CS 375 HW1

Baptiste Saliba

February 28, 2019

"I have done this assignment completely on my own. I have not copied it, nor have I given my solution to anyone else. I understand that if I am involved in plagiarism or cheating I will have to sign an official form that I have cheated and that this form will be stored in my official university record. I also understand that I will receive a grade of 0 for the involved assignment for my first offense and that I will receive a grade of "F" for the course for any additional offense."

Question 1:

a.

$T(n) = 3T(n/4) + 4$

$a = 3$, $b = 4$, $f(n) = n$

Compare:

n to $n^{log_4 3}$ //Case 3

or

$n^{log_4 4}$ to $n^{log_4 3}$

$n \in \Omega(n^{log_4 3+(log_4 4-log_4 3)}), \epsilon = log_4 4 - log_4 3$
$n \leq cn \qquad , c = 1, n_0 = 1$

Regularity Condition:
$3(n/4) \leq cn$
$3n/4 \leq cn$
$3/4 \leq c < 1$

b.

$T(n) = 2T(n/4) + \sqrt{n}lg(n)$
$a = 2, b = 4, f(n) = \sqrt{n}lg(n)$

Compare:

$\sqrt{n}lg(n)$ to $n^{log_2(4)}$
$n^{1/2}lg(n)$ to $n^{1/2}$ //Case 2

$n^{1/2}lg(n) \in \Theta(n^{log_4(2)lg^k(n)})$ when $k \leq 0$
$n^{1/2}lg(n) \in \Theta(n^{log_4(2)lg^1(n)})$ when $k = 0$
$n^{1/2}lg(n) \in \Theta(n^{log_4(2)lg^1(n)})$

*A function bounds itself by definition so this must be true

$T(n) = \Theta(n^{log_4 2}lg^2 n)$

c.

$T(n) = 5T(n/2) + n^2 \qquad a = 5 , b = 2$
Compare:
$n^{log_5 2}$ to n^2
or
$n^{log_2 5}$ to $n^{log_2 4}$ //Case 1

$n^2 \in O(n^{log_2 5-\epsilon}) \qquad \epsilon > 0$

$$n^2 \in O(n^{log_2 5-(log_2 5-log_2 4)}) \qquad \epsilon = log_2 5 - log_2 4$$

$$n^2 \leq cn^2 \qquad , c = 1, n_0 = 1$$
$$T(n) = \Theta(n^{log_2 5})$$

Question 2:

$T(n)$ = $\Theta(1)$ if $n \leq 1$
$T(n/4) + T(3n/4) + n$ otherwise

$$T(n) \in \Omega(nlog_4 n)$$
$$T(n) \in O(nlog_{4/3} n)$$

$$T(n) = \Theta(nlog(n))$$

Question 3:

Use substitution to prove $T(n) = T(n-1) + n \in O(n^2)$

WTS that $T(n) \leq cn^2 \qquad \forall n \geq n_0$
Assume that $T(k) \leq ck^2$ for $k < n$ and prove $T(n) \leq cn^2$
=> $T(n) = T(n-1) + n \leq c(n-1)^2 + n = c(n^2 - 2n + 1) + n$
=> $cn^2 - 2cn + c + n = cn^2 - n(2c-1) + c \leq cn^2 \qquad n \geq 0$

for $n(2c-1) + c \geq$ to hold, we need
$2cn - n + c \geq 0$ => $2cn + c \geq n$
so we need $n \leq 2cn + c$ which can be satisfied when $c \geq \frac{n}{2n+1}$
for all $n \geq 1$
$n \geq 1$ and $c \geq 1/3$
Base Case: Suppose $T(1) = 1$.Then $1 = T(1) \leq c(1)^2 = c$
so $c \geq max(1, \frac{1}{3}) = 1$

Question 4: Ternary Search through non-descending array

a.
Divide: Split A into 3 separate array.

Array 1: left1 = left
right1 = left+(right-left)/3

Array 2: left2 = right1+1
right2 = left2+(right-left)/3

Array 3: left3 = right2+1
right3 = right

Conquer: Compare the right most element of each subarray.If the rightmost element of that subarray is greater than x (A[right]>x), use recursion and check that subarray. If the rightmost element is equal to x return that index, otherwise don't check that subarray.

Combine: Combine is trivial because you are only looking for an element. Return index if found otherwise return -1.

b.

```
int TernarySearch(x,A,left,right)
    left1 = left
    right1 = left+(right-left)/3
    left2 = right1+1
     right2 = left2+(right-left)/3
    left3 = right2+1
    right3 = right
    if(A[right1]>x)
        TernarySearch(x,A,left1,right1)
    else if(A[right2]>x)
        TernarySearch(x, A,left2,right2)
    else if(A[right3]>x)
        TernarySearch(x,A,left3,right3)
    else if(A[right1]==x) return right1;
    else if(A[right2]==x) return right2;
    else if(A[right3]==x) return right3;
    else
        return -1;
```

c.

$T(n) =$ $\Theta(1)$ when $n = 1$

$3T(n/3) + \Theta(1)$, otherwise

d.

$T(n) = T(n/3) + c$

$a = 1, b = 3, f(n) = c$

Compare: $n^{log_3(1)}$ to 1 //Case 2

$f(n) \in \Theta(n^{log_b(a)} log^k(n)), k \geq 0$
$f(n) \in \Theta(n^{log_3(1)} log^0(n)), k = 0$
$n^{log_3 1} = 1 \in \Theta(n^{log_3(1)} log^0(n))$
$1 \in \Theta(1)$

$\Theta(n^{log_b} log^{k+1}(n) = \Theta(n^{log_3 1} log^1(n))$
$T(n) = \Theta(log(n))$

Question 5:

a.

Divide: Pick a random pivot in your list and divide your list into 2 sublists. One being all elements less than or equal to your pivot and the other being all elements greater than your pivot.

Conquer: If the size of the "lesser" list is greater than or equal to k recursively call function selection on that list. Otherwise call function selection on the "greater" list with parameters (k-lesser.length, S). Base case: Once the size of lesser is equal to 1 return the value in lesser.

Combine: Combine is trivial because we are searching for an element.

b.

int Selection(k,S)
    int pivot = S[rand(0,S.length-1)]
    list lesser
    list greater

```
for(i=0;i<S.length;i++)
    if(S[i] ≤ pivot)
        lesser.pushback(S[i])
    else
        greater.pushback(S[i])
if(lesser.length ==1) return lesser[0]
else if(lesser.length ≥ k)
    return selection(k, lesser)
else
    return selection(k-lesser.length, greater)
```

c.

Worst case: You pick the largest element in the list and only remove one element at a time until you get to k. This is highly unlikely but could happen and would result in $T(n) = \Theta(n^2)$

Best case: When you pick your random pivot, this pivot splits the list in half every time.

Question 6:

$$\sum_{i=0}^{lg(n)} \frac{n}{lg(\frac{n}{2^i})} = n\sum_{i=0}^{lg(n)} \frac{1}{lg(n) - lg_2(2^i)} = n\sum_{i=0}^{lg(n)} \frac{1}{lg(n) - i} = n\sum_{i=0}^{lg(n)} (lg(n) - i)^{-1}$$

Harmonic Series
def:

$$\sum_{i=1}^{k} \frac{1}{n} > \int_{1}^{k+1} \frac{1}{x} dx = ln(k+1)$$

$ln(lg(n) - 1 + 1) = ln(lg(n)) * n$

$T(n) = \Theta(nln(lg(n)))$