ICCS313: Assignment 2

Natthakan Euaumpon
natthakaneuaumpon@gmail.com
October 2019

1: Part1

(a)
$2T(\frac{n}{3}) + 1$
$a = 2, b = 3, d = 0$
$log_b a = log_3 2$
$log_3 2 > 0$
$T(n) = O(n^{log_3 2})$

(b)
$5T(\frac{n}{4}) + n$
$a = 5, b = 4, d = 1$
$logba = log_4 5$
$log45 > 1$
$T(n) = O(n^{log_4 5})$

(c)
$7T(\frac{n}{7}) + n$
$a = 7, b = 7, d = 1$
$log_b a = log_7 7$
$1 = 1$
$T(n) = O(nlogn)$

(d)
$9T(\frac{n}{3}) + n^2$
$a = 9, b = 3, d = 2$
$log_b a = log_3 9$
$2 = 2$
$T(n) = O(n^2 logn)$

(e)
$8T(\frac{n}{2}) + n^3$
$a = 8, b = 2, d = 3$
$log_b a = log_2 8$
$3 = 3$
$T(n) = O(n^3 logn)$

(f)
$T(n-1) + 2$
$(T(n-2) + 2) + 2$
$(T(n-3) + 2) + 2 + 2$
$(T(n-4) + 2) + 2 + 2 + 2$
.
.
.

$(T(0) + 2) + 2(n - 1)$
T(n-1) run n times so it is $2n$
$T(n) = O(n)$

(g)
$T(n - 1) + n^c$
$= (T(n - 2) + (n - 1)^c) + n^c$
$= (T(n - 3) + (n - 2)^c) + (n - 1)^c + n^c$
.
.
.
$= (T(0) + (n - (n + 1))^c) + 2^c + ... + n^c$
$= T(0) + 1^c + 2^c + ... + n^c$
$T(n) = 1^c + 2^c + ... + n^c, \forall n \geq 1$
$T(n) = n^{c+1}, \forall n \geq 1$
$T(n) \leq n^{c+1} whenc = 1and \forall n \geq n_0 = 1$
$T(n) = O(n^{c+1})$

(h)
$T(n - 1) + c^n$
$= (T(n - 2) + c^{n-1}) + c^n$
$= (T(n - 3) + c^{n-2}) + c^{n-1} + c^n$
.
.
.
$= (T(0) + c^{n(n-1)}) + c^2 + ... + c^n$
$= T(0) + c^1 + c^2 + ... + c^n$
$T(n) = c^1 + c^2 + ... + c^n$
$cT(n) = c^2 + c^3 + ... + c^{n+1}$
$cT(n) - T(n) = (c^2 + c^3 + ... + c^{n+1}) - (c^1 + c^2 + ... + c^n)$
$T(n) \cdot (c - 1) = c^{n+1} - c$
$T(n) = \frac{c \cdot (c^n - 1)}{c - 1}$
$\lim_{n \to \infty} \frac{\frac{c \cdot (c^n - 1)}{c - 1}}{c^n}$
$\lim_{n \to \infty} \frac{c \cdot (c^n - 1)}{c^n \cdot (c - 1)}$
$\lim_{n \to \infty} \frac{c^{n+1} - c}{c^{n+1} - c^n}$
$= 1$
$T(n) \in \Theta(c^n)$, therefore $T(n) \in O(c^n)$

(i)
$2T(n - 1) + 1$
$= 2(2T(n - 2) + 1) + 1$
$= 2(2(T(n - 3) + 1)) + 2 + 1$
.
.
.
$= T(0) + 1 + 2 + 4 + ... + 2^{n-1}$
$T(n) = 1 + 2 + 4 + ... + 2^{n-1}$
$2T(n) = 2 + 4 + ... + 2^n$
$2T(n) - T(n) = (2 + 4 + ... + 2^n) - (1 + 2 + 4 + ... + 2^{n-1})$
$T(n) = 2^n - 1$

$\lim_{n\to\infty} \frac{2^n}{2^n}$
$= 1$
$T(n) \in \Theta(2^n)$, therefore $T(n) \in O(2^n)$

(j)

