EECS560 Practice HW#1 Solution Fall 2018

Instruction: You must show all your work clearly for credit. Partial credit will only be given to meaningful answers.

1. When implementing an ADT for a set of records S, |S| $= 2^6$, it is determined that a find operation, find(x,S), will requires 0.5ms (10^{-3}s) to execute. If the complexity of the find operation is given by the following closed-form expressions T(n), compute the time required to execute this operation when |S| = 2^{16}.
   (a) T(n) = 560.
   (b) T(n) = nlgn.
   (c) T(n) = n^2lgn.
   (d) T(n) = n^3.

Solution:

$$\frac{T(n)}{C(n)} = \frac{T(n^*)}{C(n^*)}$$

$$C(n^*) = \frac{T(n^*)}{T(n)} * C(n) = \frac{T(n^*)}{T(n)} * 0.5ms$$

(a) T(n) = 560.

$$C(n^*) = \frac{T(n^*)}{T(n)} * 0.5ms = \frac{560}{560} * 0.5ms = 0.5ms.$$

(b) T(n) = nlgn.

$$C(n^*) = \frac{T(n^*)}{T(n)} * 0.5ms = \frac{2^{16} * \lg 2^{16}}{2^6 * \lg 2^6} * 0.5ms = 2^{10} * \frac{16}{6} * 0.5ms = 1,365.33ms.$$

(c) T(n) = n^2lgn.

$$C(n^*) = \frac{T(n^*)}{T(n)} * 0.5ms = \frac{2^{32} * \lg 2^{16}}{2^{12} * \lg 2^6} * 0.5ms = 2^{20} * \frac{16}{6} * 0.5ms = 1.33 * 2^{20}\, ms.$$

(d) T(n) = n^3.

$$C(n^*) = \frac{T(n^*)}{T(n)} * 0.5ms = \frac{2^{48}}{2^{18}} * 0.5ms = 0.5 * 2^{30}\, ms.$$

2. If an algorithm requires 0.5ms to solve a problem with input size of 100, how large a problem it can solve in 1 min if the complexity of the algorithm is given by the following function T(n) in closed-form?
   (a) T(n) = n.
   (b) T(n) = n^2.

Solution:

$$\frac{T(n)}{C(n)} = \frac{T(n^*)}{C(n^*)}$$

$$T(n^*) = \frac{C(n^*)}{C(n)} * T(n) = \frac{60*10^3}{0.5} * T(n) = 120,000 * T(n)$$

(a) T(n) = n: 120,000 times as large a problem, or input size 12,000,000.

$$T(n^*) = \frac{C(n^*)}{C(n)} * T(n) = \frac{60*10^3}{0.5} * T(n) = 120,000 * T(n)$$

Hence, $n = 120,000 * 100 = 12,000,000.$

(b) T(n) = n^2:

$$T(n^*) = \frac{C(n^*)}{C(n)} * T(n) = \frac{60*10^3}{0.5} * T(n) = 120,000 * T(n).$$

Hence, $n^2 = 120,000 * 100^2 = 1,200,000,000.$

$n = \sqrt{1,200,000,000}.$

3. Given the following algorithm for finding the two largest integers in an array A[1..n] of n distinct positive integers. Base on the number of comparisons between elements in A, compute $T_b(n)$ and $T_w(n)$. You must justify your answer and show your work clearly for credit.

```
if A[1] > A[2]                        // Initialization
   then   largest = A[1];
          s_largest = A[2]
   else   largest = A[2];
          s_largest = A[1]
endif;
for i = 3 to n do                     // Checking A[3], ..., A[n]
   if A[i] > s_largest                // A[i] is one of the two largest integers
      then  if A[i] > largest         // A[i] is the current largest integer
                then  s_largest = largest;
                      largest = A[i]
                else  s_largest = A[i]
            endif
   endif
endfor;
```

Solution:

In executing the for-loop, it will require either one or two comparisons, depending on whether A[i] is one of the 2 largest integers in A[1..i]. Hence,

$$T_b(n) = 1 + \sum_{i=3}^{n} 1 = 1 + (n-3+1) = n-1, \text{ and}$$

$$T_w(n) = 1 + \sum_{i=3}^{n} 2 = 1 + 2(n-3+1) = 2n-3.$$

Observe also that a decreasing sequence gives rise to the best-case complexity and an increasing sequence gives rise to the worst-case complexity.

