ICCS200: Assignment 5

Karom Vorasa-ngasil
karom.140598@gmail.com
24/10/2019

1: Hello, Definition

1.

$$\lim_{n\to\infty} \frac{n}{nlog\, n} = \lim_{n\to\infty} \frac{1}{logn} = 0 < \infty$$

2.proof by contradiction, we know that

$$\lim_{n\to\infty} \frac{d(n)}{f(n)} < \infty \wedge \lim_{n\to\infty} \frac{e(n)}{g(n)} < \infty$$

so we want to show that

$$\lim_{n\to\infty} \frac{d(n)e(n)}{f(n)g(n)} = \infty$$

however,

$$\lim_{n\to\infty} \frac{d(n)}{f(n)} \times \lim_{n\to\infty} \frac{e(n)}{g(n)} < \infty \times \infty$$

$$\lim_{n\to\infty} \frac{d(n)e(n)}{f(n)g(n)} < \infty$$

which contradict with the contradiction statement so this proposition is True.
3. the function is $1000 * (1 + 2 + 3 + ... + n) = 500(n)(n + 1)$ so the big-O notation is $O(n^2)$
4.

$$\lim_{n\to\infty} \frac{16n^2 + 11n^4 + 0.1n^5}{n^4} = \lim_{n\to\infty} \frac{16}{n^2} + \lim_{n\to\infty} 11 + \lim_{n\to\infty} 0.1n = 0 + 11 + \infty = \infty$$

so the function is not $O(n^4)$.

2: Poisoned Wine

Since we know that $log(n + 1)$ is $(O(log(n))$(just do simple L'Hospital) and we can represent number 1 to n by using $log(n+1)$ bits(from hint 1). As we know that we that each bit will have only one of 2 values(0 or 1), we can pick the bottle where 1 is present in each bit and combine them together. At this point we will have $log(n + 1)$ set of sample which represent each bit so we only use $log(n + 1)$ people to test each sample. After 30 days, we will see who have the symptom then we will set the bit that the tester represent as 1 and put everything together so we can have the exact number of poison bottle. For example if the tester that represent bit 2,5,7 have the symptom then we can conclude that the bottle 1010010=81 are the poisoned one.

3: How long does this take?

programA→since the for loop will divide n by half each time until the answer is less than 1 and in each loop we only take constant time for multiplication so the run time is $\theta(log(n))$
programB→since we multiply c by 3 until it surpass n and in each loop we only take constant time for multiplication so the run time is $\theta(log_3(n))$

4: Halving Sum

task 1
in n^{th} iteration of loop we will
allocate Y which cost $k_1 * \frac{z}{2^n}$
read data from X which cost $k_2 * \frac{z}{2^{n-1}}$
arithmetic operation take $k_2 * \frac{z}{2^n}$
put the value into Y cost $k_2 * \frac{z}{2^n}$
Assign Y to X cost some constant time $\boldsymbol{C}$.
so the answer is $(\frac{k_1}{2^n} + \frac{k_2}{2^{n-1}} + \frac{k_2}{2^n} + \frac{k_2}{2^n})Z + C$

task 2
from task 1 we know the exact work in each iteration and we know that it will have $log(Z)$ loops and the finite geometric sum of n terms is $\frac{a_1*(1-r^n)}{1-r}$ so the total is

$$\sum_{i=1}^{log(n)} [(\frac{k_1}{2^i} + \frac{k_2}{2^{i-1}} + \frac{k_2}{2^i} + \frac{k_2}{2^i})Z + C]$$

$$(k_1 * \sum_{i=1}^{log(Z)} \frac{1}{2^i} + k_2 * \sum_{i=1}^{log(Z)} \frac{1}{2^{i-1}} + k_2 * \sum_{i=1}^{log(Z)} \frac{1}{2^i} + k_2 * \sum_{i=1}^{log(Z)} \frac{1}{2^i})Z + C * \sum_{i=1}^{log(Z)} 1$$

applying geometric sum formula.

