CSCI 3104 **Oscar Delgado**
Problem Set 3 (110 points) **Summer 2019, CU-Boulder**

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 $1 \leq i, j \leq n$, $A[i] = A[j]$?*

   $\Theta(n^2)$ is the asymptotic running time. As it still makes the comparisons whens its equal.

   (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?*

   5. First when comparison with whole array,element 6. Then with partition of that with subarray [5,2,3,6] then with subarray [5,2,3] then with subarray [5,3] then with subarray [3]. With counting the last one when comparing to itself its 5.

   (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.*

   Worst case for Random-int is $O(n^2)$
   Best case for Random-int $\Omega(n \ln(n))$

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*
   $T(n) = T(n-4) + C + C\ /\ /\ T(n-2) = T(n-4) + C$
   $T(n) = T(n-6 + C + C + C)\ /\ /\ T(n-4) = T(n-6) + C$
   $T(n) = \sum_{i=1}^{n/2} C = \frac{c3n}{8} = \Theta(n)$

   (b) $T(n) = 3T(n-1) + 1$ *if* $n > 1$, *and* $T(1) = 3$
   $T(n) = 3^2 T(n-2) + 3 + 1\ /\ /\ T(n-1) = 3T(n-2) + 1$
   $T(n) = 3^3 T(n-3) + 3^2 + 3 + 1\ /\ /\ T(n-2) = 3^2 T(n-3) + 1$
   $T(n) = \sum_{i=0}^{n-1} 3^i => \frac{C3^n - C}{2} = \Theta(n^3)$

   (c) $T(n) = T(n-1) + 2^n$ *if* $n > 1$, *and* $T(1) = 2$
   $T(n) = T(n-2) + 2^n + 2^{n-1}\ /\ /\ T(n-1) = T(n-2) + 2^{n-1}$ $T(n) = T(n-3) + 2^n + 2^{n-1} + 2^{n-2}\ /\ /\ T(n-2) = T(n-3) + 2^{n-2}$ $T(n) = \sum_{i=0}^{n} 2^i => C2^{n+1} - C = \Theta(2^n)$

   (d) $T(n) = T(\sqrt{n}) + 1$ *if* $n \geq 2$, *and* $T(n) = 0$ *otherwise*
   $T(n) = T(n^{1/2^2} + 1 + 1$ (sum=2) $/\ /\ T(n^{1/2}) = T(n^{1/4}) + 1$
   $T(n) = T(n^{1/2^3}) + 1 + 1 + 1$ (sum=3) $/\ /\ T(n^{1/4}) = T(n^{1/8}) + 1$
   $T(n) = T(n^{1/2^k}) + k$ so $\sum_{i=0}^{k} 1$ so it will be $\Theta(k)$ and now we solve for k.
   lets assume $n^{1/2^k} = 10$
   $log(n^{1/n^k}) = log(10)$
   $\frac{log(n)}{2^k} = log(10)$
   $log(n) = 2^k log(10)$
   $log(n) = 2^k$
   $ln(log(n) = ln(2^k)$
   $ln(log(n)) = k$
   $O(ln(log(n)))$

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

   This would be case 1. so by that $\Theta(n^{\log_b a})$ and a = 1 and b = $\frac{4}{3}$ so $\Theta(n^{\log_{\frac{4}{3}}(1)})$

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

   This would be case 3. so by that $\Theta(f(n))$ so its $\Theta(n \lg(n))$.

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

   This would be case 3. so by that $\Theta(f(n))$ so its $\Theta(2^n)$.

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

   Master Theorem doesn't apply. This doesn't follow the master theorem form.

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

   This would be case 1. a = 100 and b = 42 so $\Theta(n^{\log_b a})$ therefore $\Theta(n^{\log_{42}(100)} = \Theta(n^{2.38})$

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.*

If more than half the crystals are inaccurate then more than half the comparisons are going to be inaccurate. If more than half the comparisons are inaccurate then the you can't say for certain if a crystal in accurate or not. Let's also assume that Trelawney doesn't know that half or more crystal are inaaccurate, if we take the best stragedy where we take 1 crystal and compare it to the rest we still wouldn't know if the crystal is good because we don't have the knowledge if n/2 crystals are bad so when comparing it that one we determine might be "good" we still wouldn't know if the others are good.

(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.*

(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.*

With knowledge of knowing more than half are good then that takes n time to find one good crystal then with this we can split the rest into pairs and compare them. the ones the glow green we only compare one with the one we know is good and if it lights green then they are both good if it lights reds. Then for the ones that glow red match with another until it glows green then compare with a good one you know until sorted and if the last pair is red compare one if green thats good and if red the other one is good. with this I concluded the recurrence relation should be $T(n) = 2T(\frac{n}{2}) + n$. a=2 as there are 2 cases one if it lights R R and two if light G G.

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).*

   There is a for loop inside a another for loop so n^2 but on line 4 (s = sum(A[i..j])) is another loop inside those for loops so then $\Omega(n^3)$ but because the loops don't go from 0-n every time i will give $g(n) = n^3/C$

   (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.)*

   To simplifly the question we can say g(n) $= \frac{n^3}{C} < n^3$ then we can say by definition of Upper bound $n^3 < C_1 * n^3$ when $C_1 > 1, n_0 = 1$ so we could also say $O(n^3)$ and thus saying the $\Theta(n^3)$. You can also trivially see this when you notice a for loop inside a for loop inside another loop.

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.*

   Size of S = $2\sqrt{n}$ and we know from lecture that insertion sort sorts n^2 so S would be 4n + (time to find median) and this would be half the size so 4n+$\sqrt{n}$ therefor the bound would be $\Theta(n)$

   (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?*

   The median of S should be roughly the median of A because we are taking random elements of A we can assume we get enough to get a wide spread of A elements therefor the median of S should be similar to A and if this is true than the 2 partitions would be around the same size which is n/2.

   (c) *Write a recurrence relation for the worst case running time of QuickSort with this pivoting strategy.*

   $T(n) = T(n - \sqrt{(n)}) + n$ because the worst case is when the S is the first or last subarray of A. (A[0...2$\sqrt{n}$) and the median of that would be the element half of that so the first partition is A[0...$\sqrt{n}-1$] elements and second partition A[$\sqrt{n}$....n] elements showing the recurrence relation.