CSC721 Algorithms Fall 2015

Exercises # 1

1. In class we discussed an algorithm for finding the optimal order of matrix multiplications. For your ready reference, the algorithm is given below:

   Input: Matrix sizes $r_0, r_1, ..., r_n$ corresponding to a matrix product $\hat{M} = M_1 \times M_2 \times ... \times M_n$

   Output: Optimal Number of scalar operations (multiplies) for the matrix product.
   Method:

   1. **for** $i = 1$ to n do $\quad m_{i,i} = 0 \quad ;$
   2. **for** $\ell = 1$ to $(n-1)$ do
   3. $\quad$ **for** $i = 1$ to $(n - \ell)$ do
   4. $\quad\quad j = \ell + i \quad ;$
   5. $\quad\quad m_{i,j} \ = \ \min_{i \le k < j} \ \ (m_{i,k} \ + \ m_{k+1,j} \ + \ r_{i-1} r_k r_j)$
   6. $\quad$ **end for** i
   7. **end for** ℓ

   Your task: Prove that on line 5, all values of $m_{i,k}$ and $m_{k+1,j}$ needed for the computation of $m_{i,j}$ are already in the table because they have been computed and stored during a previous iteration.

2. This problem is intended to refresh your memory for working with summations. Hint: For most of these problems, try shifting the sequence and subtracting. Simplify the following summations:

   (a) $\sum_{i=1}^{n} a^i$

   (b) $\sum_{i=1}^{n} i a^i$

   (c) $\sum_{i=1}^{n} 2^{n-i} i^2$

3. Put the following functions in the correct order from slowest asymptotic growth to fastest:

$$n \ , \ \ n^2 \ , \ \ \log n \ , \ \ \frac{n}{\log n} \ , \ \ \log^2 n \ , \ \ \sqrt{n}$$

4. Find an asymptotic upper bound for $T(n)$.

$$\begin{aligned} T(1) &= c \\ T(n) &= 2T(n/2) + \log(n) \end{aligned}$$

5. Find an asymptotic upper bound for $T(n)$. Hint: Try bounding a difficult sum with a definite integral

$$\begin{aligned} T(1) &= c \\ T(n) &= T(n-1) + \sqrt{n} \end{aligned}$$

6. Derive an asymptotic upper bound for the following algorithm. Your answer should be a simple expression in terms of `n`.

```
m = n ;  t = 0 ;    // Ok to assume n is a power of 2.
while ( m > 0 ) do
{
      for i = 1 to m do {  t = t + i ;  }
      m = m / 2 ;    //  Integer divide.
}
```

7. Derive an asymptotic upper bound for the following algorithm:

```
// Assume n is a power of 2 and ϒ is a global array of constants.
// Input:
// a[] – array of input values of length n
// m – supplemental variable for indexing ϒ
// Output: b[] – output array
procedure f( array a[], integer n, integer m ; array b[] )
{
   if ( n == 1 ) {
      b[0] = a[0] ;
      return ;
   }
   else {
      f( [ a_0, a_2, ..., a_{n-2}], n/2, 2*m ; x ) ;
      f( [ a_1, a_3, ..., a_{n-1}], n/2, 2*m ; y ) ;
      for ( j = 0 ; j < n/2 ; j++ ) {
         temp = ϒ[ m * j ] * y[j] ;
         b[j] = x[j] + temp ;
         b[j+n/2] = x[j] - temp ;
   }
}
```

8. Derive an asymptotic upper bound for the following algorithm:

```
int g( int n )
{
   if ( n == 1 ) return 1 ;
   else {
      t = 0 ;
      for ( i = 1 ; i < n ; i++ )  t += i + g(i)  ;
   }
}
```

9. Consider the following code segment:

```
for i = 1 to n do
     <<Statement>>
```

It is obvious that `Statement` is repeated n times. Let's look at two nested loops where the index variable in the outer loop is the upper bound for the inner loop. Consider:

```
for i = 1 to n do
     for j = 1 to i do
          <<Statement>>
```

Analysis shows that `Statement` is repeated $\frac{n(n+1)}{2}$ times. Now consider three nested loops.

```
for i = 1 to n do
     for j = 1 to i do
        for k = 1 to j do
            <<Statement>>
```

Analyze the code segment above and derive an exact expression (in terms of n) that describes the number of times `Statement` is repeated.

10. For this part, refer to your result in part (9). Conjecture an exact expression for the number of times `Statement` is repeated for k nested loops of this form. I.e., the upper bound for each loop (except the outermost loop) is the loop index of the enclosing loop.

Prove your conjecture by induction on the number of nested loops.

for $i_1 = 1$ to n do
    for $i_2 = 1$ to i_1 do
        for $i_3 = 1$ to i_2 do
            .
                .
                    .
                        for $i_k = 1$ to i_{k-1} do
                            `<<Statement>>`

Hint:

Use one of the identities involving binomial coefficients $\binom{n}{k}$ from the "cheat sheet".