CSCI 3104-300 Summer 2019 Problem Set 3
Hoffman, Ryan
01/11

1. (10 points) For parts (1a) and (1b), justify your answers in terms of deterministic QuickSort, and for part (1c), refer to Randomized QuickSort. In both cases, refer to the versions of the algorithms given in lecture (you can refer to the moodle lecture notes).

   (a) What is the asymptotic running time of QuickSort when every element of the input A is identical, i.e., for $1 \leq i, j \leq n$, $A[i] = A[j]$?

   Solution:

   If all of the elements are the same, then array A is just an array with one element and is therefore already sorted trivially. This falls under a worst case scenario where, from our notes, $T(n) = \Theta(n^2)$.

   (b) Let the input array $A = [9, 7, 5, 11, 12, 2, 14, 3, 10, 6]$. What is the number of times a comparison is made to the element with value 3?

   Solution:

   According to my interpretation of the pseudocode and a deterministic QuickSort python program that I ran, the number of times that a comparison is made to the value 3 is three(3).

   (c) How many calls are made to `random-int` in (i) the worst case and (ii) the best case? Give your answers in asymptotic notation.

   Solution:

   If we let $N(n)$ be the number of times the `random-int` function is called on the array, A, of size n, then we only have two cases to consider:

   i. When the partition creates two equal size subarrays/subproblems we define $N(n) = 2N(\frac{n}{2}) + 1$. This yields $N(n) = \Theta(n)$.
   ii. When the partition creates a subproblem of size $n - 1$, we define $N(n) = N(n - 1) + 1$. However, this also yields $N(n) = \Theta(n)$.

   Therefore, I conclude based on class notes and the above definition of $N(n)$, that both cases require $\Theta(n)$ recursive calls to `random-int` and there is no best or worst case.

2. (20 points) Solve the following recurrence relations using any of the following methods: unrolling, substitution, or recurrence tree (include tree diagram). For each case, show your work.

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

   Solution:

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

   $= T(n-4) + C + C$

   $= T(n-6) + C + C + C$

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

   Now, $T(0)$,$T(1) = C$

   If even: $n - 2k = 0 \Rightarrow k = \frac{n}{2}$

   Then $T(n) = T(n - 2\frac{n}{2}) + \frac{n}{2}C$

   $= T(0) + \frac{n}{2}C$

   If odd: $n - 2k = 1 \Rightarrow k = \frac{n-1}{2}$

   $T(n) = T(n - 2\frac{n-1}{2}) + \frac{n-1}{2}C$

   $= T(1) + \frac{n-1}{2}C$

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

   Solution:

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

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

   $= 9T(n-2) + 3 + 1$

   $= 9[3T(n-3) + 1] + 3 + 1$

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

   $= 3^k T(n-k) + 3^{k-1} + 3^{k-2} + 3^{k-3} \cdots + 1$

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

(c) $T(n) = T(n-1) + 2^n$ if $n > 1$, and $T(1) = 2$

Solution:

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

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

$= t(n-3) + 2^{n-2} + 2^{n-1} + 2^n$

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

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

I was not able to go any further at this point.

(d) $T(n) = T(\sqrt{n}) + 1$ if $n \geq 2$, and $T(n) = 0$ otherwise

Solution:

$T(n) = T(\sqrt{n}) + 1$

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

$= T(n^{\frac{1}{4}}) + 1 + 1$

$= T(n^{\frac{1}{8}}) + 1 + 1 + 1$

I was not able to go any further at this point.

3. (20 points) Use the Master Theorem to solve the following recurrence relations. For each recurrence, either give the asympotic solution using the Master Theorem (state which case), or else state the Master Theorem doesn't apply.

   (a) $T(n) = T(\frac{3n}{4}) + 2$

   Solution:

   a=1, b=$\frac{4}{3}$, and d=2 $\Rightarrow T(n) = \Theta(1)$

   (b) $T(n) = 3T(\frac{n}{4}) + nlgn$

   Solution:

   a=3, b=4, and d=$nlogn \Rightarrow T(n) = \Theta(nlogn)$

   (c) $T(n) = 8T(\frac{n}{3}) + 2^n$

   Solution:

   a=8, b=3, and d=2 $\Rightarrow T(n) = \Theta(2^n)$, by case (3).

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

   Solution:

   a=2, b=2, and d=n^2 $\Rightarrow$Master Theorem does not apply.

