CSE 203

Class Work on Time complexity and Big-Oh notation

Date: 15/5/19

Problem 1:

Assume that each of the expressions below gives the processing time $T(n)$ spent by an algorithm for solving a problem of size n. Select the dominant term(s) having the steepest increase in n and specify the lowest Big-Oh complexity of each algorithm.

Expression	Dominant term(s)	$O(\ldots)$
$5 + 0.001n^3 + 0.025n$		
$500n + 100n^{1.5} + 50n \log_{10} n$		
$0.3n + 5n^{1.5} + 2.5 \cdot n^{1.75}$		
$n^2 \log_2 n + n(\log_2 n)^2$		
$n \log_3 n + n \log_2 n$		
$3 \log_8 n + \log_2 \log_2 \log_2 n$		
$100n + 0.01n^2$		
$0.01n + 100n^2$		
$2n + n^{0.5} + 0.5n^{1.25}$		
$0.01n \log_2 n + n(\log_2 n)^2$		
$100n \log_3 n + n^3 + 100n$		
$0.003 \log_4 n + \log_2 \log_2 n$		

Problem 2:

The statements below show some features of "Big-Oh" notation for the functions $f \equiv f(n)$ and $g \equiv g(n)$. Determine whether each statement is TRUE or FALSE and correct the formula in the latter case.

Statement	Is it TRUE or FALSE?	If it is FALSE then write the correct formula
Rule of sums: $O(f+g) = O(f) + O(g)$		
Rule of products: $O(f \cdot g) = O(f) \cdot O(g)$		
Transitivity: if $g = O(f)$ and $h = O(f)$ then $g = O(h)$		
$5n + 8n^2 + 100n^3 = O(n^4)$		
$5n+8n^2+100n^3 = O(n^2 \log n)$		

Problem 3:

Algorithms **A** and **B** spend exactly $T_A(n) = 0.1n^2 \log_{10} n$ and $T_B(n) = 2.5n^2$ microseconds, respectively, for a problem of size n. Choose the algorithm, which is better in the Big-Oh sense, and find out a problem size n_0 such that for any larger size $n > n_0$ the chosen algorithm outperforms the other. If your problems are of the size $n \leq 10^9$, which algorithm will you recommend to use?

Problem 4:

Assume that the array a contains n values, that the method `randomValue` takes constant number c of computational steps to produce each output value, and that the method `goodSort` takes $n \log n$ computational steps to sort the array. Determine the Big-Oh complexity for the following fragments of code taking into account only the above computational steps:

```
for( i = 0; i < n; i++ ) {
  for( j = 0; j < n; j++ )
    a[ j ] = randomValue( i );
  goodSort( a );
}
```

Problem 5:

Work out the computational complexity (in the "Big-Oh" sense) of the following piece of code and explain how you derived it using the basic features of the "Big-Oh" notation:

```
for( int bound = 1; bound <= n; bound *= 2 ) {
   for( int i = 0; i < bound; i++ ) {
      for( int j = 0; j < n; j += 2 ) {
         ... // constant number of operations
      }
      for( int j = 1; j < n; j *= 2 ) {
         ... // constant number of operations
      }
   }
}
```