Problem Set 2

Assigned: June 4
Due: June 11

Problem 1 (1 point)

(Postponed from problem set 1, problem 3)

The following function calculates k^n, for integer k and n. Give its asymptotic running time.

```
// exp3 (k,n) recursively computes k**(n/2), then squares.
int exp3(k,n)
{  if  (n == 0) return(1)
     else if (n == 1) return(k)
     else {
        hpower = exp3(k,floor(n/2));
        if (even(n)) return(hpower*hpower)
           else return(hpower * hpower * k)
       }
}
```

Problem 2 (6 points)

The following recursive algorithms find the biggest element in a segment of an array `a` of integers between indices `p` and `q`. (None of them are particularly good ways of computing the property, and some are terrible, but that's beside the point.) Let $\texttt{n} = \texttt{q} + 1 - \texttt{p}$ be the length of the segment. For each of the following, write down and solve the recurrence equation for the worst-case running time of the algorithm as a function `T(n)`.

In parts A-D, `a` is a global variable. Assume that this is always called with $\texttt{p} \leq \texttt{q}$ and with both parameters in the index range of `a`. Ignore issues of rounding up and down; assume that all divisions come out to exact integers,

In parts E and F, array `a` is a parameter, and the algorithm works on the array as a whole. Succesive recursive calls have smaller and smaller arrays. Assume that `a.length` returns the length of the array.

It is not part of the assignment, but you should be sure that you understand how each of these recursive algorithms works.

In all of these, assume that the function `max(x,y)` is defined as follows:

```
int max(x,y) { if (x > y) return x else return y }
```

A. Compute the biggest element in a[p ... q] by finding the biggest element in a[p..q-1] and taking the bigger of that or a[q]

```
int biggest1(p,q) {
  if (q == p) return a[p];
  else return max(biggest1(p,q-1), a[q]))
}
```

B. Compute the biggest element in a[p .. q] by finding the biggest values in a[p .. q-1] and in a[p+1,q] and taking the larger of those.

```
int biggest2(p,q) {
   if (q <= p) return a[p];
   else return max(biggest2(p,q-1),biggest2(p+1,q))
  }
```

C. Compute the biggest elements in a[p .. q] by finding the biggest value in the first third, the second third, and the last third, and taking the largest of these.

```
int biggest3(p,q) {
   if (q <= p) return a[p];
   else if (p == q+1) return max(a[p],a[q])
   else {
      length = q+1-p;
      h = p + length/3;
      k = p + 2*length/3;
      return max(max(biggest3(p,h),biggest3(h+1,k)),  biggest3(k+1,q))
     }
   }
```

D. Compute the biggest element in a[p .. q] by finding the biggest value in the first two-thirds and the biggest in the second two-thirds, and taking the larger of these.

```
int biggest4(p,q) {
   if (q <= p) return a[p];
   else if (q == p+1) return max(a[p],a[q])
   else {
      length = q+1-p;
      h = p + length/3;
      k = p + 2*length/3;
      return max(biggest4(p,k),biggest4(h+1,q))
     }
   }
```

E. Compute the biggest element in array a by copying the first n-1 elements to array b, recursively finding the biggest element in b, and then returning that value or a[n], whichever is bigger.

```
int biggest5(int[] a) {
  n = a.length;
  if (n == 1) return a[1]
  else {
     int[]  b = new int[1 .. n-1];
     for (i = 1 to n-1)
       b[i] = a[i];
     return max(biggest5(b),a[n]);
     }
}
```

F. Compute the biggest element in array a by copying the first n/2 elements to array b and the last n/2 elements to array c, recursively finding the biggest elements in b and c, and then returning the bigger of those two.

```
int biggest6(int[] a) {
  n = a.length;
  if (n == 1) return a[1]
  else {
     m = n/2;
     int[]  b = new int[1 .. m]
     int[]  c = new int[1 .. n-m]
     for (i = 1 to m;
       b[i] = a[i];
     for (i = 1 to n-m]
       c[i] = a[m+i];
     return max(biggest6(b),biggest6(c))
     }
}
```

Problem 3 (3 points)

Use the method of recurrence trees to solve the following recurrence equations to within a constant factor: Write out an explanation in terms of recurrence trees. Do not use the Master Theorem to get your answer, though of course you may want to use it to check your answer.

$C(1) = 1$.
$C(n) = n + 4C(n/2)$, $n > 1$. Assume n is a power of 2.

$D(1) = 1$.
$D(n) = n^2 + 4D(n/2)$, $n > 1$. Assume n is a power of 2.

$E(1) = 1$.
$E(n) = n^3 + 4E(n/2)$, $n > 1$. Assume n is a power of 2.