$$(k_1 * \frac{\frac{1}{2} * (1 - (\frac{1}{2})^{log(Z)})}{1 - \frac{1}{2}} + k_2 * \frac{1 * (1 - (\frac{1}{2})^{log(Z)})}{1 - \frac{1}{2}} + k_2 * \frac{\frac{1}{2} * (1 - (\frac{1}{2})^{log(Z)})}{1 - \frac{1}{2}} + k_2 * \frac{\frac{1}{2} * (1 - (\frac{1}{2})^{log(Z)})}{1 - \frac{1}{2}})Z + C * log(Z)$$

since we know that $2^{log(Z)} = Z$ we will have

$$(k_1 * (1 - \frac{1}{Z}) + 2 * k_2 * (1 - \frac{1}{Z}) + k_2 * (1 - \frac{1}{Z}) + k_2 * (1 - \frac{1}{Z}))Z + C * log(Z)$$

$$k_1 * (Z - 1) + 4k_2 * (Z - 1) + C * log(Z)$$

$$(k_1 + 4k_2)Z + C * log(Z) - k_1 - 4k_2$$

since we know by simple application of L'Hospital that

$$\lim_{n \to \infty} \frac{log(n)}{n} = 0$$

so we can conclude that the run time of this program is $O(Z)$.

5: More Running Time Analysis

method1
It is a method to sort an int[] from max to min and we see 2 for loop that depend on the size of the input array so

Best-case: the size of input array is 1. The function will take $\theta(1)$ assuming that calling array.length take constant time.

Worst-case: as both the for loop is depend on the size of an array which we will assume to equal to n and assume the element of input array is sorted from min to max.
in the i^{th} iterations of for loop in method1 we will take $2C + (n-1-i) * 3C + 3C$ where C is some constant represent the time used for assign value to variable, create new variable and comparing the integer(assuming reading data from list take 0 time,if not, then it will just be some constant that we will ignore it later). So the total time taken is

$$\sum_{i=0}^{n-2}[2C + 3C(n-1-i) + 3C]$$

$$2C(n-1) + 3C(n-1) + 3C(\sum_{i=0}^{n-2}(n-1-i))$$

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

$$2C(n-1) + 3C(n-1) + 3C(\frac{(n)(n-1)}{2})$$

so we can pretty much ignore all the term which is a multiplication of 1 or n(due to the definition of θ that we will take the limit approach ∞, so all this terms will become 0 anyway) and concluded that this function has running time of $\theta(n^2)$.

method2
This is a method to check whether there are a key in array

Best-case: the key is exist at the beginning of the array, so it will take $\theta(1)$ assuming calling array.length,read value at index of array and comparing 2 integers all taking constant time(C).

Worst-case: the key is not present in the array. This mean we have to go through the for loop which each loop take C and we have $n-1$ iterator, so the total time is $Cn - C$ which correspond to $\theta(n)$

method3
Best-case: if the length of array is 0 then this method will take $\theta(1)$ time as the only thing to consider is the first for loop that going to do nothing and by assuming that calling array.length,declare variable sum, and doing arithmetic calculation all taking constant time(C) this method will take $2C + 97$(97 are from the for loop that have nothing to do as n=0).

Worst-case: assume the input array has length n
This method will take $2C + 97 * (2n-1) * (log(4n)) * C$ since from second term we know that this is not going to be $\theta(1)$ for sure so we can ignore the "$2C$" term. Then, I just guess that this function is $\theta(n * log(n))$ and going to prove it.

$$\lim_{n\to\infty} \frac{97 * (2n-1) * (log(4n)) * C}{n * log(n)}$$

$$\lim_{n\to\infty} \frac{194n * log(4n) * C - 97 * log(4n) * C}{n * log(n)}$$

$$\lim_{n\to\infty} \frac{194n * log(4n) * C}{n * log(n)} - \lim_{n\to\infty} \frac{97 * log(4n) * C}{n * log(n)}$$

since $log(4n) = log(n) + 2$ we can replace it in the calculation and get,

$$\lim_{n\to\infty} \frac{194n * log(n) * C}{n * log(n)} + \lim_{n\to\infty} \frac{194n * 2 * C}{n * log(n)} - \lim_{n\to\infty} \frac{97 * log(n) * C}{n * log(n)} + \lim_{n\to\infty} \frac{97 * 2 * C}{n * log(n)}$$

$$\lim_{n\to\infty} 194 * C + \lim_{n\to\infty} \frac{194 * 2 * C}{log(n)} - \lim_{n\to\infty} \frac{97 * C}{n} + 0$$

$$= 194 * C$$

which is some constant that not 0 so my guess is correct and this method is run at $\theta(n * log(n))$

