Bryan Burkhardt - XMV643
CS3343 Section 01
15 Sep 2017

CS3343 Analysis of Algorithms Fall 2017

Homework 2

Due 9/15/17 before 11:59pm (Central Time)

1. Master theorem (6 points)

Use the master theorem to find tight asymptotic bounds for the following recurrences. If the master theorem cannot be applied, state the reason and give an upper bound (big-Oh) which is as tight as you can justify. Justify your answers (by showing which case of the master theorem applies, and why.)

(1) $T(n) = 9T(n/3) + \sqrt{n}$
Solution:
$a = 9, b = 3, f(n) = \sqrt{n}$
$n^{\log_b a} = n^{\log_3 9} = n^2$
$Case1:$
$\sqrt{n} \in O(n^{2-\varepsilon}), \varepsilon = 1.5$
$\sqrt{n} \in O(n^{0.5})$
$\sqrt{n} \in O(\sqrt{n})$ Trivially True
$\Rightarrow \boxed{T(n) \in \Theta(n^2)}$

(2) $T(n) = 2T(n/2) + n \log n$
Solution:
$a = 2, b = 2, f(n) = n \log n$
$n^{\log_b a} = n^{\log_2 2} = n$
Master Theorem does not apply so we need to manipulate the function so it does.

$2T(n/2) + n \log n \leq 2T(n/2) + n * n = 2T(n/2) + n^2 \qquad n \geq logn$
$a = 2, b = 2, f(n) = n^2$
$n^{\log_b a} = n^{\log_2 2} = n$
$Case3:$
$n^2 \in \Omega(n^{1+\varepsilon}), \varepsilon = 1$
$n^2 \in \Omega(n^2)$ Trivially True
Now Show: $af(n/b) \leq cf(n)$ for $c < 1, n \geq logn$
$2f(n/2) \leq cn^2$
$2f(n/2) = 2(n^2/4) = (n^2/2)$

$\boxed{(n^2/2) \leq cn^2 \text{ when } c = (1/2) \text{ and } n \geq logn}$

$\Rightarrow \boxed{T(n) \in \Theta(n^2)}$

(3) $T(n) = 36T(n/6) + n^2$
Solution:
$a = 36, b = 6, f(n) = n^2$
$n^{\log_b a} = n^{\log_6 36} = n^2$
$Case2:$
$n^2 \in \Theta(n^2)$ Trivially True
$\Rightarrow \boxed{T(n) \in \Theta(n^2 \log n)}$

(4) $T(n) = T(2n/5) + n$
Solution:
$a = 1, b = (5/2), f(n) = n$
$n^{\log_b a} = n^{\log_{(5/2)} 1} = n^0 = 1 = \text{constant}$
$Case3:$
$n \in \Omega(n^{0+\varepsilon}), \varepsilon = 1$
$n \in \Omega(n)$ Trivially True
Now Show: $af(n/b) \leq cf(n)$ for $c < 1$
$1f(2n/5) = (2n/5) \leq cn$

$\boxed{(2n/5) \leq cn \text{ when } c = (2/5) < 1}$

$\Rightarrow \boxed{T(n) \in \Theta(n)}$

(5) $T(n) = T(2n/3) + T(n/3) + n$
Solution:
Master Theorem will not work because $T(n)$ contains two recurrences so we must manipulate the function to fit the master theorem:
$T(2n/3) + T(n/3) + n \leq T(2n/3) + T(2n/3) + n$
$= 2T(2n/3) + n$
$a = 2, b = (3/2), f(n) = n$
$n^{\log_b a} = n^{\log_{(3/2)} 2}$
$1.5^x = 2, x > 1$ therefore:
$n^{\log_{(3/2)} 2} > n > f(n)$
$Case1:$
$n \in O(n^{(\log_{(3/2)} 2) - \varepsilon}), \varepsilon = 0.01$
$n \in O(n^{(\log_{(3/2)} 2) - 0.01})$
$n \leq (n^{(\log_{(3/2)} 2) - 0.01})$
$\Rightarrow \boxed{T(n) \in \Theta(n^{\log_{(3/2)} 2})}$

(6) $T(n) = T(n-2) + n$

Master Theorem cannot be applied because function is in the form: $T(n-x)$

$T(0) = 0$
$T(2) = T(0) + 2 = 0 + 2 = 2$
$T(4) = T(2) + 4 = 2 + 4 = 6$
$T(6) = T(4) + 6 = 6 + 6 = 12$

$T(6) = T(4) + 6 = T(2) + 4 + 6 = T(0) + 2 + 4 + 6 = 0 + 2 + 4 + 6 = 12$
$= \sum_{i=1}^{(n/2)} 2i = 2\sum_{i=0}^{(n/2)} i$, Apply arithmetic series
$= 2(\frac{((n/2)+1)(n/2)}{2})$
$= ((n/2) + 1)(n/2)$
$= (n^2/4) + (n/2)$
$\Rightarrow \boxed{T(n) \in O(n^2)}$

2. Recursive Program (5 points)

Consider the following recursive function for $n \geq 0$:

Algorithm 1 int recFunc(int n)

```
//Base Case:
**if** $n <= 2$ **then**
    return n;
**end if**
//Recursive Case:
$i = 0$;
**while** $i < 100$ **do**
    print("Hello!");
    $i = i + 1$;
**end while**
int $a$ = 2*recFunc($n/2$); //Integer division
$j = 0$
**while** $j < a$ **do**
    print("Bye!");
    $j = j + 1$;
**end while**
return $a$;
```

(1) Use strong induction to prove the value returned by recFunc(n) is $\leq n$.

