Data Structure and Algorithm

Homework #1 Solution

TA email: dsa1@csie.ntu.edu.tw

Problem 1. Basic Matrix Operation (30%)

Sample code is shown on the course website.

Problem 2. Asymptotic Notation (20%)

2.1. (10%)

(a) for $i \in \{0, 1, 2, ...d\}, a_i \cdot n^i \leq a_d \cdot n^k$ when $n \geq \max\{1, \frac{a_i}{a_d}\}$

By summing the $d+1$ inequalities, we can obtain that $p(n) \leq (d+1) \cdot a_d \cdot n^k$ when $n \geq \frac{\max\{a_0, a_1, a_2, a_3, ..., a_d\}}{a_d}$.

$\rightarrow p(n) = O(n^k)$

(b) The key idea is making $|a_i| \cdot n^i \leq \frac{1}{d+1} \cdot a_d \cdot n^d$ for $i \in \{0, 1, 2, ...d-1\}$

We know that the equality holds when $n \geq \max\{1, (d+1) \cdot \frac{a_i}{a_d}\}$

$$p(n) = \sum_{i=0}^{d} a_i \cdot n^i = a_d \cdot n^d + \sum_{i=0}^{d-1} a_i \cdot n^i$$

$$\geq a_d \cdot n^d - \sum_{i=0}^{d-1} |a_i| \cdot n^i$$

$$\geq a_d \cdot n^d - d(\tfrac{1}{d+1} \cdot a_d \cdot n^d) = \tfrac{1}{d+1} \cdot a_d \cdot n^d$$

$\geq \frac{1}{d+1} \cdot a_d \cdot n^k$ when $n \geq \max\{1, (d+1) \cdot \frac{a_0}{a_d}, (d+1) \cdot \frac{a_1}{a_d}, (d+1) \cdot \frac{a_2}{a_d}, ..., (d+1) \cdot \frac{a_d}{a_d}\}$

$\rightarrow p(n) = \Omega(n^k)$

(c) We know that $f(n) = \Theta(g(n))$ if and only if $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$

(The proof is provided in 2.2(b))

From (a) and (b), we can obtain that $p(n) = \Theta(n^k)$.

(d) $\because k > d \therefore \lim_{n\to\infty} \dfrac{p(n)}{n^k} = 0$ (approach 0 from the right side)

by L'Hôpital's rule, for any $\epsilon > 0$, we can find an n_0 s.t. $0 \leq \frac{p(n)}{n^k} < \epsilon$ when $n \geq n_0$.

$\rightarrow$ for any $c > 0$ we can find an n_0 s.t. $p(n) < c \cdot n^k$ when $n \geq n_0$

$\rightarrow p(n) = o(n^k)$

(e) $\because k < d \therefore \lim_{n\to\infty} \dfrac{n^k}{p(n)} = 0$ (approach 0 from the right side)

by L'Hôpital's rule, for any $\epsilon > 0$, we can find an n_0 s.t. $0 \leq \frac{n^k}{p(n)} < \epsilon$ when $n \geq n_0$.

$\rightarrow$ for any $c > 0$ we can find an n_0 s.t. $n^k < c \cdot p(n)$ when $n \geq n_0$

$\rightarrow p(n) = \omega(n^k)$

2.2. (10%)

(a) $f(n) = O(g(n))$

$dualarrow$We can find a $c > 0$ and an n_0 s.t. $f(n) \leq c \cdot g(n)$ when $n \geq n_0$

*dualarrow*We can find a $c' > 0$ and an n_1 s.t. $g(n) \geq c' \cdot f(n)$ when $n \geq n_1$ ($c' = \frac{1}{c}$)

dualarrow$g(n) = \Omega(f(n))$

(b) $f(n) = \Theta(g(n))$

*dualarrow*We can find a $c_1 > 0$, a $c_2 > 0$ and an n_0 s.t. $c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$ when $n \geq n_0$

*dualarrow*We can find a $c_1 > 0$, a $c_2 > 0$ and an n_0 s.t. $f(n) \geq c_1 \cdot g(n)$ when $n \geq n_0$ and $f(n) \leq c_2 \cdot g(n)$ when $n \geq n_0$

dualarrow$f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$

(c) $\because g(n)$ is an asymptopically positive function.

$\therefore$We can find an n_0 s.t. $g(n) > 0$ when $n \geq n_0$

$\because f(n) = O(g(n))$ $\therefore$We can find a c and an n_1 s.t. $f(n) \leq c \cdot g(n)$ when $n \geq n_1$

$\rightarrow f(n) \cdot g(n) \leq c \cdot g(n) \cdot g(n)$ when $n \geq \max\{n_0, n_1\}$

$\rightarrow f(n) \cdot g(n) = O((g(n))^2)$

(d) $\because f(n)$ is an asymptopically positive function.

