Comp2013 Assignment1

23097386d FangChuming

February 2025

Question 1
Find the running time of the following recurrence in O-notation by using the Recursion-Tree method. Assume that the base case has problem size n=1. The derived O-notation should be as tight as possible; otherwise, marks will be deducted. (For example, $O(n^9)$ is tighter than $O(n^{10})$.

$$\begin{cases} O(1) & n = 1 \\ T(n) = 5T(n/3) + 2n & n > 1 \end{cases}$$

Solution:
Firstly, calculate the number of levels(k):

$$\frac{n}{3^k} = 1$$

by solving this equation, we get the value of k:

$$k = log_3 n$$

then, calculate the cost of i-th level(c):

$$c_i = 2 * 5^i * \frac{n}{3^i}$$

so the total cost:

$$c = 2n + \sum_{i=1}^{log_3 n} c_i = 2n * \sum_{i=0}^{log_3 n} (\frac{5}{3})^i$$

which can be simplified to obtain

$$c = 5n(\frac{5}{3})^{log_3 n} - 3n$$

through the logarithmic constant:

$$a^{log_b c} = c^{log_b a}$$

we can get the final cost:

$$c = 5n^{log_3 5} - 2n$$

so the time complexity of the algorithm is $O(n^{log_3 5})$

Question 2
For each statement below, identify if it is True or False, and provide the reason.
Statement 1 : If the worst-case running time of the algorithm A is $O(n^2)$, then the worst-case running time of A must be $\theta(n^2)$.
Conclusion: **False**
Reason: f(n) = n is a $O(n^2)$ funtion, assume there is a such scale $c \geq 0$ and $n_0 \geq 0$, if f(n) is $\Theta(n^2)$, then it must satisfy:

$$n > cn^2, \forall n \geq n_0$$

$$n(1 - cn) \geq 0$$

Let $n_1 = \frac{cn_0+c}{c}$ and $n_1 > n_0$

$$n_1(1 - cn_1) = n_1(-n_0) \leq 0$$

which violate the assumption, so that the scale is not exist, so an algorithm is $O(n^2)$, it not necessary it must be a $\Theta(n^2)$ algorithm.
Statement 2 : If, for all problem instances, the running time of the algorithm B is $\Omega(n^2)$, then the worst-case running time of B must be $\Omega(n^2)$.
Conclusion: **True**
Reason: for i case, there exist a constant $c_i \geq 0$ and $n_i \geq 0$ which can satisfy:

$$f(n) > c_i n^2, \forall n \geq n_i$$

so we can have a list $[c_1, c_2, ..., c_k]$ and $[n_1, n_2, ..., n_k]$which c_i, n_i corresponding to the i case, obviously from the list we can get a c_k which is the smallest among the c list and get n_k which is the biggest one of n list. that for all case,

$$f(n) > c_k n^2, \forall n \geq n_k$$

so that he worst-case running time of B must be $\Omega(n^2)$.
Statement 3 : If, for all problem instances, the running time of the algorithm C is $\Theta(n^{2.5})$, then the running time of C must be $O(n^{2.5})$
Conclusion: **True**
Reason: for i case, there exist $c_i \geq 0$ and $e_i \geq 0$ and $n_i \geq 0$ which satisfy:

$$e_i n^{2.5} \geq f(n) \geq c_i n^{2.5}, \forall n \geq n_i$$

so for all problem instances,If, for all problem instances, the running time of the algorithm C is $O(n^{2.5})$, similar to the statement 2, we can find a biggest e_k and biggest n_k so that for all the cases

$$f(n) \leq e_k n^{2.5}, \forall n \geq n_k$$

So that he running time of C must be $O(n^{2.5})$
Statement 4 : If, the best-case running time of the algorithm D is $O(nlogn)$,

it is possible that the running time of D is $\Omega(n^{1.5})$ for all problem instances.
Conclusion: **False**
Reason: for the best case, there exist $c_0 \geq 0$, and $n_0 \geq 0$, that

$$f(n)_{best} \leq c_0 nlogn, \forall n \geq n_0$$

and if the running time of f(n) is $\Omega(n^{1.5})$, that for all cases, there exist $c_1 \geq 0$ and $n_1 \geq n_0 \geq 0$ that

$$f(n) \geq c_1 n^{1.5}, \forall n \geq n_1$$

so we have G(n):

$$G(n) = c_1 n^{1.5} - c_0 nlogn \geq c_1 n(n^{0.5} - logn)$$

the increasing speed of $n^{0.5}$ is more quick than the $logn$, so there is no such n_1 can satisfy:

$$c_1 n(n^{0.5} - logn), \forall n \geq n_1$$

so it is impossible that the running time of D is $\Omega(n^{1.5})$ for all problem instances when the best-case running time of the algorithm D is $O(nlogn)$.

Question 3
(i)

$$F(x, y, A) = \sum_{i=1}^{n} A[i] \times \frac{x^i}{y^{n-i}}$$

Algorithm 1 Evaluate

Input: int x, int y, array A, length n
Output: F(x, y, A)

```
sum = 0, numerator = 1, i = 1, j = 1
molecular = 1
while i < n do
  numerator <- numerator*y
  i <- i+1
end while
while j <= n do
  molecular <- molecular*x
  sum <- sum+ A[i]*molecular/numerator
  numerator <- numerator/y
end while
F(x, y, A) <- sum
```

(ii) Analyze the worst-case running time of your algorithm in O-notation in terms of n.
Solution:
for the part one, I use the iteration method to get y^{n-1}, so the time cost of this

part is $O(n)$, the second part I alse use the iteration method to travel throughout the elements of the array A and it running time is also $O(n)$, so the total running time is $O(n) + O(n)$ and it is $O(n)$.

Quesrion 4
Prove that 2^n is not $O(n^2)$.
Proof: Assume 2^n is $O(n^2)$, there exist $C_0 \geq 0$ and $n_0 \geq 0$ which satisfy

$$2^n \leq C_0 n^2, \forall n \geq n_0$$

now, let $G(n) = 2^n - C_0 n^2$

$$\frac{\partial G(n)}{\partial n} = n2^{n-1} - 2c_0 n = 0$$

we can get a extreme point

$$n_1 = log_2 C_0 + 2$$

take the second order derivative of G(n) and observe the class of extreme points of G(n):

$$\frac{\partial^2 G(n_1)}{\partial n^2} = n_1(n_1 - 1)2^{n_1 - 2} - 2C_o = (log_2 C_0 + 2)(log_2 C_0 + 1)C_0 - 2C_0 > 0$$

so n_1 is the local miniest point of G(n), and $\forall n \geq n_1$, $G(n) \geq 0$

$$2^n \geq C_0 n^2$$

which violate our assumption, so 2^n is not $O(n^2)$
Question 5
Analyze the worst-case running time of the following algorithm in O-notation. Show your steps. We consider the worst case, $argmin\, A[i] \geq argmax\, B[j]$, for

Algorithm 2 Algo

```
Input: Array A, Array B, length n
Output: Null
  for integer i <- i to n do
    for  integer j <- 1 to n do
      if (A[i] > B[j]) print A[i]+B[j]
      else break
    end for
  end for
```

the first for iteration, it will run n times, and in every iteration, the second itertion will run n times, so the time is $O(n) + O(n^2) + O(n^2)$, so the running time is $O(n^2)$