Problem Set 2

Assigned: June 5
Due: June 12

Problem 1

(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) then return(k)
    else {
        hpower = exp3(k,floor(n/2));
        if (even(n)) return(hpower*hpower)
           else return(hpower * hpower * k)
           }
}
```

Problem 2

The following recursive algorithm compute a property of a segment of an array `a` of integers between indices `l` and `u`. (None of them are particularly good ways of computing the property, and some are terrible, but that's beside the point.) Let $\texttt{n} = \texttt{u} + 1 - \texttt{l}$ be the length of the segment. For each of the following, write down a recurrence equation for the worst-case running time of the algorithm as a function `T(n)`. For parts (A)-(D), solve the recurrence equation. For (rather difficult) extra credit solve the recurrence equation in (E).

In all these, `a` is a global variable. Assume that this is always called with $\texttt{l} \leq \texttt{u}$ and with both parameters in the index range of `a`. Ignore issues of rounding up and down; assume that all divisions and square roots come out to exact integers,

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

```
A.
// compute the sum of the elements in a[l .. u]
// by adding a[u] to  addUp1[l .. u-1]

int addUp1(l,u) {
  if (u == l) return a[l];
  else return a[u] + addUp1(l,u-1);
}
```

B.

```
// compute the sum of the elements in a[l .. u]
// by adding up the first half, adding up the second half, and adding the two
// sums

int addUp2(l,u) {
   if (u == l) return a{l};
   else {
      m = (u+l)/2;
      return addUp2(l,m) + addUp2(m+1,u);
     }
   }
```

C.

```
// Is the segment a[l .. u] in strictly increasing order?

boolean increasing?(l,u) {
   if (u == l) return true;
   else return increasing?(l,u-1) && (a[u-1] < a[u]);
 }
```

D.

```
// Find the length of the longest subsequence in a[l..u] that is increasing
// order.
// Check whether the entire segment is sorted.
// If not, find the longest subsequence in [l..m-1] and the longest in
// [l+1, m]


int longestInc(l,u) {
  if (increasing?(l,u)) return u+1-l;    // a[l..u] is increasing
  else return max(longInc(l,u-1), longInc(l+1,u))
}
```

E.

```
// compute the sum of the elements in a[l .. u]
// by dividing into chunks of size sqrt(n), recursively adding up each chunk
// and adding them together.

int addUp3(l,u) {
   if (u == l) return a[l];
   if (u == l+1) return a[l]+a[u];
   sum = 0;
   n = u+1-l
   c = floor(sqrt(n));  // size of chunk
   nChunks = ceil(n/c); // number of chunks
   j = l;               // beginning of chunk
   for (i=0; i < nChunks; i++) {
       t = min(j+c-1,u);
       sum = sum + addUp3(j,t):
       j = j+c;
      }
   return sum;
 }
```

Problem 3

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.

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

$C(1) = 1$.
$C(n) = n + 4C(n/2)$, $n > 1$. 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.