Problem Set 1

Assigned: May 29
Due: June 5

Problem 1
For each of the following pairs of functions $f(n)$ and $g(n)$, state whether $f$ is $o(g)$, $f$ is $\Theta(g)$; or $g$ is $o(f)$. (Exactly one of these is true in all cases.)

a. $f(n) = n^{10}$; $g(n) = 2^{n/2}$.

b. $f(n) = n^{3/2}$; $g(n) = n \log^2(n)$.

c. $f(n) = \log(n^3)$; $g(n) = \log(n)$.

d. $f(n) = \log(3^n)$; $g(n) = \log(2^n)$.

e. $f(n) = (\log(n))^3$; $g(n) = (\log(n))$.

f. $f(n) = 2^n$; $g(n) = 2^{n/2}$.

g. $f(n) = n^2$; $g(n) = (n/2)^2$.

h. $f(n) = n^2$; $g(n) = (n + 2)^2$.

i. $f(n) = 2^n$; $g(n) = 2^{n+2}$

j. $f(n) = n!$; $g(n) = (n + 2)!$

Problem 2
The following two functions both compute the same thing. They take as arguments two arrays $A$ and $B$ and return TRUE if every element of $A$ is less than every element of $B$.

```c
AllLessThan1(int[] A, B) return bool {
    for (i = 1 to |A|)
        for (j = 1 to |B|)
            if (A[i] >= B[j]) return FALSE;
    return TRUE;
}

AllLessThan2(int[] A, B) return bool {
    largeA = A[1]
    for (i = 2 to |A|)
        if (A[i] > largeA) largeA = A[i];
    for (j = 1 to |B|)
        if (largeA >= B[j]) return FALSE;
    return TRUE;
}
```

A. Give the asymptotic worst-case running time of each algorithm as a function of $|A|$ and $|B|$. When does the worst case occur?

B. Give the asymptotic best-case running time of each algorithm as a function of $|A|$ and $|B|$. When does the best case occur?
C. For any particular arrays $A, B$, let $t_1(A, B)$ and $t_2(A, B)$ be the running times of the two algorithms for inputs $A, B$. Design an algorithm $\text{AllLessThan3}(A, B)$ that has a running time that is no more than $2 \min(t_1(A, B), t_2(A, B))$ for every input $A, B$.

**Problem 3**

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

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

int exp2(k,n)
{ power = 1;
  for (i=0; i<n; i++) power := power * k;
  return(power)
}

/* exp3 (k,n) recursively computes $k^{\lfloor n/2 \rfloor}$, 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)
  }
}

int exp4(k,n)
{ power = 1;
  for (i=0; i<n; i++) {
    newpower = 0;
    for (j=0; j<k; j++)
      for (q=0; q<power; q++)
        newpower++;
    power = newpower;
  }
  return(power)
}
```

(Hint for $\text{exp4}$: On the $i$th iteration of the main loop, the algorithm is computing $k^i$ by starting at 0 and repeatedly adding 1.)