6: Recursive Code

halvingSum
this problem size will be the length of input array(n).
In this method we have a for loop which has $\frac{n}{2}$ iteration and we call the method again with the input size of $\frac{n}{2}$ so the recurrence relation will be

$$T(n) = T(\tfrac{n}{2}) + \tfrac{n}{2} \text{ with T(1)=1 and } n = 2^k | \exists k \in \mathbb{I}^+$$

By solving this recurrence relation we get that $T(2^k) = 2^k - 1 \to T(n) = n - 1$ so this method is run at $O(n)$.

anotherSum
this problem size will be the length of input array(n).
In this method we call the method again with input size of $n-1$ and call the Arrays.copyOfRange which take $O(n)$(This are from Stack Overflow.) so the recurrence relation will be

$$(n) = T(n-1) + n - 1 \text{ with T(1)=1}$$

So we can solve it and get that $T(n) = \frac{n(n-1)}{2} + 1$ so this method is run at $O(n^2)$

prefixSum
this problem size will be the length of input array(n).
In this method we call Arrays.copyOfRange to copy half of the input array twice+call the method with input size $\frac{n}{2}$ twice and have 2 for loop with $\frac{n}{2}$ iteration each so the recurrence relation will be

$$T(n) = 2 * T(\tfrac{n}{2}) + \tfrac{4n}{2} \text{ with T(1)=0 and } n = 2^k | \exists k \in \mathbb{I}^+$$

By solving this recurrence relation we get that $T(n) = 2n * log(n) \in O(n * log(n))$

7: Counting Dashes

$g(n) = a * f(n) + bn + c$ where $g(0) = 0$.
part i
$g(0) = a * f(0) + b * 0 + c$
$c = 0$
part ii
$af(n) + bn = 2(af(n-1) + b(n-1)) + n$
$af(n) + bn = 2af(n-1) + 2bn - 2b + n$
$a(f(n) - 2f(n-1)) - bn + 2b - n = 0$
$a - bn + 2b - n = 0 \rightarrow (-b-1)n + (a+2b) = 0$
$b = -1, a = 2$
part iii
now we get that $g(n) = 2 * f(n) - n$ and we know that the closed form of $f(n) = 2^n - 1$, so we can construct the close form of $g(n) = 2 * (2^n - 1) - n \rightarrow 2^{n+1} - 2 - n$
part iv
Theorem: If $g'(n) = 2^{n+1} - 2 - n$ and $g(n) = 2g(n-1) + n$ with $g(0) = 0$, then $g'(n)$ will equal to $g(n)$ for all $n \in \mathbb{I}^+$.
Predicate: $P(i) = g'(i)$ is equal to $g(i)$.
Base case: $P(0) \rightarrow 2^{0+1} - 2 + 0 = 0$
Inductive Hypothesis: If $P(n-1)$ is true,then $P(n)$ will also be true.
Inductive Step:
$g(n) = 2g(n-1) + n$ since $P(n-1)$ is true that mean $g(n-1) = g'(n-1)$ so we get

$$
\begin{aligned}
g(n) &= 2(2^{(n-1)+1} - 2 - (n-1)) + n \\
g(n) &= 2(2^n - n - 1) + n \\
g(n) &= 2^{n+1} - 2n - 2 + n \\
g(n) &= 2^{n+1} - n - 2 \\
g(n) &= g'(n)
\end{aligned}
$$

Since our predicate and base case are true we can conclude that our theorem hold.As a result, the closed form of $g(n)$ that we derived in part 3 are correct.