$T(n^{\frac{1}{2}}) + 1$
$= (T(n^{\frac{1}{4}}) + 1) + 1$
$= (T(n^{\frac{1}{8}}) + 1) + 1 + 1$
.
.
.
$= T(n^{\frac{1}{2^k}}) + k$
n need to be greater than or equal to 2 to get $T(0)$.
So, $2 = n^{\frac{1}{2^k}}$
$\log_n 2 \frac{1}{2^k}$
$2^k = \frac{1}{\log_n 2}$
$k = \log_2 \frac{1}{log_n 2}$
$k = \log_2 \log_2 n$
$T(n) = \log_2 \log_2 n$
$\lim_{n\to\infty} \frac{\log_2 \log_2 n}{\log_2 \log_2 n}$
$= 1$
$T(n) \in \Theta(\log_2^2 n)$, therefore $T(n) \in O(\log_2^2 n)$

2: Part2

Algorithm A:
$T(n) = 3T(\frac{n}{3}) + O(n)$
By using master theorem:
$a = 3, b = 3, d = 1$
$\log_b a = log_3 3 = 1$
$1 = 1$
$= O(n \log n)$
Algorithm B:
$T(n) = T(n-1) + O(n)$
$T(n-1) + O(n)$
$= (T(n-2) + O(n)) + O(n)$
.
.
.
$T(n) = O(n^2)$
Algorithm C:
$T(n) = 2T(\frac{n}{3}) + O(n^2)$
By using master theorem:
$\log_b a = \log_3 2$
$\log_3 2 < 2$
$= O(n^2)$
Algorithm D:
$T(n) = 5T(\frac{n}{4}) + O(n)$
By using master theorem:

$\log_b a = \log_4 5$
$\log_4 5 > 1$
$= O(n^{\log_4 5})$
We should use algorithm d as it has lowest upper bound of time complexity.

3: Problem3

The function f(n) use n as an input and the loop will stop running when $n = 1$
This program calls itself 3 times each take $\frac{n}{2}$
The time complexity for printing is $O(1)$
This means:
$T(n) = 3T(\frac{n}{2}) + O(1)$
By using master theorem:
$\log_b a = \log_2 3$
$\log_2 3 > 1$
$= O(n^{\log_2 3})$

4: Problem4

(a) Brute Force Algorithm
Do nested for loop and keep the maximum occurrence.
1)Instantiate counter, max and current which is the type of that element in the array to keep track. Where at first we set the counter and max to be 0.
2)Loop through every element by using and out side iteration. For each element we loop through and entire list and count all the occurrence.
3)After we done one inside iteration we compare the counter with the max value. If it is greater then we make greater equal to counter and set current to that element. Then we reset counter and continue the loop until the outer loop finish.
4)After that we compare our max value to the length of the array divided by 2. If it is greater then we return that current.
The time complexity of this algorithm is $O(n^2)$ as instantiate element take O(n). As an inside iteration take $O(n)$and the out side run $O(n)$ times. The if statement and other implementation inside the loop take constant time. So the time complexity is $O(n \cdot n) = O(n^2)$

(b)$n \log n$ time complexity
Using divide and conquer method.
1)Divide the array into half, left and right.
2)Continue dividing the array until we can compare the element.
3)Recursively keep track of the element, checking the majority elements of each sub array and combine them.
4)There are two possible cases. The first case is when two sub-arrays, this means we can just return as they are the same. This is because the definition given is it need to be greater than half the length of the array. The second cases is when the sub-arrays has different majority element. This means we need to count and compare the majority element from both sub-aray to know what we need to return.
Time complexity:
$T(n) = 2T(\frac{n}{2}) + O(n)$
By using master theorem:
$\log_b a = \log_2 2 = 1$

$1 = 1$
$= O(n \log n)$

(c)linear time algorithm
1)Divide array A into $\frac{n}{2}$ sub arrays of A.
2)We only keep the array that the 2 element matches.
3)If the elements are the same, we return one of the element. Else we remove that sub-array. If the length is odd we do not discard it but the last element with out a pair will be considered as a unique element.
4)Continue paring the element until we have one element left by combining the sub-arrays, we search for the duplicate element. We can compare bt counting the majority element in the input array with our majority element if there is more than $\frac{n}{2}$ elements we return them.
$\frac{n}{2}$ element Proof:
When it is match there will be 1 element that is remove. As one element remove each time we will have $\frac{n}{2}$ element left.
Time complexity:
$T(n) = T(\frac{n}{2}) + O(n)$
Using master theorem:
$\log_b a = \log_2 1$
$\log_2 1 < d$
$= O(n^1)$
$= O(n)$

5: Problem 5

My account: Natthakan EUAUMPON
Email address use: natthakan.eua@student.mahidol.edu
My team name is Natthakan Euaumpon
I submit the Brute force algorithm first to check my main. Then I submit the divide and conquer method.