Sam Nelson CS 325 Homework 2 15 Jan 2016

1.)

a)

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

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

Hypothesis: $T(n) \geq cn^2$ or $T(n)=\Omega(n^2)$

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

$c(n-2)^2+n$

$c(n-2)(n-2)$ -> foil -> $c(n^2-n-n+4)+n$ -> $cn^2 - 2cn + 4c + n$

$\geq cn^2$

If $n(-2c+1)+c \geq 0$ which requires $n \geq 0$ and c is $0 < c \leq 1/2$.

Upper bound $T(n)=O(n^2)$

Hypothesis: $T(n) \leq cn^2$

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

$c(n-2)^2+n$

$c(n-2)(n-2)$ -> foil -> $c(n^2-n-n+4)+n$ -> $cn^2 - 2cn + 4c + n$

$\geq cn^2$

If $n(-2c+1)+c \geq 0$ which requires $n \geq 1$ and c is 1 .

$T(n)=\Omega(n^2)$ and $T(n)=O(n^2)$ thus $T(n)=\Theta(n^2)$

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

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

$=T(n-2)+6$

$=T(n-3)+9$

$=T(n-4)+12$

$=T(n-k) + 3k$

Replace k with n-1

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

$=T(n)=3n-3$

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

c) Case 3 as pointed out by Tim Thomas.

$n^{\log_4 2}=n^{1/2}=\mathrm{sqrt}(n)$

so $f(n) = \Omega(n^{.5+\varepsilon})$ for $\varepsilon < .5$

Regularity check:

$af(n/b) = 2(n/4) = n/2 \leq cn$

satisfied for c of -> $1/2 \leq c < 1$

d) $T(n) = 4T\left(\frac{n}{2}\right) + n^2\sqrt{n}$

Master theorem: $T(n) = aT\left(\frac{n}{b}\right) + f(n)$

$f(n)= n^2\sqrt{n}=n^{5/2}$ and $n^{\log_b a}=n^{\log_2 4}= n^2$

since $n^{5/2}=\Omega(n^{lg2+3/2})$ - $n^{5/2} == n^{lg2+3/2}$ ✓

Case 3:

$af(n/b) = 4(n/2)^2\sqrt{n}/2 = n^{5/2}\sqrt{2} \leq cn^{\frac{5}{2}} for\ 1/\sqrt{2} \leq c < 1$

$T(n)= \theta(n^2\sqrt{n})$

2.)

A.)

T(n) = 5* T(n/2) + O(n)
a=5; b=2; d=1
d<log(a) = 1< $\log_2(5)$
so $O(n^{log_2 5})$ = **$O(n^{2.3219})$**

B.)

T(n) = 2*T(n-1)+O(1)
T(2)= 2*T(1)
T(3)= 2*T(2) = 2*2*T(1)
T(5)= 2*T(4) = 2*2*2*2*T(1)
so **$O(2^n)$**

C.)

T(n) = 9*T(n/3) + $O(n^2)$
a=9; b=3; d=2
d=$log_2 a$ ➡ 2=$log_3 9$
so **$O(n^2 \log n)$** ✓

3.)

T(n) = 4T($\frac{n}{2}$)+n
4T for the 4 recursive calls, n/2 for the work done, and n for the for loop

Master Method: a=4, b=2, f(n)=n
$n^{log_2 4}$= n^2
Case 1 satisified.
$\Theta(n^2)$

4.)

```
ternaryFunction(array[], searchValue, int startOfArray, int endOfArray)
{
        int firstThird = end/3;
        int secThird = 2*end/3;
        if(firstThird == searchValue)
                return true;
        else if(secThird == searchValue)
                return true;
        else if(searchValue < array[firstThird])
                then return first third to function
                return ternaryFunction(array, searchValue, startOfArray, firstThird – 1;
                        //- 1 because we already checked the firstThird value.
        else if(searchValue > array[secThird])
                then return the third third to function to search
                return ternaryFunction(array, searchValue, secThird+1, endOfArray)
                        //+1 because of similar reasons above.
        else
                return ternaryFunction(array, searchValue, firstThird, secThird)
}
```

Binary = $T_2(n/2) + 1$
Ternary = $T_3(n/3) + 2$
Just like binary search, the second case of Masters Theorem is relevant with a = 2, b = 3
And f(n) = Θ(2)
Thus: f(n) = Θ(2) = $\Theta(n^{\log_b a})$
Using the second form, **$T_3(n) = \Theta(\log n)$**

5.)

A.)

Merging two sorted arrays of n_1 and n_2 elements will take $O(n_1+n_2)$ time,
Then we will add another sorted array of size n, and so on.
$(n_1+n_2)+(2n+n_3)+(3n+n_4)+\ldots+((k-1)n+n))$
$=2n+3n+4n+\ldots+kn$
$=$**$\underline{O(k^2n)}$**

B.)

O(kn) work required to move k arrays of size n into k/2 arrays of size 2n. We continue doing O(kn) work O(log k) times until we have an array of size kn.
Runtime=**O(kn log k)**

6.)

A.)

Insertion sort takes $\Theta(k^2)$ time to sort a list size k. Since we have a list size of $\frac{n}{k}$ we have $\frac{k^2 * n}{k}$ or this breaks down into Θ(nk) worst case.

B.)

$\Theta(n \lg\left(\frac{n}{k}\right))$

C.)

$T(nk + n\lg(\frac{n}{k})) = \Theta(n \lg n)$ ➡
Let k= $\Theta(\lg n)$
$\Theta(n\lg n + n\lg(n/\lg n))$
$\Theta(n\lg n + n(\lg(n) - \lg(lgn))$
$\Theta(n\lg n + n\lg n - n\lg(\lg n))$
$\Theta(2n\lg n - n\lg(\lg n))$
$\Theta(n\lg n)$ so k **$= \Theta(\lg n)$**

D.)

For certain values of k, insertion sort runs much faster than merge sort. We could pick k values by setting the insertion sort equation equal to the Merge sort equation, simplify, and plug and chug. After k is larger than 43, merge sort is quicker than insert sort. We determined this in last week Homework.

7.)

compare(array[(a-3),(a-2),(a-1), a] of numbers) -> (minLR, maxLR)

```
if (n == 1)
        return (array[1], array[1])
else if (n == 2)
        if( array[1] < array[2])
                return (array[1], array[2])
        else
                return (array[2], array[1])
else
```

```
        (maxL, minL) = compare(array[1...(n/2)])
        (maxR, minR) = compare(array[(n/2 +1)...n])
    if (maxL < maxR)
        max = maxR
    else
        max = maxL
    if (minL < minR)
        max = minL
    else
        min = minR
```

Recurrence = $T(n) = 2T(n/2) + 3$

$\mathbf{\Theta(n)}$