CSCI 3104 Spring 2018 Problem Set 2
Mohamed Al-Rasbi (07/25)

---

1. *Solve the following recurrence relations using any of the following methods: unrolling, tail recursion, recurrence tree (include tree diagram), or expansion.*

   *Each each case, show your work.*

   (a) $T(n) = T(n-2) + Cn$ *if* $n > 1$, *and* $T(n) = C$ *otherwise*

$$\begin{aligned}
T(n) &= T(n-2) + Cn \\
&= T(n-4) + C(n-2) + Cn \\
&= T(n-6) + C(n-4) + C(n-2) + Cn \\
&. \\
&. \\
&. \\
&= T(0) + 2C + 4C + ... + nC \\
&= C + C\frac{n}{2}(\frac{n}{2} + 1) \quad ^{(1)} \\
&= \frac{C}{4}(n^2 + 2n) + C \\
&= \frac{C}{4}(n^2 + 2n + 4) \\
&= \Theta(n^2)
\end{aligned}$$

$^{(1)}$ I arrived at this conclusion by finding the summation of n integers and excluding even numbers. For example, if $n = 10$, then we will sum $10 + 8 + 6 + 4 + 2 = 30$. By the equation: $\frac{10}{2}(\frac{10}{2} + 1) = 5 * 6 = 30$
If n is odd, then the equation will be a little bit different, but eventually $T(n) = \Theta(n^2)$ in either case.

(b) $T(n) = 3T(n-1) + 1$ *if* $n > 1$ *and* $T(1) = 3$

$$\begin{aligned}
T(n) &= 3T(n-1) + 3^0 \\
&= 3^2T(n-2) + 3^1 + 3^0 \\
&= 3^3T(n-3) + 3^2 + 3^1 + 3^0 \\
&= 3^4T(n-4) + 3^3 + 3^2 + 3^1 + 3^0 \\
&. \\
&. \\
&. \\
&= 3^nT(1) + \sum_{i=0}^{n-1} 3^i = \frac{1-3^n}{1-3} - 1 \\
&= \Theta(3^n)
\end{aligned}$$

(c) $T(n) = T(n-1) + 3^n$ *if* $n > 1$ *and* $T(1) = 3$

$$\begin{aligned}
T(n) &= T(n-1) + 3^n \\
&= T(n-2) + 3^{n-1} + 3^n \\
&= T(n-3) + 3^{n-2} + 3^{n-1} + 3^n \\
&= T(n-4) + 3^{n-3} + 3^{n-2} + 3^{n-1} + 3^n \\
&. \\
&. \\
&. \\
&= T(1) + 3^2 + 3^3 + ... + 3^n \\
&= \sum_{i=1}^{n} 3^i = \frac{1-3^{n+1}}{1-3} - 1 \\
&= \frac{1}{2}(3^{n+1} - 1) - 1 \\
&= \Theta(3^n)
\end{aligned}$$

(d) $T(n) = T(n^{1/4}) + 1$ *if* $n > 2$, *and* $T(n) = 0$ *otherwise*

$$\begin{aligned}
T(n) &= T(n^{1/4}) + 1 \\
&= T(n^{1/16}) + 2 \\
&= T(n^{1/64}) + 3 \\
&. \\
&. \\
&. \\
&= T(n^{1/4^i}) + i
\end{aligned}$$

We seek to find the value of i when:

$$\begin{aligned} n^{1/4^i} &\leq 2 \\ \frac{1}{4^i}\log_2 n &\leq \log_2 2 \\ \log_2 n &\leq 4^i \\ \log_2(\log_2 n) &\leq i\log_2 4 \end{aligned}$$

So, $i \geq \dfrac{1}{2}\log_2(\log_2 n)$. Therefore, $T(n) = T(2) + \dfrac{1}{2}\log_2(\log_2 n) = \Theta(\log_2(\log_2 n))$

2. *Consider the following function:*

```
def foo(n) {
    if (n > 1) {
    print( ''hello'' )
    foo(n/3)
    foo(n/3)
}}
```

*In terms of the input n, determine how many times is "hello" printed. Write down a recurrence and solve using the Master method.*

Let H(n) be the number of times "hello" is printed:

1. $H(n) = 2H(n/3) + 1$

2. $a = 2$ is the number of recursive calls made, $g(n) = n/3$ is the size of sub-problem passed to each of these calls, and $f(n) = 1$ is the cost of the current call.

3. $(n^{\log_b a} = n^{\log_3 2}) \geq (f(n) = 1)$, so $f(n) = n^{\log_3 2 - \epsilon}$, which is case 1.

