Kent Mark

2/21/2021

Com S 311 – Homework 1

Homework 1

1. 1. To prove n2 – 10n + 2 = O(n2) using the asymptomatic notations –

The function f(n) = O(g(n)) if Ǝ constant c and n0

Such that,

f(n) ≤ c \* g(n) for all n ≥ n0

So, f(n) = n2 – 10n + 2

n2 – 10n + 2 ≤ n2\*c for n ≥ 1, 1 – (10/ n) + (2/ n2) ≤ c

c ≥ -7

Therefore, f(O) = O(n2)

* 1. Using asymptomatic notations again –

The function f(n) = O(g(n)) if Ǝ constant c and n0

Such that,

f(n) ≤ c \* g(n) for all n ≥ n0

So, f(n) = 2n^2

2n^2 ≤ c \* 2n^2 -> n ≥ O

(2n^2/22n) ≤ c

2n^2 – 2n ≤ c -> c ≥ 1

Therefore, f(n) = O(22n)

* 1. Using asymptomatic notations again –

The function f(n) = O(g(n)) if Ǝ constant c and n0

Such that,

f(n) ≤ c \* g(n) for all n ≥ n0

n log2(n) ≤ c \* log10(n)

(n log \* n / log 2) ≤ c \* (n log(n) / log(10))

n log(n) ≤ c \* n log(n) \* (log 2 / log 10)

n log(n) ≤ c \* n log(n) \* log102

n log(n) ≤ c \* n log(n) \* 0.3010 -> n ≥ O

(1/0.3010) ≤ c

c ≥ 3.322 Therefore, n log2(n) = O(n log10 (n))

* 1. Using asymptomatic notations again –

The function f(n) = O(g(n)) if Ǝ constant c and n0

n log2(n) ≤ c \* n

log2(n) ≤ c -> n ≥ 0

c ≥ 0 Therefore, n log2(n) = O(n)

1. 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 = j2 = 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))

1. 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(nlog(n))

1. I do not know how to solve this problem.
2. There are n levels in the tree

Sum of work done at each level:

n(log(n/20) + log(n/21) + log(n/22) + ... + log(n/2log n))

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

Therefore, n((log n - log 20) + (log n - log 21) + (log n - log 22) + ... + log n - (log 2log n))

Since log 2k = 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 log2 n)

1. 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 = a2 \* T(n/42) + c \* n \* [1 + a/4] (Eqn 3)

Subbing in Eqn 3 we get:

T(n) = a3 \* T(n/43) + c \* n \* [1 + a/4+ (a/4)2] … (Eqn 4)

After subbing in h the equation becomes:

T(n) = ah \* T(n/4h) + c \* n \* [1 + a/4+ (a/4)2 + .... + (a/4)h-1] … (Eqn 5)

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

As such, n/4h = 1

4h = n

h = log4(n)

Now we find the sum of:

1 + (a/4) + (a/4)2 + … + (a/4)h – 1 = ((a/4)h – 1)/((a/4) – 1) = (4/4h) \* ((ah – 4h)/(a – 4)) =

(4/n) \* ((ah – n)/(a – 4))

Subbing values into Eqn 5

T(n) = nlog4 a + 4c \* ((nlog4 a – n)/(a – 4))

Disregarding constants

T(n) = nlog4 a

For a faster algorithm we do:

log4 a < log2 3

(log2 a/ log2 4) < log2 3

log2 a < 2 log2 3

log2 a < log2 9

The max value of a = 8