$\therefore$We can find an n_0 s.t. $f(n) > 0$ when $n \geq n_0$

$\because g(n)$ is an asymptopically positive function.

$\therefore$We can find an n_1 s.t. $g(n) > 0$ when $n \geq n_1$

$\because f(n) = O(g(n))$ $\therefore$ We can find a c and an n_2 s.t. $f(n) \leq c \cdot g(n)$ when $n \geq n_2$

$\rightarrow f(n) \cdot f(n) \leq c \cdot f(n) \cdot g(n) \leq c \cdot (c \cdot g(n)) \cdot g(n)$ when $n \geq \max\{n_0, n_1, n_2\}$

$\rightarrow (f(n))^2 = O((g(n))^2)$

(e) Let $f(n) = 2 \cdot g(n)$, then $2^{f(n)} = 2^{2 \cdot g(n)} = 2^{g(n)} \cdot 2^{g(n)} = (2^{g(n)})^2$.

It's esay to see that $(2^{g(n)})^2 = O(2^{g(n)})$ doesn't necessarily hold for most cases.

$g(n) = \log_2 n$ is one of them.

Problem 3. Time and Space Complexities (15%)

3.1. (4%) $O(mlogn)$. Since the outer while-loop must run m times and the inner while-loop will run $logn$ times in worst case. All of other operations only runs in constant time.

3.2. (4%) $O(n + m)$. Since the for-loop must run n times and the while-loop must run m times. All of other operations only runs in constant time.

3.3. (3%) $O(K)$. Except the space cost of the input, the function uses three variables $C, i, searchnum$. The total size of them is $K + 1 + 1 = O(K)$.

3.4. (4%) *binary_search*. Since $m = O(1)$, we can reduce the time complexity of each function. ($T(binary\_search) = O(logn), T(count\_search) = O(n)$) In result of comparison, *binary_search* dominates *count_search* in time and space both.

***Problem* 4.** Stack and Queue (20%)

4.1. (5%) 5 4 3 1 2

4.2. (5%) 1 5 4 2 3

4.3. (6%) Set 1: stack. Set 2: queue. Set 3: neither.

4.4. (4%) Assume that the hidden data structure is a stack (queue) and simulate the operations. If all the output from stack (queue) fit the output of sequence in problem description, then the data structure may be a stack (queue).

***Problem* 5.** Scorched Toast (15%)

5.1. (5%)

```
Function brute-force{
  maxArea = 0
  for( upper = 1 ; upper ≤ M ; upper++ ){
    for( lower = upper; lower ≤ M ; lower++ ){
      for( left = 1 ; left ≤ N ; left++ ){
        for( right = left ; right ≤ N ; right++ ){
          flag = True
          for( i = upper ; i ≤ lower ; i++ ){
            for( j = left ; j ≤ right ; j++ ){
              if( B_ij == 0 ){
                flag = False
              }
            }
          }
          tmpArea = (lower - upper + 1) × (right - left + 1)
          if( flag == True and tmpArea > maxArea){
            maxArea = tmpArea
          }
        }
      }
    }
  }
  return maxArea
}
```

The first four loops, they cost $O(M^2N^2)$ to enumerate the four boundaries of sub-rectangles. To check the status of each sub-rectangle, it needs $O(MN)$ in the worst case. So the time complexity of this algorithm is $O(M^2N^2) \times O(MN) = O(M^3N^3)$.

5.2. (5%)

```
Function calculate-S{
  for( i = 1 ; i ≤ M ; i++ ){
    for(j = 1 ; j leq N ; j++ ){
      if( i == 1 ){
        S(i,j) = B_ij
      }else{
        if( B_ij == 1 ){
          S(i,j) = S(i − 1,j) + 1
        }else{
          S(i,j) = 0
        }
      }
    }
  }
}
```

5.3. (5%)

```
Function solve-by-row{
  maxArea = 0
  for( row = 1 ; row ≤ M ; row++ ){
    for( left = 1 ; left ≤ N ; left++ ){
      for( right = left ; right ≤ N ; right++ ){
        H = ∞
        for( j = left ; j ≤ right ; j++ ){
          if( S(row,j) < H ){
            H = S(row,j)
          }
        }
        tmpArea = H × (right - left + 1)
        if( flag == True and tmpArea > maxArea){
          maxArea = tmpArea
        }
      }
    }
  }
  return maxArea
}
```