Therefore, the number of times "hello" is printed is $H(n) = \Theta(n^{\log_3 2})$

3. *Professor McGonagall asks you to help her with some arrays that are* **raludominular**. *A raludominular array has the property that the subarray $A[1..i]$ has the property that $A[j] > A[j+1]$ for $1 \leq j < i$, and the subarray $A[i..n]$ has the property that $A[j] < A[j+1]$ for $i \leq j < n$. Using her wand, McGonagall writes the following raludominular array on the board $A = [7, 6, 4, -1, -2, -9, -5, -3, 10, 13]$, as an example.*

   (a) *Write a recursive algorithm that takes asymptotically sub-linear time to find the minimum element of $A$.*

   I wrote this algorithm using python.

```
0    def findMin(A, n):
1          '''
2         Assuming length of A is at least 2.
3         A is the raludominular array
4         n is the length of the array
5         return minumum element of A
6         '''
7         midIndex = n/2
8         mid = A[midIndex]
9
10        left = A[midIndex - 1]
11        if(len(A) == 2):
12              if(left > mid):
13                  return mid
14              else:
15                  return left
16
17         right = A[midIndex + 1]
18         if(left > mid and right > mid):
19              return mid
20
21         elif(left > mid and mid > right):
22              return findMin(A[midIndex:], len(A[midIndex:])) #right subarray
23
24         elif(left < mid and mid < right):
25              return findMin(A[:midIndex], len(A[:midIndex])) #left subarray
```

   (b) *Prove that your algorithm is correct. (Hint: prove that your algorithm's correctness follows from the correctness of another correct algorithm we already know.)*

   Before proving this algorithm is correct, I will show how it works.
   Let's say that M is the sub-array of A that has the property $A[j] > A[j+1]$ and N is the sub-array that has the property $A[j] < A[j+1]$.
   `(lines 7, 8)` Finding middle element and its index.
   `(line 10)` Find the element that is on the left of the middle element.
   `(lines 11-15)` Base case. If the size of the array(or sub-array after recursion calls) is 2, then

return the smallest element.
`(line 17)` Find the element that is on the right of the middle element. I wrote this line after the if-else statement because if there are only two elements, then middle element will be the second one, so there will be no right element.
`(lines 18, 19)` If left is larger than mid, and right is also larger than mid, then we know that mid is the minimum element in the array, so we return it.
`(lines 21-25)` If mid is in M, then return the sub-array starting from the middle element until the end. If mid is in N, then return the left sub-array.

`PROOF: using loop invariant`
`Initialization:` Find the middle element. It is the minimum element before comparing it to left and right elements. If it is not the minimum, then continue.
`Maintenance:` If mid is in M, return right sub-array starting from mid, if it is in N return left sub-array ending at mid. After n recursions, we either find the minimum element by comparing it to left and right elements, or continue executing by either passing right or left sub-array, until we find the minimum element. Therefore, invariant is preserved.
`Termination:` When there are 3 elements left, we have three options. If mid element(the second one) is the minimum, then it will be returned. If first element is the minimum, then left sub-array will be passed. If third element is the minimum, then right sub-array will be passed. In those two cases, we will reach the base case and the minimum element will be returned. So, the function will be terminated.

`We also know that binary search is correct, so this algorithm is correct.`

(c) *Now consider the **`multi-raludominular`** generalization, in which the array contains k local minima, i.e., it contains k subarrays, each of which is itself a raludominular array. Let $k = 2$ and prove that your algorithm can fail on such an input.*

The code assumes that there are only one minimum value, and it compares the middle value to its left and its right. If we assume there are two sub-arrays and each sub-array is raludominular, then there will be two minimum values.
This code could get one of them, and it could the "larger minimum value" between the two. Therefore, this proves that the algorithm fails when the array is multi-raludominular.

(d) *Suppose that $k = 2$ and we can guarantee that neither local minimum is closer than $n/3$ positions to the middle of the array, and that the "joining point" of the two singly-raludominular subarrays lays in the middle third of the array. Now write an algorithm that returns the minimum element of A in sublinear time. Prove that your algorithm is correct, give a recurrence relation for its running time, and solve for its asymptotic behavior.*

I don't know

4. *Asymptotic relations like $O$, $\Omega$, and $\Theta$ represent relationships between functions, and these relationships are transitive. That is, if some $f(n) = \Omega(g(n))$, and $g(n) = \Omega(h(n))$, then it is also true that $f(n) = \Omega(h(n))$. This means that we can sort functions by their asymptotic growth.*[1]

   *Sort the following functions by order of asymptotic growth such that hte final arrangement of functions $g_1, g_2 \ldots, g_{12}$ satisfies the ordering constraint $g_1 = \Omega(g2)$, $g_2 = \Omega(g_3)$, ..., $g_{11} = \Omega(g_{12})$.*

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $n$ | $n^{1.5}$ | $8^{\lg n}$ | $4^{\lg * n}$ | $n!$ | $(\lg n)!$ | $(\frac{5}{4})^n$ | $n^{1/\lg n}$ | $n \lg n$ | $\lg(n!)$ | $e^n$ | 42 |

   *Give the final sorted list and identify which pair(s) functions $f(n)$, $g(n)$, if any, are in the same equivalence class, i.e., $f(n) = \Theta(g(n))$.*

   I used these identities to find the answer:
   $a^{\log_b c} = c^{\log_b a}$
   $(a^b)^c = a^{a.c}$

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| $n!$ | $e^n$ | $(\frac{5}{4})^n$ | $(\lg n)!$ | $n^{1.5}$ | $n \lg n$  $\lg(n!)$ | $n$ | $8^{\lg n}$ | $4^{\lg * n}$ | $n^{1/\lg n}$  42 |

   These have the same equivalence classes:
   $[n \lg n$, $\lg(n!)]$ and $[n^{1/\lg n}$, $42]$

I worked with:
Omar Mohammed, Mahmoud Almansouri, Firas Al Mahrouky, Yazeed Almuqwishi, Absdullah Bajkhaif