Problem Set 1

Assigned: May 25
Due: June 1

1. For each of the following pairs of functions $f(n)$ and $g(n)$, state whether $f$ is $O(g)$; whether $f$ is $o(g)$; whether $f$ is $\theta(g)$; and whether $f$ is $\Omega(g)$. (More than one of these can be true for a single pair.)

   a. $f(n) = n^{10}; g(n) = 2^{n/2}$.
   b. $f(n) = n^{3/2}; g(n) = n \lg^2(n)$.
   c. $f(n) = \lg(n^3); g(n) = \lg(n)$.
   d. $f(n) = 2^n; g(n) = 2^{n/2}$.
   e. $f(n) = n^2; g(n) = (n/2)^2$.

2. List the following functions in increasing order of growth. If two functions have the same order of growth, state the fact.

   $$(\lg(n))^2, \ (2^n), \ (\lg(n))!, \ 2^{(n^2)}, \ n^n, \ n \ lg(n), \ n! \ lg(2^n), \ lg(n^2), \ n \ (2^n), \ lg(lg(n))$$

3. The following three functions calculate $k^n$, for integer $k$ and $n$. Give the asymptotic running time of each. Assume that arithmetic operations take unit time.

   int exp1(k,n)
   {
      power = 1;
      for (i = 1 to n) {
         newpower = 0;
         for (j = 1 to k) {
            newpower = newpower + power;
         }
         power = newpower;
      }
      return(power)
   }

   int exp2(k,n)
   {
      power = 1;
      for (i = 1 to n) power := power * k;
      return(power)
   }
/* exp3 (k,n) recursively computes k**((n/2), then squares. */
int exp3(k, n)
{
    if (n == 0) return(1)
    else if (n == 1) then return(k)
    else {
        hpower := exp3(k, floor(n/2));
        if (even(n)) return(hpower*hpower)
        else return(hpower * hpower * k)
    }
}

4. The function “element” below checks whether integer I is element of list L. The function “subset” below checks whether list L is a subset of list M; both are lists of integers. Give the asymptotic worst-case running time of element and subset. When is this worst case achieved?

bool element(X : int; Q : intlist)
bool found = false; /* Flag stating whether X has been found */
{
    while (Q != NULL && !found)
    {
        found = (Q->value == X);
        Q := Q->next;
    }
    return(found)
}

bool subset(L, M : intlist)
bool success = true; /* Flag whether L is a subset so far */
{
    while ((L != NULL) && success)
    {
        success := element(L->value, M);
        L := L->next;
    }
    return(success)
}