4. Assuming that all basic operations require the same constant cost C, by concentrating on the dominating step(s), compute the cost of the resource function R(n) for the following program segment in closed-form.

```
x = 2;
y = 10;
for i = 1 to n do
  for  j = i to n do
        y = x * y / 2;
  endfor;
  for k = 1 to n do
        x = x + y − 10;
  endfor;
endfor;
```

Solution:

$$
\begin{aligned}
&R(n)\\
&=\sum_{i=1}^{n}\Big(\sum_{j=i}^{n}+\sum_{k=1}^{n}\Big)C,\ C-\text{constant}\\
&=C\sum_{i=1}^{n}[(n-i+1)+n]\\
&=C\sum_{i=1}^{n}[(2n+1)-i]\\
&=C[(2n+1)n-\frac{n(n+1)}{2}].
\end{aligned}
$$

5. By concentrating on the dominating step and by assuming that all basic operations require the same constant cost C, compute $T_W(n)$ in closed-form for the following program segment as discussed in class.
Remark: You must first set up the equation for $T_W(n)$ and then evaluate its sums for credits. Do not simplify the final expression.

```
x = 2;
y = 10;
k = 1;
while k ≤ n do
      x = x + x*y +210;
      y = y – x + 560;
      k = k+1;
endwhile;
for i = 1 to n do
   for  j = i to n do
         y = x * y / 2;
         for k = j to n do
               x = x + y – 10;
         endfor;
   endfor;
endfor;
```

Solution:
T(n)

$$= \sum_{i=1}^{n} (\sum_{j=i}^{n} \sum_{k=j}^{n} C),\ C - \text{constant}$$

$$= C\sum_{i=1}^{n} \sum_{j=i}^{n} (n - j + 1)$$

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

$$= \frac{C}{2} \sum_{i=1}^{n} (n^2 - 2ni + i^2 + 3n - 2i + 2)$$

$$= \frac{C}{2} [n^3 - n^2(n+1) + \frac{n(n+1)(2n+1)}{6} + 3n^2 - n(n+1) + 2n].$$

6. Let A_1 and A_2 be two algorithms with closed-form complexity $T_1(n) = 10n^2$ and $T_2(n) = 499n + 50$. Find smallest integer n_0 such that for all $n > n_0$, algorithm A_2 will always be more efficient than algorithm A_1.
   Solution:
   Consider $10n^2 \geq 499n + 50$.
   $10n^2 - 499n - 50 \geq 0,$
   $(10n+1)(n-50) \geq 0,$
   $\therefore n = -\frac{1}{10},\ 50.$
   $\therefore$ Smallest $n_0 = 50.$
   Conclusion:
   For all $n > 50$, algorithm A_2 will always be more efficient than algorithm A_1.

7. Use the definition of big-O to prove or disprove that $2^{2n} = O(3^n)$.
   Solution:
   If true, there exists constants $k > 0$ and $n_0 > 0$ such that for all $n > n_0$, $2^{2n} \leq k3^n$.
   Hence, $2^{2n} = 4^n \leq k3^n$, for all $n > n_0$.
   Dividing both sides by 4^n, we have $\frac{4^n}{3^n} = (\frac{4}{3})^n \leq k.$
   As $n \to \infty$, we have $\infty \leq k$ = constant.
   Hence, a contradiction is reached and $2^{2n} \neq O(3^n)$.

