15Fall CSC 505 HW1

Suggested solution

1. (6 points) Goal: Practice analysis of algorithms. Consider the algorithm represented by the following program fragment.

```
1   MYSTERY(int x, int n)
2   {
3    int sum=0;
4    if x>10 then {
5           for (int i=1; i<=n^2; i++) {
6                  sum=sum*i;
7           }
8    } else {
9           for (int i=1; i<=n; i++) {
10                 sum=sum+i;
11          }
12   }
13   return x + sum;
14   }
```

a) (2 points) As a function of x and n, what value does MYSTERY(x,n) return?
b) (2 points) Use only ARITHMETIC operations as basic operations, and assume that the size of the input is measured by n. Compute the worst-case running time of MYSTERY.
c) (2 points) Assume that x remains constant while n goes to infinity. Derive a tight, big-Oh expression (dependent on the value of x) for the running time of MYSTERY. Justify your solution.

a . Mystery(x,n) = x, if x > 10 and
x+n(n+1)/2, if x <=10.

b. Considering the Arithmetic operations $+$ and $*$, the worst-case running time of $Mystery(x, n)$ based on an input size of n is $O(n^2)$, because for $x > 10$, the algorithm will loop n^2 times, performing one $+$ operation (line 5) and two $*$ operations (lines 5 and 6; the first * is when n is squared [n*n]) each time. At the end of the algorithm, it performs one last $+$ operation (line 13). This puts the running time for $x > 10$ as $3n^2 + 1$, which is in the class of $O(n^2)$.

c. As above, the running time for $x > 10$ is $3n^2 + 1$. This means that the tight-big-oh expression for $x > 10$ is $O(n^2)$. For $x \leq 10$, the worst case running time must perform two $+$ operations n times (lines 9 and 10), followed by one more $+$

operation at the end (line 13). Thus, the running time for $x \leq 10$ is $2n + 1$, putting the tight-big-oh expression for $x \leq 10$ at $O(n)$. This means that the overall tight-big-oh expression for Mystery, dependent on x, is $O(n^2)$ for $x > 10$, and $O(n)$ for $n <= 10$.

2. (8 points) Purpose: Learn about bubblesort, practice running time analysis, learn how loop invariants are used to prove the correctness of an algorithm. Tip: re-read Section 2.1 in our textbook. Solve problem 2-2 [a-d] on page 40 of the textbook.

a) The A could be any permutation of A'[1], A'[2],...,A'[n], , and the elements in A and A' are the same.

b) **Loop invariant**: Before the loop starts with j = a, elements from A[a+1] to A[length] are no less than A[j]. Namely, A[j] is the least element from A[j] to A[length].

Initialization:

Before the first loop starts with j = A.length, since there is no elements from A[a+1] to A[length], so they are no less than A[j].

Maintenance:

We compare A[j] with A[j-1], if A[j] is less than A[j-1], we swap the value of A[j] and A[j-1], so now, A[j-1] ≤ A[j], since A[j+1] to A[length] are no less than A[j-1], therefore, A[j] to A[length] are no less than A[j-1], namely, A[j-1] is the least element from A[j-1] to A[length], j = j-1 for the next iteration, then loop invariant is preserved.

Termination:

It terminates when j = i, according to the loop invariant, elements from A[i+1] to A[length] are no less than A[i], Namely, A[i] is the least element from A[i] to A[length]. Therefore, we find the least element from A[i] to A[length].

c) **Loop invariant:**

Before the *j*th loop starts, A[1] to A[j-1] are in sorted order and they all no bigger than any element from A[j] to A[length].

Initialization:

Before the first loop, since there is no elements before A[1] , we can say elements before A[i] are in sorted order. Also, we can say they all no bigger than any element from A[1] to A[length].