Solution:

Base case:

if $n \leq 2$, return n

$n = 2$, return 2

Inductive step:

Assume: $recFunc(x) \leq x$ for $2 \leq x < n$

Prove: $recFunc(n) \leq n$

$LHS = recFunc(n) = 2 * recFunc(n/2)$

$2 * recFunc(n/2) \leq 2 * (n/2)$ by the inductive hypothesis

$2 * (n/2) = n = RHS$

$\Rightarrow \boxed{recFunc(n) \leq n}$

(2) Set up a runtime recurrence for the runtime $T(n)$ of this algorithm.

$T(n) = a * T(n/b) + f(n)$

$a = 1$ because the recursive function is only called once in the algorithm.

$b = 2$ because the value passed to the recursive funcion is $(n/2)$.

$f(n) = n$ because the first while loop runs 100 times which is a constant and the second while loop runs a times which is $\leq n$ since we proved the value assigned to a is $\leq n$ in part (1). Also, there are zero nested loops.

$\boxed{T(n) = T(n/2) + n}$

(3) Solve this runtime recurrence using the master theorem.

$T(n) = T(n/2) + n$

$a = 1, b = 2, f(n) = n$

$n^{\log_b a} = n^{\log_2 1} = n^0 = 1$

$Case3:$

$n \in \Omega(n^{0+\varepsilon}), \varepsilon = 1$

$n \in \Omega(n)$ Trivially True

Now Show: $af(n/b) \leq cf(n)$ for $c < 1$

$1 * f(n/2) \leq cn$ for $c < 1$

$(n/2) \leq cn$ for $c = (1/2) < 1$

$\Rightarrow \boxed{T(n) \in \Theta(n)}$

3. Big-Oh Induction (4 points)

Use strong induction to prove that $T(n) \in O(\log n)$ where $T(1) = 90$ and $T(n) = T(n/2) + 10$ (for $n \geq 2$).

$T(n) \leq c * \log n$
Base case:
$T(2) = T(2/2) + 10 = T(1) + 10 = 90 + 10 = 100$
Find c value to make true:
$c * \log n = c * \log_2 2 = c * 1 = c$
$100 \leq c$ so choose $c \geq 100$

Inductive step:
Assume: $T(x) \leq c \log x$ for $2 \leq x < n$
Prove: $T(n) \leq c \log n$

$LHS = T(n) = T(n/2) + 10$
$T(n/2) \leq c \log(n/2) + 10$ by the inductive hypothesis
$c \log(n/2) + 10 = c(\log_2 n - \log_2 2) + 10 = c(log_2 n - 1) + 10 = c \log_2 n - c + 10$
$-c + 10$ must be negative because $c \geq 100$ Therefore $c \log_2 n - c + 10 \leq c \log_2 n$
$RHS = c \log_2 n$
$\Rightarrow \boxed{T(n) \in O(\log n)}$

4. Recursive Algorithm and Analysis (6 points)

Suppose company X has hired you as a software engineer.

Company X is desperate to improve the run time of their software. Rather than allow you to use the standard binary search algorithm, company X demands you create a new version. Since recursively dividing the array into two parts is so efficient for searching, the bigwigs at company X conclude dividing the array into three parts will be even more efficient.

This new "3-ary" search divides the given sorted array into 3 equal sized sorted subarrays. It finds which subarray will contain the given value and then recursively searches that subarray. If the given value is in the array you should return true. Otherwise you should return false.

(1) (2 points) Write up this new recursive algorithm using pseudocode. (Hint: For simplicity you can assume the number of elements in the given sorted array, n, is a power of 3)

Algorithm 2 int binarySearch(int $A[1 \ldots n]$, int val)

```
if $A[n/3] == val$ then
    //Found val at the current location
    return True;
else if $A[2(n/3)] == val$ then
    //Found val at the current location
    return True;
else if $n == 2$ then
    //val is not in the array
    return False;
else if $A[n/3] >= val$ then
    //val is in first third of the array
    return binarySearch($A[1 \ldots n/3]$, val);
else if $A[n/3] <= val$ and $A[2(n/3)] >= val$ then
    //val is in second third of the array
    return binarySearch($A[n/3 \ldots 2(n/3)]$, val);
else$A[2(n/3)] <= val$
    //val is in last third of the array
    return binarySearch($A[2(n/3) \ldots n]$, val);
end if
```

(2) (1 point) Create a recurrence to represent the worst case run time of our new "3-ary" search from the pseudocode.
(Hint: The worst case run time occurs when the given value is not in the array)

$$\boxed{T(n) = 3T(n/3) + d_n}$$

(3) (2 points) Use master theorem to solve your above recurrence relation.
Solution:
$T(n) = 3T(n/3) + d_n$
$a = 3, b = 3, f(n) = d_n$
$n^{\log_b a} = n^{\log_3 3} = n$
$Case1:$
$d_n \in O(n^{1-\varepsilon}), \varepsilon = 1$
$d_n \in O(d_n)$ Trivially True
$\Rightarrow \boxed{T(n) \in \Theta(n)}$

(4) (1 point) Compare this asymptotic run time to that of binary search. Does our new algorithm perform significantly better than binary search?

(5) (0 points - Just for fun) Try comparing the constants of binary search

and “3-ary” search. You can use the change of base formula for log to convert both to $\log_2 n$.