Homework Four

Q1 Show that if p(n) is a polynomial in n, then log p(n) is O(log n).

Solution: Let $p(n) = a_m n^m + a_{m-1} n^{m-1} + \ldots + a_1 n + a_0$
$p(n) < \max\{|a_m|, |a_{m-1}|, \ldots, |a_0|\} (m+1) n^m$
$\log p(n) < \log(\max\{|a_m|, |a_{m-1}|, \ldots, |a_0|\} (m+1) + m \log n$
Since m and a_i (i=0, 1, …, m) are constants, we have $O(\log p(n)) = O(\log(\max\{|a_m|, |a_{m-1}|, \ldots, |a_0|\}) (m+1) + O(m \log n) = O(\log n)$.

Q2 Show that $1^2 + 2^2 + \ldots + n^2$ is $O(n^3)$.

Solution 1: The area of the shape enclosed by the x-axis, the vertical line x=0, the vertical line x=n+1, and the curve $y=x^2$, is given by $\int_0^{n+1} x^2\, dx$. This shape contains every rectangle formed by the four lines x=i, x=i+1, y=0 and $y=i^2$ (i=1, 2, …, n). Therefore, we have the following inequality:

$$\sum_{i=1}^{n} i^2 < \int_0^{n+1} x^2 dx$$
$$\int_0^{n+1} x^2 dx = \frac{(n+1)^3}{3} = O(n^3)$$

Solution 2: $1^2 + 2^2 + \ldots + n^2 = n(n+1)(2n+1)/6 = n^3/3 + n^2/2 + n/6 = O(n^3)$.

Q3 Show that $\sum_{i=1}^{n} \frac{i}{2^i}$ is O(1).

Solution: Let $S = \sum_{i=1}^{n} \frac{i}{2^i}$. $s = \sum_{i=1}^{n} \frac{i}{2^i} = \sum_{i=1}^{n} \frac{1}{2^i} + \sum_{i=2}^{n} \frac{i-1}{2^i} = \sum_{i=1}^{n} \frac{1}{2^i} + \sum_{i=1}^{n-1} \frac{i}{2^{i+1}} < 1 + s/2$.
Therefore, S<2. As a result, $\sum_{i=1}^{n} \frac{i}{2^i}$ is O(1).
Comments: It is easy to prove $\sum_{i=1}^{n} \frac{1}{2^i} < 1$. Let $A = \sum_{i=1}^{n} \frac{1}{2^i}$. $2A = \sum_{i=1}^{n} \frac{2}{2^i} = \sum_{i=1}^{n} \frac{1}{2^{i-1}} = 1 + \sum_{i=1}^{n-1} \frac{1}{2^i}$
<1+A. Threfore, A<1.

Q4 Show that $\sum_{i=1}^{n} \log i$ is O(n log n).

Solution: $\sum_{i=1}^{n} \log i < n \log n = O(n \log n)$

Q5 Consider the following algorithm:

Algorithm Unknown(A, B)
Input: Arrays A and B each storing n>0 integers.
Output: The number of elements in B equal to the sum of prefix sums in A.

```
c=0;
for i=0 to n-1 do
  { s=0;
    for j=0 to n-1 do
      { s=s+A[0];
        for k=0 to n-1 do
          s=s+A[k];
      }
    if ( B[i]=s )
      c=c+1;
```

```
  }
return c;
```

What is the time complexity of this algorithm in big-Oh notation?

Solution: The number of primitive of each statement is shown as follows.

Statements	Number of primitive operations of each statement
c=0;	1
for i=0 to n-1 do	n
{ s=0;	n
for j=0 to n-1 do	n^2
{ s=s+A[0];	n^2
for k=0 to n-1 do	n^3
s=s+A[k];	n^3
}	
if (B[i]=s)	n
c=c+1;	n
}	
return c;	1

The total number of primitives is $3n^3+2n^2+4n+2=O(n^3)$.