8. Prove or disprove that if $T_1(n) = O(f(n))$ and $T_2(n) = O(f(n))$, the $T_1(n) + T_2(n) = O(f(n))$.
   Solution:
   True.
   $T_1(n) = O(f(n))$ implies that $\exists$ constants k_1 and $n_1 \ni T_1(n) \leq k_1f(n), \forall n \geq n_1$.
   $T_2(n) = O(f(n))$ implies that $\exists$ constants k_2 and $n_2 \ni T_2(n) \leq k_2f(n), \forall n \geq n_2$.
   Hence, for $n = \max\{ n_1, n_2\}$, we have
   $T_1(n) + T_2(n) \leq k_1f(n) + k_2f(n) = (k_1 + k_2)f(n) = Kf(n), K = k_1 + k_2$ = constant.

9. Prove or disprove that if $T_1(n) = O(f(n))$ and $T_2(n) = O(f(n))$, then $\frac{T_1(n)}{T_2(n)} = O(1)$.
   Solution:
   False.
   Counterexample:
   Take $T_1(n) = n^2 = O(n^2)$, $T_2(n) = n = O(n^2)$ with $f(n) = n^2$.
   Hence, $T_1(n)/T_2(n) = n = \Theta(n)$, which is not $O(1)$.

10. Using the definition of big-O to prove that $\dfrac{n^4 - n^3 - 2n^2 + 4}{2n^2 - 2n - 27} = \Omega(n^2)$.

Solution:

$$\frac{n^4 - n^3 - 2n^2 + 4}{n^2 - 2n - 27}$$

$$\geq \frac{n^4 - n^3 - 2n^2}{n^2 - 2n - 27}, \forall n \geq 1$$

$$\geq \frac{\frac{n^4}{3} + (\frac{n^4}{3} - n^3) + (\frac{n^4}{3} - 2n^2)}{n^2 - 2n - 27}, \forall n \geq 1$$

$$\geq \frac{\frac{n^4}{3}}{n^2 - 2n - 27}, \forall n \geq 3$$

$$\geq \frac{\frac{n^4}{3}}{n^2}, \forall n \geq 3$$

$$\geq \frac{1}{3} n^2, \forall n \geq 3.$$

Hence, by choosing k = 1/3, n_0 = 3, we proved that $\dfrac{2n^4 - n^3 - 2n^2 + 4}{2n^2 - 2n - 27} = O(n^2)$.

11. Use the definition of big-Θ to prove that $\dfrac{2n^4 - n^3 - 5n^2 + 4}{n^2 - 6n + 7} = \Theta(n^2)$.

Solution:

$$(i)\frac{2n^4 + n^3 - 5n^2 + 4}{n^2 - 6n + 7}$$
$$\le \frac{2n^4 + n^4 + 4n^4}{n^2 - 6n + 7}, \forall n \ge 1$$
$$\le \frac{7n^4}{n^2 - 6n}, \forall n \ge 1$$
$$= \frac{7n^4}{\frac{n^2}{2} + (\frac{n^2}{2} - 6n)}, \forall n \ge 1$$
$$\le \frac{7n^4}{\frac{n^2}{2}}, \forall n \ge 12$$
$$= 14n^2, \forall n \ge 12.$$

Hence, by choosing k = 14, n_0 = 12, we proved that $\dfrac{2n^4 - n^3 - 5n^2 + 4}{n^2 - 6n + 7} = O(n^2)$.

$$(ii)\frac{2n^4 + n^3 - 5n^2 + 4}{n^2 - 6n + 7}$$
$$\ge \frac{2n^4 - 5n^2}{n^2 - 6n + 7}, \forall n \ge 1$$
$$= \frac{n^4 + (n^4 - 5n^2)}{n^2 - 6n + 7}, \forall n \ge 1$$
$$\ge \frac{n^4}{n^2 - 6n + 7}, \forall n \ge 5$$
$$\ge \frac{n^4}{n^2 + 7n^2}, \forall n \ge 5$$
$$= \frac{1}{8}n^2, \forall n \ge 5.$$

Hence, by choosing k = 1/8, n_0 = 5, we proved that $\dfrac{2n^4 - n^3 - 5n^2 + 4}{n^2 - 6n + 7} = \Omega(n^2)$.

By the definition of big-Θ to prove that $\dfrac{2n^4 - n^3 - 5n^2 + 4}{n^2 - 6n + 7} = \Theta(n^2)$.