

CIS121 - Fall 2009

Homework 2 Friday, Sep 25

Due: Wednesday, September 30 at 10:00am in Levine 459

You don't need to type up this assignment if you don't want to but keep in mind that illegible answers will not be graded. Please write your name and lab section at the top of each page, then staple your pages together securely. Turn in your assignment to the course administrator, Charity Payne, in Levine 459.

Problem 1: 25 points

Solve the recurrence relation $T(n) = T(\sqrt{n}) + 1$ for $n = 2^k \geq 2$ and $T(2) = 1$.

Solution: For $n = 2^m$ the recurrence has the form $T(2^m) = T(2^{m/2}) + 1$. Let $S(m) = T(2^m)$:

$$\begin{aligned} S(m) &= S(m/2) + 1 \\ S(m/2) &= S(m/4) + 1 \\ S(m/4) &= S(m/8) + 1 \\ &\vdots \\ S(2) &= S(1) + 1 \\ S(1) = T(2^1) &= 1 \end{aligned}$$

Adding the above equations gives $S(m) = \log(m) + 1$.

Therefore $T(n) = T(2^m) = S(m) = \log(m) + 1 = \log(\log(n)) + 1$.

Problem 2: 25 points

Solve the recurrence relation and calculate a Θ -bound.

$$T(n) = T(n/2) + n^2 \text{ and } T(1) = 1$$

Homework 2

cis121

Solution: For $n = 2^m$ the recurrence has the form $T(2^m) = T(2^{(m-1)}) + 2^{2m}$.

$$\begin{aligned}T(2^m) &= T(2^{(m-1)}) + 2^{2m} \\T(2^{(m-1)}) &= T(2^{(m-2)}) + 2^{2m-2} \\T(2^{(m-2)}) &= T(2^{(m-3)}) + 2^{2m-4} \\&\vdots \\T(2^1) &= T(2^0) + 2^0\end{aligned}$$

Adding the above equations gives $T(2^m) = T(1) + (1 + 2^2 + \dots + 2^{2m-2} + 2^{2m})$. Using the sum of a geometric series $\sum_{i=0}^m 2^{2i} = \sum_{i=0}^m (2^2)^i = \sum_{i=0}^m 4^i = \frac{4^{m+1} - 1}{4 - 1} = \frac{4 \cdot 4^m - 1}{3} = \frac{4 \cdot (2^m)^2 - 1}{3} = \frac{4n^2 - 1}{3}$
 $T(n) = 4n^2/3 - 1/3$. Hence $T(n)$ is $O(n^2)$.

Problem 3: 25 points

Compute the big- Θ for the following methods. If recursive, then write the recurrence relation and solve it. Pay attention to operations on strings (which are really character arrays, concatenation takes $\Theta(\text{string.length})$). State any assumptions you make about the running time of basic operations.

```
public static String method1(int n) {
    String s = "a";
    for (int i = 0; i < n; i++)
        s = s + s;
    return s;
}
```

Solution: Both the initialization and return take constant time c_1 . The concatenation inside the loop takes $2 * s.length$ operations. The loop has n iterations (each with some operations c_2), more importantly in each iteration the size of s is *doubled*.

Therefore summing the concatenation over all iterations results in $2 + 4 + 8 + \dots + 2^n$ operations. Overall we have $c_1 + nc_2 + (2 + 4 + \dots + 2^n) = c_1 + c_2 + 2^{n-1} - 2$ which is $\Theta(2^n)$.

```
public static String method2(int n) {
    if (n == 0) return "a";
    else return method2(n-1) + method2(n-1);
}
```

Solution: Let $T(n)$ symbolize the running time of `method2`, and $L(n)$ the length of the result. For $n = 0, T(0) = c$ and $L(0) = 1$. Otherwise the function contains 2 recursive calls and concatenates their result, or $T(n) = 2T(n-1) + 2L(n-1)$, similarly $L(n) = 2L(n-1)$. Given $L(n) = 2L(n-1)$ and $L(0) = 1$ results in $L(n) = 2^n$. Substituting that back into $T(n)$ gives $T(n) = 2T(n-1) + 2L(n-1) = 2T(n-1) + 2 * 2^{n-1} = 2T(n-1) + 2^n$.

$$\begin{aligned} T(n) &= 2T(n-1) + 2^n \\ T(n-1) &= 2T(n-2) + 2^{n-1} \\ &\vdots \\ T(1) &= 2T(0) + 2 \\ T(0) &= c \end{aligned}$$

Multiplying each line by the appropriate factor.

$$\begin{aligned} T(n) &= 2T(n-1) + 2^n \\ 2T(n-1) &= 4T(n-2) + 2 * 2^{n-1} = 4T(n-2) + 2^n \\ &\vdots \\ 2^{n-1}T(1) &= 2 * 2^{n-1}T(0) + 2^{n-1} * 2 = 2^n T(0) + 2^n \\ 2^n T(0) &= 2^n c \end{aligned}$$

Adding the above equations gives $T(n) = 2^n c + n2^n$ which is $\Theta(n2^n)$

Problem 4: 25 points

Analyze the following code and give a big-Theta characterization of the running time of `mystery`.

```
static void mystery(int[] a) {
    aux(a, 0, a.length-1);
}

static void aux(int[] a, int bot, int top) {
    if (bot >= top) return;
    int x = a[top];
    int j = bot;
    int k = top-1;
    while (j <= k) {
        while (j<=k && a[j]<=x) j++;
        while (k>=j && a[k]>=x) k--;
    }
}
```

Homework 2

cis121

```
    if (j < k) {
        int tmp = a[j]; a[j] = a[k]; a[k] = tmp;
    }
}
a[top] = a[j]; a[j] = x;
aux(a, bot, j-1);
aux(a, j+1, top);}
```

Solution: As in the analysis in the lecture notes, we find a Big-Theta characterization of the running time of `aux(a,bot,top)` in terms of `bot` and `top`. For the code before the recursive calls we observe that only the while loops are not constant time. During the execution of the while loops, j increments and k decrements but we always have $j \leq k$. For each incrementation of j or decrementation of k , only a constant amount of steps are taken. Therefore the while loops and in fact everything in `aux` except the recursive calls takes $O(\text{top} - \text{bot})$.

For the recursive calls we realize that the worst case is when the array segment between `bot` and `top` splits as unevenly as possible, i.e, when before the recursive calls j ends up being `bot + 1` or `top - 1` for every call of `aux`. We set up a recurrence relation for $T(n)$, the worst-case running time for `aux(a, bot, top)` where $n = \text{top} - \text{bot}$:

$$T(n) = T(n-1) + T(0) + c_1n + c_2$$

where c_1, c_2 are constants.

$$T(n) = T(n-1) + c_1n + (c_2 + T(0))$$

Solving this we get

$$T(n) = T(0) + c_1 \sum_{i=1}^n i + (c_2 + T(0))n = c_1 \frac{n^2 + n}{2} + (c_2 + T(0))n + T(0)$$

Since $T(0)$ is $O(1)$ we obtain that $T(n)$ is $O(n^2)$. We conclude that `mystery(a)` runs in time $O(n^2)$ where $n = a.length$.