Homework 2

Lauren Lyons

1▷ Loop Invariant: $S_i = x$ is not equal to any of the first i elements of the array

Initialization: Before the start of the loop where $i = 0$, we know that x is not equal to the first 0 elements since $i = 0$. This is true because x cannot equal nothing.

Maintenance: Before the start of the j^{th} iteration, where $j = i+1$, x is not equal to any of the first i elements of the array. During the j^{th} iteration, the if statement $x = A[i]$ is computed, and if it is true, then i is returned and the loop terminates. If it is not true, then i is computed as $i+1$, so $j+1 = i+1+1$. The next iteration then requires x to not equal any of the first $j+1$ elements of the array. This will still hold true because the loop checked one element to see if it was the element x and then we added 1 to i if it wasn't. This means that we will have always checked all of the numbers before the j^{th} element in the array for x. When we iterate $j-1$ times where each iteration checks one element, we will have checked $j-1$ elements for x. Before the next iteration j, we will have checked i elements. This is true because $i = j-1$. Therefore, x is not equal to any of the first i elements. For example, the 1st iteration ($j=1$) of the loop will check the first element of the array. So when we iterate to the 2nd iteration ($j=2$) and before we enter the loop, we will have checked one element, $j=2$. $i = j-1$, $i = 2-1$, $i = 1$. So, x is not equal to any of the first 1 elements of the array.

Termination: Since initialization shows that the LI will be true before the loop begins and Maintenance shows

Lauren
Lyons
Pg2

that it will continue to be true before each iteration of j, so it must be true before the last iteration, the n^{th} iteration. The loop ends after the n^{th} iteration, so before the $(n+1)^{th}$ iteration, the LI must be true. Before the j^{th} iteration, i will equal j-1. Therefore, before the $(n+1)^{th}$ iteration, i will equal n because $j = n+1$ & $i = j-1 \Rightarrow i = n+1-1 = n$. The LI says that x will not equal to any of the first n elements of the array, since $i = n$. This means that the algorithm works correctly, because if the loop is never terminated by a a found x element, it will go through all n elements of the array to check.

2. Least Complex (Big-Oh)

$\frac{1}{n}$
2^{100}
$\log \log n$
$\sqrt{\log n}$
$\log^2 n$
$n^{0.01}$
$\lceil \sqrt{n} \rceil$
$3n^{0.5}$ } Θ
$\frac{2^{\log n}}{\sqrt{5n}}$ } Θ

$n \log_4 n$
$\frac{1}{2} n \log n$
$\lfloor 2n \log^2 n \rfloor$ } Θ
$4n^{3/2}$
$4^{\log n}$
$n^2 \log n$
n^3
2^n
$\frac{4^n}{2^{2n}}$ } Θ

Most Complex (Big-Oh)

3 ▷ A method for finding both the minimum and maximum of n numbers.

▷ We will have 2 variables, min and max. Min is initialized to the first element, A[0], and max is initialized to the second element, A[1],. We iterate through the array in pairs, with i and i+1 giving A[i] & A[i+1]. so each iteration will make i = i+2, initially starting with i=2. We first compare A[i] to A[i+1]. If A[i] is greater than A[i+1], then A[i] is a candidate for max and A[i+1] is a candidate for min. We would then compare A[i] with max and replace if its greater than max. Then we would compare A[i+1] with min and replace if it's less than min. If A[i+1] is greater than A[i], then we would do the same, just switching the comparisons for A[i] and A[i+1]. Each iteration will have 3 comparisons and it iterates by 2, resulting in less than $\frac{3n}{2}$ comparisons since we initialize max to A[0], min to A[1], and i to 2.

