Homework 1. Algorithm Analysis I: Solution

Prof. Karpenko

(12 pts)

Your Name:

1. **(4 pt)** Order the functions in increasing order of growth rate (from slowest growing to fastest). **You need to explain your answers in detail to get full credit for this question.** When two adjacent functions have the same growth rate, note that by underlining them. You can assume that the base of the log is 2:
   $10n$, $2^{\frac{n}{2}}$, $\lg(n^5)$, $\lg\lg n$, $n^3 + n^2 + n\lg n + 2^{10}$, $2^{\lg n}$, $\sqrt{n}$, 10^5, $n^{\frac{1}{4}}$, $n\lg n$

   Solution: The following functions are ordered in the increasing order of growth:
   10^5, $\lg\lg n$, $\lg(n^5)$, $n^{\frac{1}{4}}$, $\sqrt{n}$, $2^{\lg n}$, $10n$, $n\lg n$, $n^3 + n^2 + n\lg n + 2^{10}$, $2^{\frac{n}{2}}$.

   Justification:
   10^5 is a constant, the slowest growing function.
   $\lg(n^5) = 5\lg(n) = \Theta(\lg(n))$, so it grows faster than $\lg\lg n$.

   Note that $2^{\lg n}$ can be rewritten as $2^{\lg n} = n$. $\sqrt{n} = n^{\frac{1}{2}}$, and it grows faster than $n^{\frac{1}{4}}$, but slower than $2^{\lg n} = n$, because they are all polynomial functions, so we can just compare their exponents.

   $n^3 + n^2 + n\lg n + 2^{10} = \Theta(n^3)$ since n^3 is the fastest growing term. $n^3 = n * n^2$ grows faster than $n\lg n$, because n^2 grows faster than $\lg n$.

   Finally, $2^{\frac{n}{2}}$ is an exponential function that grows faster than any polynomial, including n^3.

2. Use the formal definitions of Θ, O and Ω to prove the following:

   (a) **(1.5 pt)** f(n) = n^2 + 2n + 5 $\in \Theta(n^2)$
   Solution: Let us first prove that $f(n) = O(n^2)$. We need to prove that there exist some positive constants C_1 and N such that $f(n) <= C_1 n^2$ for $n > N$. $f(n) = n^2 + 2n + 5 <= n^2 + 2n^2 + 5n^2 = 8n^2$, so we found $C_1 = 8$, $N = 1$.
   Let us now prove that $f(n) = \Omega(n^2)$. We need to prove that there exist some positive constants C_2 and N such that $f(n) >= C_2 n^2$ for $n > N$. We can write that $f(n) = n^2 + 2n + 5 >= n^2$ (we will make everything smaller by throwing away some positive terms), hence we can take $C_2 = 1$, $N = 1$ to satisfy the definition of the Omega. Since $f(n) \in O(n^2)$ and $f(n) \in \Omega(n^2)$, $f(n) \in \Theta(n^2)$.

   (b) **(1.5 pt)** f(n) = $\sum_{i=1}^{n}((i-1) * i) \in \Theta(n^3)$
   For (b), you will need a formula for the sum of the squares of the first n positive integers:

   http://www.9math.com/book/sum-squares-first-n-natural-numbers

Solution: $\sum_{i=1}^{n}((i-1)*i) = \sum_{i=1}^{n} i^2 - \sum_{i=1}^{n} i$. The first term is the sum of squares of the first n positive integers - the link in the homework provides a formula for this summation: $\frac{n(n+1)(2n+1)}{6}$. The second term is arithmetic series, with the closed form solution of $\frac{n(n+1)}{2}$. We can then rewrite the expression for f(n): $f(n) = \frac{n(n+1)(2n+1)}{6} - \frac{n(n+1)}{2} = \frac{n(n+1)(2n+1)-3n(n+1)}{6} = \frac{2n^3+2n^2+n^2+n-3n^2-3n}{6} = \frac{2n^3-2n}{6} = \frac{n^3-n}{3}$.
It is easy to see that $f(n) = \frac{n^3-n}{3} <= \frac{1}{3}n^3$ (because we will make everything larger if we get rid of the negative term) and hence, we can pick $C_1 = 1/3$, $N = 1$ and according to the definition of the big O, $f(n) \in O(n^3)$.
Let us now prove that $f(n) \in \Omega(n^3)$. First, note that $\frac{n}{3} = \frac{2n}{6} < \frac{n^3}{6}$ for $n >= 2$ (because if we get rid of n in the numerator of both terms, we will be comparing 2 and n^2 and $2 < n^2$ for $n >= 2$). Then we can write that $\frac{n^3-n}{3} = \frac{n^3}{3} - \frac{n}{3} >= \frac{n^3}{3} - \frac{n^3}{6}$ (we will make everything smaller by subtracting a larger term) $= \frac{n^3}{6}$. So we can pick $C_2 = \frac{1}{6}$, and conclude that $f(n) \in \Omega(n^3)$. Since $f(n) \in O(n^3)$ and $f(n) \in \Omega(n^3)$, $f(n) = \Theta(n^3)$.

3. Give $\Theta()$ running times for each of the following functions, as a function of n. Provide an **explanation** for your answers. Note that when an inner loop variable depends on the outerloop variable, you need to do a summation to get the Θ.

   (a) **(1.5 pt)**

```
public static int func1(int n) {
   int res = 1;
   for (int i = 1 ; i <= n; i++) {
      for (int j = 2*n; j >= 1; j--) {
         res *= j*j;
      }
      for (int k = 1; k <= n/2; k++) {
         res *= k;
      }
      for (int m = 1; m <= n*n; m++) {
         res *= m;
      }
   }
   return res;
}
```

   Solution The outer loop has n iterations. The loops inside it are executed sequentially, the j loop has $2*n$ iterations, the k loop has $n/2$ iterations. and the m loop has n^2 iterations.Out of three inner loops, the loop that has the largest running time is n^2. The total running time is n^3.

   (b) **(1.5 pt)** Note that i is *multiplied* by 2 after each iteration, and that i goes to $4n$, not to n.

```
public static int func2(int n) {
   int sum = 0;
   for (int i = 1; i <= 4n; i *= 2)
         sum++;
   return sum;
}
```

   Solution i is multiplied by 2 at each iteration. $2^k = 4n$. $k = \lg(4n) = \lg 4 + \lg n = \Theta(\lg n)$.

(c) **(2 pt)** Note that in the j loop, j goes to n^3, and that the number of iterators of the innermost loop depends on j.

```
public static int func3(int n) {
   int sum = 0;
   for ( int i = 0 ; i < n ; i++)
       for (int  j = 0 ; j < n*n*n; j++)
            for( int k = 0; k < j ; k++)
                 sum++;
   return sum;
}
```

Solution Because k depends on j, we need to compute the running time of the j and k loops together and do a summation: $f(n) = 0 + 1 + 2 + \cdot + n^3 = \frac{n^3 * (n^3 + 1)}{2} = \Theta(n^6)$. The outermost loop has n iterations, so we multiple the running time of the j and k loops by n to get the total running time of $\Theta(n^7)$.