
for i in range(len(A)):
    for j in range(len(B[0])):
        for k in range(len(B)):
            C[i][j] += A[i][k] * B[k][j]

for r in C:
    print(r)
```

A and B are two matrices and C is another empty matrix (actually it just has zeroes on all entries). A matrix is simply a 2d-array, i.e. A, B, and C are `int[][]`:

a. Figure out what exactly the code does. **Matrix multiplication**

b. What is the time complexity of the algorithm? $\mathcal{O}(n^3)$.

c. Implement the algorithm in Java

```java
// Assume initialisation has been done:
// int C[][] = new int[A.length][B[0].length];

// Core logic for multiplying two matrices
for (int i = 0; i < A.length; i++) {
    for (int j = 0; j < B[0].length; j++) {
        for (int k = 0; k < B.length; k++) {
            C[i][j]+= A[i][k] * B[k][j];
        }
    }
}
return C;
```

Exercise 5

Exercise 2.7 from the course book.

For each of the following six program fragments:

a. Give an analysis of the running time (Big-Oh will do). (see below for solution)

b. Implement the code in Java, and give the running time for several values of n. (no solution, just do it!)

(1)
```
sum = 0;
for( i = 0; i < n; i++ )
    sum++;
```

(2)
```
sum = 0;
for( i = 0; i < n; i++ )
    for( j = 0; j < n; j++ )
        sum++;
```

(3)
```
sum = 0;
for( i = 0; i < n; i++ )
    for( j = 0; j < n * n; j++ )
        sum++;
```

(4)
```
sum = 0;
for( i = 0; i < n; i++ )
    for( j = 0; j < i; j++ )
        sum++;
```

(5)
```
sum = 0;
for( i = 0; i < n; i++ )
    for( j = 0; j < i * i; j++ )
        for( k = 0; k < j; k++ )
            sum++;
```

(6)
```
sum = 0;
for( i = 1; i < n; i++ )
    for( j = 1; j < i * i; j++ )
       if( j % i == 0 )
           for( k = 0; k < j; k++ )
               sum++;
```

1. $\mathcal{O}(n)$.
2. $\mathcal{O}(n^2)$
3. $\mathcal{O}(n^3)$
4. $\mathcal{O}(n^2)$
5. $\mathcal{O}(n^5)$: j can be as large as i^2, which could be as large as n^2, and k can be as large as j, which is n^2. The running time is thus proportional to $n \cdot n^2 n^2$, which is $\mathcal{O}(n^5)$
6. The if statement is executed at most n^3 times, by previous arguments (answer to 5), but it is true only $O\left(n^2\right)$ times (because it is true exactly i times for each i). Thus the innermost loop is only executed $O\left(n^2\right)$ times. Each time through, it takes $O\left(j^2\right) = O\left(n^2\right)$ time, for a total of $O\left(n^4\right)$. This is an example where multiplying loop sizes can occasionally give an overestimate.