CS1102: Data Structure and Algorithms

Tutorial 6 - Analysis of Algorithms

Week of 08 March 2010

1. Rearrange the following functions in the increasing order of their Big Oh complexity:

   $4n^2$, $\log_3(n)$, $20n$, $n^{2.5}$, 2, n^n, 3^n, $n \log(n)$, $100n^{2/3}$, 2^n, 2^{n+1}, $n!$, $(n-1)!$, 2^{2n}

2. Write the following different version of the exponential functions X^n

   ```
   double Exponential(double base, int exp)
   //Assumption: exp >= 0
   ```

   a. Iterative version
   b. Recursive version, utilizing the fact $X^n = X * X^{n-1}$
   c. Recursive version, utilizing the fact $X^{2n} = (X^n)^2$

   Briefly comment on the time complexity of each of the version using Big-O notation.

3. Analyze the Big Oh complexity of each of the following code fragments.

a)
```
// loop 1
for(int i = 0; i < n; i++)
     // loop 2
     for(int j = 0; j < n; j++)
          System.out.println(“*”);
```

b)
```
// loop 1
for(int i = 0; i < n; i++)
     // loop 2
     for(int j = 0; j < i; j++)
          System.out.println(“*”);
```

c)
```
// loop 1
for(int i = 0; i < n; i++)
     // loop 2
     for(int j = i+1; j > i; j--)
          // loop 3
          for(int k = n; k > j; k--)
               Sytem.out.println(“*”);
```

d)
```
// loop 1
for (int i = 0; i < n; i++) {
     // loop 2
     for (int j = 0; j < n; j++) {

          if ((j % 2) == 0) {
               // loop 3
               for (int k = i; k < n; k++)
                    System.out.println(“*”);
               }
          else {
               // loop 4
               for (int l = 0; l < i; l++)
                    System.out.println(“*”);
          }
     }
}
```

4. We consider the problem of finding the maximum sum of a contiguous subsequence of integers from a given input data sequence. For example, if the input data sequence is {-1, 12, -7, 13, -5, 2, -3}, the answer is 18 (12-7+13). The maximum subsequence sum is defined as 0 if all the integers are negative. Your may use the following formulas:

$$\sum_{j=i}^{n-1} j = i+(i+1)+\ldots+(n-2)+(n-1) = (n+i-1)(n-i)/2$$
$$\sum_{j=0}^{n-1} j^2 = 0^2+1^2+\ldots+(n-2)^2+(n-1)^2 = (n-1)n(2n-1)/6$$

(a) The most obvious way to solve this problem is to compute the sum of each possible subsequence, and retain the largest sum as the result.

```
// a is an array which stores the input data sequence of length
//n
int maxSubSum1( int[]a )
{
      int max_sum = 0; // maximum sum of a contiguous
                       //subsequence
      int n = a.length;// the length of the array

      //loop 1:  i is the starting index of a subsequence
      for(int i=0; i< n; i++)
      {
            //loop 2: j is the ending index of a
            //subsequence
            for(int j=i; j< n; j++)
            {
                  int this_sum=0; // the sum of the current
                                  //subsequence
                  for(int k=i; k <= j; k++) //loop 3
                  {
                        this_sum += a[k];
                  }
                  if(this_sum > max_sum) max_sum = this_sum;
            }
      }
      return max_sum;
}
```

Each subsequence is identified by its starting index and ending index in the array. In the code, loop 1 and loop 2 are used to generate all possible pairs of starting and ending indices. For each subsequence, loop 3 is used to compute the sum of the elements in the subsequence.

What is the Big Oh complexity of this algorithm?

(b) The following code is an improved version of the solution in (a).

```
int maxSubSum2( int[]a )
{
      int max_sum = 0; // maximum sum of a contiguous
                       //subsequence
      int n = a.length;//length of the array

      //loop 1: i is the starting index of a subsequence
      for(int i=0; i< n; i++)
      {
            int this_sum= 0; // the current calculated sum
            //loop 2: j is the ending index of a subsequence
            for(j=i; j< n; j++)
            {
                  this_sum += a[j]; // the sum of the current
                                    //subsequence
                  if(this_sum > max_sum) max_sum = this_sum;
            }
      }
      return max_sum;
}
```

Explain why this solution correctly computes the maximum subsequence sum. Analyze its Big Oh complexity.

(c) Below is our final algorithm for computing the maximum subsequence sum. Explain why it is correct and analyze its Big Oh complexity.

```
int maxSubSum4( int[]a )
{
      int max_sum = 0;
      int this_sum = 0;
      int n = arr.length;
      for(int i=0; i< n; i++)
      {
            this_sum += a[i];
            if(this_sum > max_sum)
                  max_sum = this_sum;
            else if(this_sum < 0)
                  this_sum = 0;
      }
      return max_sum;
}
```