Maintenance:

Before the *i*th loop, A[1] to A[i-1] are in sorted order and they all no bigger than any element from A[i] to A[length]. According to the termination condition of invariant proved in part(b), the inner “for” loop will make A[i] the least element from A[i] to A[length]. Therefore, A[1] to A[i] are in sorted order and they all no bigger than any element from A[i+1] to A[length]. So loop invariant is preserved.

Termination:

It terminates when i = length, according to the loop invariant, before this loop starts, A[1] to A[length-1] are in sorted order and they all no bigger than any element f rom A[length] to A[length]. Therefore, A[1] to A[length] are in sorted order.

d). O(n^2), regardless of the input, the number of execution in bubble sort is the sa me. For the *i*th outer "for" loop, the inner loop will execute length-i times, so the to tal execution time:

$$\sum_{i=1}^{length-1} length-i = \sum_{i=1}^{n-1} n-i = \frac{[(n-1)+1](n-1)}{2} = \frac{n(n-1)}{2} \in O(n^2)$$

Therefore, it has the same time complexity as insertion sort for the worst case.

3. Purpose: practice working with asymptotic notation. Please solve(12 points) 3-2 on page 61, (8 points) 3-4 [a-c] on page 62. For full marks justify your solutions.

	A	B	O	o	Ω	ω	Θ	Explanation
a.	$\lg^k n$	n^ε	*Y*	*Y*	*N*	*N*	*N*	Apply L'Hospital's Rule: $\lg^k n = (\log_2 n)^k = \left(\frac{\ln n}{\ln 2}\right)^k$ $\lim_{n\to\infty} \frac{\lg^k n}{n^\varepsilon} = \left(\frac{1}{\ln 2}\right)^k \lim_{n\to\infty} \frac{(\ln n)^k}{n^\varepsilon} = \left(\frac{1}{\ln 2}\right)^k \lim_{n\to\infty} \frac{k(\ln n)^{k-1}}{\varepsilon n^\varepsilon} = \left(\frac{1}{\ln 2}\right)^k \lim_{n\to\infty} \frac{k!}{\varepsilon^k n^\varepsilon} = \frac{k!}{(\ln 2)^k \varepsilon^k} \lim_{n\to\infty} \frac{1}{n^\varepsilon} = 0$
b.	n^k	c^n	*Y*	*Y*	*N*	*N*	*N*	Apply L'Hospital's Rule: $\lim_{n\to\infty} \frac{n^k}{c^n} = \lim_{n\to\infty} \frac{kn^{k-1}}{c^n(\ln c)} = \lim_{n\to\infty} \frac{k!}{c^n(\ln c)^k} = \frac{k!}{(\ln c)^k} \lim_{n\to\infty} \frac{1}{c^n} = 0$
c.	$\sqrt{n}$	$n^{\sin n}$	*N*	*N*	*N*	*N*	*N*	The value of $\sin n$ fluctuates between -1 and 1 and is therefore not comparable.
d.	2^n	$2^{\frac{n}{2}}$	*N*	*N*	*Y*	*Y*	*N*	Apply L'Hospital's Rule: $\lim_{n\to\infty} \frac{2^n}{2^{\frac{n}{2}}} = \lim_{n\to\infty} 2^{\frac{n}{2}} = \infty$
e.	$n^{\lg c}$	$c^{\lg n}$	*Y*	*N*	*Y*	*N*	*Y*	$n^{\lg c} = c^{\lg n}$
f.	$\lg(n!)$	$\lg(n^n)$	*Y*	*N*	*Y*	*N*	*Y*	$\lg(n!) = \sum_{i=1}^{n} \lg n$ $\lg(n^n) = n \lg n$ Because $\sum_{i=1}^{n} \lg n \le n \lg n$, $\lg(n!)$ is Big-Oh of $\lg(n^n)$. Because $\sum_{i=1}^{n} \lg i \ge \sum_{i=i=\frac{n}{2}+1}^{n} \lg i \ge \sum_{i=i=\frac{n}{2}+1}^{n} \lg\left(\frac{n}{2}\right) = \left(\frac{n}{2}-1\right) \lg\left(\frac{n}{2}\right)$, $\lg(n!)$ is Big-Omega of $\lg(n^n)$.