Lauren
4/09
pg 4

4 What is wrong with this proof?

- $F(2)=2$ is not $O(2)$, it is $O(1)$. Since 2 is a constant, it will have a Big-Oh of 1 ($O(1)$). 2 is not a complexity used to describe Big-Oh. Then in the inductive step, we cannot apply changes to the Big-Oh complexities like we do to $F(n) \to F(n-1)$, when $F(n)$ goes to $F(n-1)$, its time complexity would stay $O(n)$ and not change to $O(n-1)$, as that is not a complexity used to describe Big-Oh either. The same applies for $F(n-2)$ and $O(n-2)$, it will be $O(n)$. Big-Oh is not a recursive function call like $F(n)$ and all the Fibonacci calls, it is a label/evaluation of the algorithm. Likewise, we cannot do $O((n-1)+(n-2))$ as it cannot be transformed that way. Overall, we get the correct time complexity through an incorrect inductive proof.

5 a) What does the recursive algorithm above compute?

- This algorithm will return the minimum element in the array passed to it.

b) $T(n) = 7 \quad n=1$

$T(n) = T(n/2) + T(n/2) + 17 \quad n \geq 2$

```
if i = j                                    cost   reasoning
    return A[i]                             3      read i, read j, compare
else                                        4      read i, find mem. loc., read value, return
    k = i + floor((j-i)/2)                  7      read i, read j, subtract, divide, floor, add, assign
    temp1 = Mystery(A[i...k])               1      assign
    temp2 = Mystery(A[(k+1)...j])           1      assign
    if temp1 < temp2                        3      read temp1, read temp2, compare
        return temp1                        2      read temp1, return
    else return temp2                       2      read temp2, return
```

5cont ▷ c)

Level	Level Number	Total #	Input size	Work done by each	Total work
Root	0	1	n	c	c
One level below	1	2	$n/2$	c	$2c$
Two levels below	2	4	$n/4$	c	$4c$
Above base case	$\lg n - 1$	$n/2$	2	c	$\frac{n}{2}c$
Base case	$\lg n$	n	1	c	nc

d) $c\sum_{i=0}^{\log n-1} 2^i = c\frac{2^{\lg n}-1}{2-1} = c\frac{n-1}{1} = c(n-1) = cn-c$

$T(n) = c\sum_{i=0}^{\log n-1} 2^i + cn = cn-c+cn = 2cn-c$

so $T(n) \Rightarrow O(n)$

6 ▷

Level	Level Number	Total #	Input size	Work done by each	Total work
Root	0	1	n	cn	cn
One level below	1	7	$n/8$	$cn/8$	$\frac{7}{8}cn$
Two levels below	2	49	$n/64$	$cn/64$	$\frac{49}{64}cn$
Above base case	$\lg n - 1$	$(\frac{n}{8})\log_8 7$	8	$8c$	$8c(\frac{n}{8})^{\log_8 7}$
Base Case	$\lg n$	$n^{\log_8 7}$	1	c	$cn^{\log_8 7}$

$\sum_0^\infty x^i = \frac{1}{1-x}; x<1$

$T(n) = cn\sum_0^{\infty}(7/8)^i + \Theta(n^{\log_8 7}) = \frac{cn}{1-7/8} + \Theta(n^{\log_8 7}) = 8cn + \Theta(n^{\log_8 7})$

so $T(n) \Rightarrow O(n)$

Lauren Lyons pg 6

7 Guess: $T(n) = O(n)$ $\quad T(n) = 3T(n/3) + 5 \quad T(1) = 5$

Prove: $T(n) \leq cn$
Base Case: $T(1) = 5 \leq c$ (if $c \geq 5$)
Inductive Hypothesis: $T(n/3) \leq cn/3$
Inductive Step: $T(n) \leq 3(\frac{cn}{3}) + 5$
$T(n) \leq cn + 5 \not\leq cn$ → Proof fails, change proof

⇓

Prove: $T(n) \leq cn - 5$
Base Case: $T(1) = 5 \leq c(1) - 5$ (if $c \geq 10$)
Inductive Hypothesis: $T(n/3) \leq \frac{cn}{3} - 5$
Inductive step: $T(n) \leq 3(\frac{cn}{3} - 5) + 5$
$\leq cn - 15 + 5$
$\leq cn - 10 \leq cn$ for all c

