Kent Mark

2/21/2021

Com S 311 – Homework 1

Homework 1

1)

a) To prove $n^2 - 10n + 2 = O(n^2)$ using the asymptomatic notations –
The function $f(n) = O(g(n))$ if $\exists$ constant c and n_0
Such that,
$f(n) \leq c * g(n)$ for all $n \geq n_0$
So, $f(n) = n^2 - 10n + 2$
$n^2 - 10n + 2 \leq n^2 * c$ for $n \geq 1$, $1 - (10/n) + (2/n^2) \leq c$
$c \geq -7$
Therefore, $f(O) = O(n^2)$

b) Using asymptomatic notations again –
The function $f(n) = O(g(n))$ if $\exists$ constant c and n_0
Such that,
$f(n) \leq c * g(n)$ for all $n \geq n_0$
So, $f(n) = 2^{n \wedge 2}$
$2^{n \wedge 2} \leq c * 2^{n \wedge 2}$ -> $n \geq O$
$(2^{n \wedge 2}/2^{2n}) \leq c$
$2^{n \wedge 2 - 2n} \leq c$ -> $c \geq 1$

Therefore, $f(n) = O(2^{2n})$

c) Using asymptomatic notations again –
The function $f(n) = O(g(n))$ if $\exists$ constant c and n_0
Such that,
$f(n) \leq c * g(n)$ for all $n \geq n_0$
$n \log_2(n) \leq c * \log_{10}(n)$
$(n \log * n / \log 2) \leq c * (n \log(n) / \log(10))$
$n \log(n) \leq c * n \log(n) * (\log 2 / \log 10)$
$n \log(n) \leq c * n \log(n) * \log_{10}2$
$n \log(n) \leq c * n \log(n) * 0.3010$ -> $n \geq O$
$(1/0.3010) \leq c$
$c \geq 3.322$ Therefore, $n \log_2(n) = O(n \log_{10}(n))$

d) Using asymptomatic notations again –
The function $f(n) = O(g(n))$ if $\exists$ constant c and n_0

$n \log_2(n) \leq c * n$

$\log_2(n) \leq c$ -> $n \geq 0$

$c \geq 0$ Therefore, $n \log_2(n) = O(n)$

2) Alg1(A)

For i = 1 to n , constant number of operations

For j = n to 1, do

for j = k to 1 do

constant number of expectations

Here, (j = n) is greater than 1,

j > n

So for (j = n) to 1 to 1 do

For k = j to 1 do

Time complexity $(j = n) * j = j^2 = jn = j = n$

O(j = n) {j = n} with the upper loop also running through 1 to n

Alg2(A)

For i = n to 1 do

constant number of operations

i = i/2

So every time 1 is divided 2,

(n/1) + (n/2) + (n/4) + … log(n)

Time complexity = O(log(n))

3) My algorithm was written in C

```
# include<stdio.h>

int count(int &k, int left, int right, int element) { //assume that there is an int array k in main

int count = 0;
for(int i = left; i <= right; i++){
    if (k[i] == element){
    count++;
    }
}
return count;

}
```

```
int gme(int &k, int left, int right) {
if (left > right) {
return -1;
}
if (left == right) {
return k[left];
}
int mid = left + (right – left) / 2;
int lCnt = gme(k, left, right);
int rCnt = gme(k, mid + 1, right);
if (lCount == -1 && rCount != -1) {
	int n = count(k, left, right, rCount);
	if (n > (right – left + 1) / 2) {
	return rCount;
	}
	else {
	return -1;
	}
}
else if (rCount == -1 && lCount != -1) {
	int num = count(k, left, right, lCount);
	if (num > (right – left + 1) / 2) {
	return lCount;
	}
	else {
	return -1;
	}
}
```

```
else if (lCount != -1 && rCount != -1) {
int leftn = count(k, left, right, lCount);
int rightn = count(k, left, right, rCount);
        if (leftn > (right – left + 1) / 2)) {
        return lCount;
        }
        else if (rightn > (right – left + 1) / 2) {
        return rCount;
        }
        else {
        return -1;
        }
}
        else {
        return -1;
        }
}
```

Time Complexity: $T(n) = 2 * T(n/2) + O(n)$ or $O(n\log(n))$

4) I do not know how to solve this problem.

5) There are n levels in the tree

   Sum of work done at each level:

   $n(\log(n/2^0) + \log(n/2^1) + \log(n/2^2) + ... + \log(n/2^{\log n}))$

   We know that $\log(a/b) = \log a - \log b$.

   Therefore, $n((\log n - \log 2^0) + (\log n - \log 2^1) + (\log n - \log 2^2) + ... + \log n - (\log 2^{\log n}))$

   Since $\log 2^k = k \log(2)$

   We have:

   $n((\log n - 0) + (\log n - 1) + (\log n - 2) + ... + (\log n - \log n))$

   $n(\log n + (\log n - 1) + (\log n - 2) + ... + 2 + 1 + 0)$

we know $1+2+3+4+........(x-1) + x = x(x + 1)/2$ -> $x = \log n$

$n (\log n)(1 + \log n) / 2$

$O(n \log^2 n)$

6) The recurrence relation given is: $T(n) = a * T(n/4) + O(n)$ … (Eqn 1)

$O(n)$ can be written as some constant $c * n$

So Eqn 1 becomes: $T(n) = a * T(n/4) + c * n$ … (Eqn 2)

Subbing in $T(n/4)$ into Eqn 2 we get:

$T(n) = a * [a * T(n/16) + c * (n/4)] + c * n = a^2 * T(n/4^2) + c * n * [1 + a/4]$ (Eqn 3)

Subbing in Eqn 3 we get:

$T(n) = a^3 * T(n/4^3) + c * n * [1 + a/4+ (a/4)^2]$ … (Eqn 4)

After subbing in h the equation becomes:

$T(n) = a^h * T(n/4^h) + c * n * [1 + a/4+ (a/4)^2 + ... + (a/4)^{h-1}]$ … (Eqn 5)

For termination $T(n/4^h)$ will be equal to $T(1)$

As such, $n/4^h = 1$

$4^h = n$

$h = \log_4(n)$

Now we find the sum of:

$1 + (a/4) + (a/4)^2 + \ldots + (a/4)^{h-1} = ((a/4)^h - 1)/((a/4) - 1) = (4/4^h) * ((a^h - 4^h)/(a - 4)) =$

$(4/n) * ((a^h - n)/(a - 4))$

Subbing values into Eqn 5

$T(n) = n^{\log_4 a} + 4c * ((n^{\log_4 a} - n)/(a - 4))$

Disregarding constants

$T(n) = n^{\log_4 a}$

For a faster algorithm we do:

$\log_4 a < \log_2 3$

$(\log_2 a/ \log_2 4) < \log_2 3$

$\log_2 a < 2 \log_2 3$

$\log_2 a < \log_2 9$

The max value of a = 8