CSCI 3104
Spring 2018
Problem Set 2

Allison, George
02/17

February 1, 2018

1a)$T(n) = T(n-2) + Cn$ if $n > 1$ and $(T) = C$
$T(n) = T(n-4) + C(n-2) + Cn$
$T(n) = T(n-6) + C(n-4) + C(n-2) + Cn$
...
$\sum_{i=0}^{\frac{n}{2}} n - 2i$ which equals $\frac{n^2}{8} - \frac{n}{4}$
therefore $T(n)$ has $\Theta(n^2)$

1b)Using substitution:
$T(n) = 3T(n-1) + 1$
$T(n) = 3^2T(((n-1)-1)+1)+1$
$T(n) = 3^3T(((n-1)-1-1)+1+1)+1$
...
$3^iT(n-1)+i$. The variable i is solved for by using the base case $T(n-1) = 3$ therefore $i = (n-3)$ which can be substituted in.
$3^{(n-3)}(3) + (n-3) = O(n)$

1c) $T(n) = T(n-1) + 3^n$ if $n > 1$, and $T(1) = 3$
$T(n) = T(n-2) + 3^{n-1} + 3^n$
$T(n) = T(n-3) + 3^{n-2} + 3^{n-1} + 3^n$
...
$T(n) = T(1) + 3^2 + ... + 3^n$ therefore $T(n) = \Theta(3(3^n - 1))$

1d) $T(n) = T(n^{1/4}) + 1$ if $n > 2$ and $T(n) = 0$
$T(n) = T(n^{1/8}) + 2$
$T(n) = T(n^{1/16}) + 3$
...
$T(n) = T(n^{1/2^x}) + x$. To find when $n^{1/2^x}$ is less that 2 calculate $n^{\frac{1}{2^x}} \log(n) < 1$ which yeilds $x > lg(lg(n))$ therefore $\Theta(lg(lg(n)))$

2)In terms of input n, "hello" is printed $n^{\log_3 2}lg(n)$ times. This answer is found by representing the function foo() as the recurrence $T(n) = 2T(n/3)+1$ where $2T(n/3)$ represents the two recursive calls with parameter $n/3$ and the $+1$ represents the atomic operation print("hello"). Via the master method, with $a = 2$, $b = 3$, and $c = f(1)$, $log_b a > c$ therefore $T(n) = \Theta(n^{\log_b a} lg(n))$

3a)

```
# @params
# A is the raludominular input array
# l is the left placeholder
# r is the right placeholder
# l and r are the dynamic bounds of the subsection of the array that the function iterates
    through in a way similar to a binary search. They are necessary to keep an
    asymptotically sub-linear time.

findMin(A, l, r){
  mid = floor(l + r/2)
  if (A[mid] > A[mid+1])
    findMin[A, mid,r]
  else if (A[mid > A[mid-1])
    findMin[A, l, mid]
  else
    return A[mid]
}
```

3b) The algorithm follows the correctness of binary search which has a time complexity of $O \log(n)$

3c) If the array was multi-raludominular the algorithm would fail because it would return only the left-most local minima. This would happen because the midpoint of the search is found by flooring the sum of the left placeholder and half of the right placeholder, effectively moving 50% across the array each iteration. The function findMin is only capable of returning one value, so it would surely fail.

3d) For arrays that have 2 local minima the algorithm will need to search for the max value between the two minima and then divide the array into two parts. Then the minimum of each part can be found then compared to find the minimum value of the entire array.

```
# A is the input array, l and r are the dynamic bounds of the subsection of the array that
    the function iterates through in a way similar to a binary search.

findMax(A, l, r){
  mid = floor(l + r/2)
  if(l-r+1 == 3)
    return mid
  if (A[mid] > A[mid+1])
    return findMax[A, mid,r]
  else if (A[mid > A[mid-1])
    return findMax[A, l, mid + 1]
}
```

4) Sorted list: (*Brackets signify an equivalence class)

$$n!\ ,\ e^n\ ,\ (\tfrac{5}{4})^n\ ,\ lg(n)!\ ,\ n^{1.5}\ ,\ [nlg(n)\ ,\ lg(n!)]\ ,\ n,\ 8^{lg(n)}\ ,\ 4^{lg*n}\ ,\ [n^{1/lg(n)}\ ,\ 42]$$