Values of c: $c \geq 10$

8 $T(n) = T(n/2) + T(n/4) + T(n/8) + T(n/8) + n \quad T(1) = c$

Recursion tree (recursive call : input size : workdone by each call)

Lvl
0 (1) $T(n) : n : n$

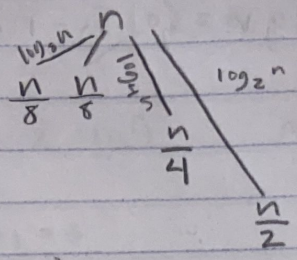
1 $T(\frac{n}{2}) : \frac{n}{2} : \frac{n}{2}$ $\quad T(\frac{n}{4}) : \frac{n}{4} : \frac{n}{4}$ $\quad T(\frac{n}{8}) : \frac{n}{8} : \frac{n}{8}$ $\quad T(\frac{n}{8}) : \frac{n}{8} : \frac{n}{8}$

2 $T(\frac{n}{4})$: $\frac{n}{4}$: $\frac{n}{4}$ $\quad T(\frac{n}{8})$: $\frac{n}{8}$: $\frac{n}{8}$ $\quad T(\frac{n}{16})$: $\frac{n}{16}$: $\frac{n}{16}$ $\quad T(\frac{n}{16})$: $\frac{n}{16}$: $\frac{n}{16}$ $\quad T(\frac{n}{8})$: $\frac{n}{8}$: $\frac{n}{8}$ $\quad T(\frac{n}{16})$: $\frac{n}{16}$: $\frac{n}{16}$ $\quad T(\frac{n}{32})$: $\frac{n}{32}$: $\frac{n}{32}$ $\quad T(\frac{n}{32})$: $\frac{n}{32}$: $\frac{n}{32}$ $\quad T(\frac{n}{16})$: $\frac{n}{16}$: $\frac{n}{16}$ $\quad T(\frac{n}{32})$: $\frac{n}{32}$: $\frac{n}{32}$ $\quad T(\frac{n}{64})$: $\frac{n}{64}$: $\frac{n}{64}$ $\quad T(\frac{n}{64})$: $\frac{n}{64}$: $\frac{n}{64}$ $\quad T(\frac{n}{16})$: $\frac{n}{16}$: $\frac{n}{16}$ $\quad T(\frac{n}{32})$: $\frac{n}{32}$: $\frac{n}{32}$ $\quad T(\frac{n}{64})$: $\frac{n}{64}$: $\frac{n}{64}$ $\quad T(\frac{n}{64})$: $\frac{n}{64}$: $\frac{n}{64}$

Lvl	total work
0	n
1	n
2	n

Lauren Lyons 097

8cont (2) Shape of Tree



(3) Shallowest Part: $\log_8 n$

(4) Deepest Part: $\log_2 n$

(5) Guess: $O(n \log n)$

9 $T(n) = T(n/2) + T(n/4) + T(n/8) + T(n/8) + n \quad T(1) = c$

Guess: $O(n \log n)$

Prove: $T(n) \leq dn \log n$

Base Case: $T(1) = c \leq d \log 1 \rightarrow 0?$ $c \leq 0?$ X

↓

Base Case: $T(2) = c \leq 2d \log 2 = 2d$ if $(d \geq \frac{c}{2})$

Inductive Hypothesis: $T(n/2) \leq d(\frac{n}{2}) \log(n/2)$

$T(n/4) \leq d(\frac{n}{4}) \log(n/4)$

$T(n/8) \leq d(n/8) \log(n/8)$

Inductive Step: $T(n) \leq \frac{dn}{2} \log(\frac{n}{2}) + \frac{dn}{4} \log(\frac{n}{4}) + \frac{2dn}{8} \log(\frac{n}{8}) + n$