Q6 Let p(x) be a polynomial of degree n, where $p(x)= a_nx^n + a_{n-1}x^{n-1} + \ldots + a_1x + a_0$. Design an O(n)-time algorithm for computing p(x).

Solution: Rewrite p(x) as $p(x) = (\ldots((a_nx + a_{n-1})x + a_{n-2})x + \ldots + a_1)x + a_0$, which can be recursively computed as follows: $p_0=a_n$, $p_{i+1}=p_i\, x+ a_i$ (i=0, 1, …, n-1). The algorithm is as follows:

Algorithm computingPloynomial(x, n, A[])

Input: A polynomial where all the coefficients are stored in a one-dimensional array A of size n+1 and x stores the value of the variable.
Output: The result of the polynomial for x.

```
p= A[n];
for (i=1; i<n+1; i++)
  {
    p=p*x+A[n-i];
  }
return p;
```

Q7 The Tower of Hanoi is a classical problem which can be solved by recurrence. There are three pegs and N disks of different sizes. Originally, all the disks are on the left peg, stacked in decreasing size from bottom to top. Our goal is to transfer all the disks to the right peg, and the rules are that we can only move one disk at a time, and no disk can be moved onto a smaller one. We can easily solve this problem with the following recursive algorithm: If N = 1, move this disk directly to the right peg and we are done. Otherwise (N >1), first transfer the top N-1 disks to the middle peg applying the algorithm recursively, then move the largest disk to the right peg, and finally transfer the N-1 disks on the middle peg to the right peg applying the algorithm recursively. Let T(N) be the total number of moves needed to transfer N disks. We have that T(1) = 1, and T(N) = 2T(N-1)+ 1. What is the time complexity of this algorithm in big-Oh notation?

Solution:
$$\begin{aligned} T(N) &= 2(2T(N-2)+1)+1 \\ &= 4T(N-2)+2+1 \\ &= 4(2T(N-3)+1)+2+1 \\ &= 8T(N-3)+4+2+1 \end{aligned}$$

$= \ldots$
$= 2^{(N-1)}T(1) + 2^{N-2} + \ldots + 2+1$
$= 2^{(N-1)}T(1) + 2^{N-1} - 1$
$= 2^N - 1$

Therefore, the time complexity of this algorithm is $O(2^N)$.

Q8 The Towers of Providence is a variation of the classical Towers of Hanoi problem. There are four pegs, denoted A, B, C, and D, and N disks of different sizes. Originally, all the disks are on peg A, stacked in decreasing size from bottom to top. Our goal is to transfer all the disks to peg D, and the rules are that we can only move one disk at a time, and no disk can be moved onto a smaller one.

We can solve this problem with a recursive algorithm: If N = 1,move this disk directly to peg D, and we are done. Otherwise (N > 1), perform the following steps:

1. transfer the top N-2 disks on peg A to peg B applying the algorithm recursively;
2. move the second largest disk from peg A to peg C;
3. move the largest disk from peg A to peg D;
4. move the second largest disk from peg C to peg D;
5. transfer the N-2 disks on peg B to peg D applying the algorithm recursively.

Let T(N) be the total number of moves needed to transfer N disks. We have:
T(1) = 1; T(N) = 2T(N-2) + 3: What is the time complexity of this algorithm in big-Oh notation?

Solution: $T(N) = 2(2T(N-4) + 3) + 3$
$= 4T(N-4) + 3*(2 + 1)$
$= 4(2T(N-6) + 3) + 3*(2 + 1)$
$= 8T(N-6) + 3*(4 + 2 + 1)$
$= \ldots$
$= 2^{(N-1)/2}T(1) + 3*(2^{(N-3)/2} + \ldots + 2+1)$
$= 2^{(N-1)/2}T(1) + 3*(2^{(N-1)/2} - 1)$
$= 2^{(N-1)/2} + 3*2^{(N-1)/2} - 3$
$= 2^2*2^{(N-1)/2} - 3$
$= 2^{(N+3)/2} - 3$

Therefore, the time complexity of this algorithm is $O(2^{(N+3)/2} - 3) = O(2^N)$.