3-4

[a] False.

We can disprove by counterexample. For example, assume that $f(n) = n$, $g(n) = n^2$,

then $f(n) = O(g(n))$, but $g(n) \ne O(f(n))$.

[b] False.

We can disprove by counterexample. For example, assume $f(n)=n$, $g(n)=n^2$, then

$f(n)+g(n)=n+n^2=\Theta(n^2)$, but $\min(f(n),g(n))=\Theta(n)$. They are not equal.

[c] True.

$f(n)=O(g(n))$, there exists some constant $c>0$ and $n_0>0$ such that

$0\le f(n)\le c\cdot g(n)$ for all $n\ge n_0$.

Since $f(n)\ge 1$ and $\lg(g(n))\ge 1$,

$$\lg(f(n))\le \lg(c\cdot g(n))=(\frac{\lg c}{\lg(g(n))}+1)\cdot\lg(g(n))\le(\lg c+1)\cdot\lg(g(n)).$$

We can always find a $c'=\lg c+1>0$, so that $\lg(f(n))=O(\lg(g(n)))$.

4. Purpose: Practice working with asymptotic notation. Rank the following functions by order of growth.

a) (5 points) Find an arrangement f1, f2, ..., fn of the functions satisfying f1 = Ω(f2), f2= Ω(f3), ..., fn-1 = Ω(fn). (Here, lg := logarithm base 2)

b) (2 point) The order is NOT unique. Which functions can be exchanged, i.e. $f_k\in\Theta(f_{k+1})$

n^2, log(n!), n^{-2}, $e^{2\ln(n)}$, $\ln(n^2)$, 1000, sqrt(n), $3^{\lg(n)}$

(a) n^2, $e^{2\ln(n)}$, $3^{\lg n}$, log(n!), sqrt(n), $\ln(n^2)$, 1000, n^{-2}

(b) n^2 and $e^{2\ln(n)}$ is not unique because $e^{2\ln(n)}=e^{\ln(n^\wedge 2)}=n^2$

5. (2 points) Purpose: Practice working with asymptotic notation. Assume that a and b are integers, and a,b>0. Show $e^{an}\in\omega((bn)^t)$.

$$\lim_{n\to\infty}\frac{e^{an}}{(bn)^t}=\lim_{n\to\infty}\frac{ae^{an}}{bt(bn)^{t-1}}=\cdots=\lim_{n\to\infty}\frac{a^te^{an}}{b^t\times t!}=\infty$$

6. (8 points) Purpose: Practice algorithm design and algorithm analysis. Give pseudocode for a $\Theta(\lg n)$ algorithm which computes a^n,given a and n. Justify the asymptotic running time of your algorithm. Do not assume that n is a power of 2.

Algorithm Fast_Power(a,n)

```
result = 1
if n == 0 then
    return result
product = a
k = ⌊lg n⌋+1
for i = 1 to k do
    if n is even then
        n =   n/2
    else
        n = (n-1)/2
        result = result * product
    product = product * product
return result
```

This algorithm calculates a^n in $\lfloor \lg n \rfloor + 1$ iterations of the for loop. Since the number of operations in each loop is bound by a constant, the algorithm has $\Theta(\lg n)$ worse-case running time.

The algorithm calculates $a^{(a_0 2^0)}, a^{(a_0 2^0 + a_1 2^1)}, ..., a^{(a_0 2^0 + ... + a_k 2^k)}$ and store these values in the variable result. For example, for $n = 5 = 1*2^2 + 0*2^1 + 1*2^0$, the algorithm successively computes:

Initialization:

$result = 1,\ n = 5,\ product = a$

for-loop:

$i = 1, result = a, n = 2, product = a^2$

$i = 2, result = a, n = 1, product = a^4$

$i = 3, result = a^5, n = 0, product = a^8$

return result a^5