   (e) $T(n) = 100T(\frac{n}{42}) + \lg n$

   Solution:

   a=100, b=42, and d=$n^2 \Rightarrow T(n) = \Theta(lgn)$

4. (30 points) Professor Trelawney has acquired n enchanted crystal balls, of dubious origin and dubious reliability. Trelawney needs your help to identify which crystal balls are accurate and which are inaccurate. She has constructed a strange contraption that fits over two crystal balls at a time to perform a test. When the contraption is activated, each crystal ball glows one of two colors depending on whether the other crystal ball is accurate or not. An accurate crystal ball always glows correctly according to whether the other crystal ball is accurate or not, but the glow of an inaccurate crystal ball glows the opposite color of what the other crystal ball is (i.e. If the other crystal ball is accurate, it will glow red. If the other crystal ball is inaccurate it will glow green). You quickly notice that there are two possible test outcomes:

crystal ball i glows	crystal ball j glows		
red	red	$\Longrightarrow$	at least one is inaccurate
green	green	$\Longrightarrow$	both are accurate, or both inaccurate

(a) Prove that if $n/2$ or more crystal balls are inaccurate, Professor Trelawney cannot necessarily determine which crystal balls are tainted using any strategy based on this kind of pairwise test.

Solution:

Using CLRS Chip Testing Method as a guide, we see that, due to symmetric behaviors exhibited on the sets of accurate and inaccurate crystal balls, they basically cancel each other out. Therefore it is not possible to determine which crystal balls are tainted or which are accurate.

(b) ~~Consider the problem of finding a single good crystal ball from among the n crystal balls, and suppose Professor Trelawney knows that more than $n/2$ of the crystal balls are accurate, but not which ones. Prove that $\lfloor n/2 \rfloor$ pairwise tests are sufficient to reduce the problem to one of nearly half the size.~~

Solution:

This question has been eliminated.

(c) Now, under the same assumptions as part (4b), prove that all of the accurate crystal balls can be identified with $\Theta(n)$ pairwise tests. Give and solve the recurrence that describes the number of tests.

Solution:

Using the last sentence from part (b), we find a single accurate crystal ball using the recurrence relation:

$T(n) \leq T(\lceil n/2 \rceil) + \lfloor n/2 \rfloor$

and, by the Master theorem, $\Rightarrow T(n) = \mathcal{O}(n)$. Then we add the remaining $n-1$ crystal balls to get $T(n) = \mathcal{O}(n) + n - 1 = \Theta(n)$ tests.

5. (10 points) Harry needs your help breaking into a dwarven lock box. The lock box projects an array A consisting of n integers $A[1], A[2], \ldots, A[n]$ and has you enter in a two-dimensional $n \times n$ array B – to open the box – in which $B[i, j]$ (for $i < j$) contains the sum of array elements $A[i]$ through $A[j]$, i.e., $B[i, j] = A[i] + A[i + 1] + \cdots + A[j]$. (The value of array element $B[i, j]$ is left unspecified whenever $i \geq j$, so it doesn't matter what the output is for these values.)
Harry suggests the following simple algorithm to solve this problem:

```
dwarvenLockBox(A) {
   for i = 1 to n {
      for j = i+1 to n {
         s = sum(A[i..j])       // look very closely here
         B[i,j] = s
}}}
```

(a) For some function g that you should choose, give a bound of the form $\Omega(g(n))$ on the running time of this algorithm on an input of size n (i.e., a bound on the number of operations performed by the algorithm).

Solution:

The outer for loop runs n times and the inner loop will run a maximum of n times for each iteration of the outer loop, which then sums the elements in A and stores them in B giving us a total of $\Omega(n^3)$

(b) For this same function g, show that the running time of the algorithm on an input of size n is also $O(g(n))$. (This shows an asymptotically tight bound of $\Theta(g(n))$ on the running time.)

Solution:

The same function g must have the same upper bound as the lower bound$\Omega(n^3)$ which yields an asymptotically tight bound of $\Theta(g(n))$ on the running time.

6. (20 points) Consider the following strategy for choosing a pivot element for the `Partition` subroutine of QuickSort, applied to an array A.

   - Let n be the number of elements of the array A.
   - If $n \leq 24$, perform an Insertion Sort of A and return.
   - Otherwise:
     - Choose $2\lfloor n^{(1/2)} \rfloor$ elements at random from n; let S be the new list with the chosen elements.
     - Sort the list S using Insertion Sort and use the median m of S as a pivot element.
     - Partition using m as a pivot.
     - Carry out QuickSort recursively on the two parts.

   (a) How much time does it take to sort S and find its median? Give a Θ bound.
   Solution:
   I was unable to formulate a solution for the last question.

   (b) If the element m obtained as the median of S is used as the pivot, what can we say about the sizes of the two partitions of the array A?
   Solution:
   I was unable to formulate a solution for the last question.

   (c) Write a recurrence relation for the worst case running time of QuickSort with this pivoting strategy.
   Solution:
   I was unable to formulate a solution for the last question.