$T(n) \leq \frac{dn}{2}(\log n - \log 2) + \frac{dn}{4}(\log n - \log_2 4)^2 + \frac{dn}{4}(\log n - \log_2 8) + n$

$\leq dn(\frac{2}{4}\log n + \frac{1}{4}\log n + \frac{1}{4}\log n - \frac{2}{4} - \frac{2}{4} - \frac{3}{4}) + n$

$\leq dn(\log n - 7/4) + n$

$\leq dn \log n - \frac{7dn}{4} + n \leq dn \log n$ (when $d \geq 4/7$)

b/c then $dn \log n - n + n \leq dn \log n$

10 (a) $T(n) = 2T(\frac{99}{100} n) + 100n \quad a=2 \quad b=\frac{100}{99} \quad f(n) = 100n$

$n^{\log_{\frac{100}{99}} 2} \approx n^{.69-\varepsilon} = O(f(n)) = O(n)$ Case 1 ✓

$T(n) = \Theta(n^{\log_{\frac{100}{99}} 2})$

Lauren Lyons pg 8

10 cont (b) $T(n) = 16T(\frac{n}{2}) + n^3 \lg n$ $a=16$ $b=2$ $f(n)=n^3 \lg n$

$n^{\log_2 16} = n^{4-\epsilon} > n^3 \lg n$ Case 1 ✓

$T(n) = \Theta(n^4)$

(c) $T(n) = 16T(\frac{n}{4}) + n^2$ $a=16$ $b=4$ $f(n)=n^2$

$n^{\log_4 16} = n^2 = n^2 \Leftarrow f(n)$ Case 2 ✓

$T(n) = \Theta(n^2 \log n)$

11 $T(n) = 2T(n-1) + 1$ $T(0)=1$

- Backwards

(1) 3 expansions

$T(n-1) = 2T(n-2) + 1$

1 $T(n) = 2(2T(n-2)+1)+1$

$= 4T(n-2) + 3$

$T(n-2) = 2T(n-3) + 1$

2 $T(n) = 4(2T(n-3)+1)+3$

$= 8T(n-3) + 7$

$T(n-3) = 2T(n-4) + 1$

3 $T(n) = 8(2T(n-4)+1)+7$

$= 16T(n-4) + 15$

$T(n) \Rightarrow O(2^n)$

(2) $\sum_{k=0}^{n} x^k = \sum_{k=0}^{n} 2^k = \frac{2^{n+1}-1}{2-1} = T(n)$

$T(n) = 2^{n+1} - 1$

(3) LHS

$T(n) = 2^{n+1} - 1$

RHS

$2T(n-1) + 1 = 2(2^{(n-1)+1} - 1) + 1$

$= 2(2^n - 1) + 1 = 2(2^n) - 2 + 1$

$= 2^{n+1} - 1$

LHS = RHS

11 cont. $T(n) = 2T(n-1) + 1 \quad T(0) = 1$

- Forwards

(1) 3 expansions

$T(0) = 1$

$T(1) = 2T(0) + 1 = 2 + 1 = 3$

$T(2) = 2T(1) + 1 = 2(3) + 1 = 7$ $\Rightarrow T(n) = 2^{n+1} - 1$

$T(3) = 2T(2) + 1 = 2(7) + 1 = 15$

(2) $\sum_{k=0}^{n} x^k = \frac{x^{k+1} - 1}{x - 1}$ $\quad T(n) = \sum_{k=0}^{n} 2^k = \frac{2^{n+1} - 1}{2 - 1} = 2^{n+1} - 1$

(3) LHS

$T(n) = 2^{n+1} - 1$

RHS

$2T(n-1) + 1 = 2(2^{(n-1)+1} - 1) + 1 = 2(2^n) - 2 + 1 = 2^{n+1} - 1$

LHS = RHS