Data Structures & Algorithms 2
Tutorial 2
Algorithm Analysis -part1

OBJECTIVE

Compute the computational complexity for an algorithm

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

Exercise 2

Order the following functions by growth rate:

N, $\sqrt{N}$, $N^{1.5}$, N^2, $N \log N$, $N \log \log N$, $N \log^2 N$, $N \log(N^2)$, $2/N$, 2^N, $2^{N/2}$, 37, $N^2 \log N$, N^3.

Indicate which functions grow at the same rate. Give insights of how you solved this?

Exercise 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 $n > 10^9$, which algorithm will you recommend to use?

Exercise 4

Prove that $T(n) = a_0 + a_1 n + a_2 n^2 + a_3 n^3$ is $O(n^3)$ using the formal definition of the Big-Oh notation.
Hint: Find a constant C and threshold n_0 such that $cn^3 \geq T(n)$ for $n \geq n_0$.

Exercise 5

Assuming that $f_1(n)$ is $O(g_1(n))$ and $f_2(n)$ is $O(g_2(n))$, Disprove the following statements by providing counterexamples:

a. $f_1(n) - f_2(n)$ is $O(g_1(n) - g_2(n))$.

b. $f_1(n)/f_2(n)$ is $O(g_1(n)/g_2(n))$.

Exercise 6

a)

```
for (cnt3 = 0, i = 1; i <= n; i *= 2)
        for (j = 1; j <= n; j++)
                cnt3++;
```

b)

```
for (cnt4 = 0, i = 1; i <= n; i *= 2)
        for (j = 1; j <= i; j++)
                cnt4++;
```

c)

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

d)

```
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
      }
   }
}
```

e)

```
int fct2 (int n, int m) {
        if (n < 10) return n;
                else if (n < 100)
                        return fct2 (n - 2, m);
                   else
                        return fct2 (n/2, m);
}
```

Additional exercises

Exercise 7

Find the complexity of the function used to find the kth smallest integer in an unordered array of integers:

```
int selectkth(int a[], int k, int n) {
int i, j, mini, tmp;
for (i = 0; i < k; i++) {
          mini = i;
          for (j = i+1; j < n; j++)
              if (a[j]<a[mini])
                      mini = j;
                      tmp = a[i];
                     a[i] = a[mini];
                      a[mini] = tmp;
                 }
         return a[k-1];
}
```

Exercise 8

Write the fast exponentiation routine without recursion using the squaring method.
Estimate the complexity of the algorithm.