Karim Maamari

***Exercises***

**1.**

***(a)*** Let's assume that n is a power of 5, now let’s assume n = 5 k for some non-negative integer k. We can then write the recurrence relation as:

T (5 k) = 5 \* T (5 k-1) + 5 k

Substituting k-1 for k in the second term of the equation we get:

T (5 k) = 5 \* T (5 k-1) + 5 k = 5 \* (5 \* T (5 k-2) + 5 k-1) + 5 k = 5 2 \* T (5 k-2) + 2 \* 5 k

Continuing this process, we can see that:

T (5 k) = 5 i \* T (5 k-i) + i \* 5 k

where i ranges from 0 to k.

When i = k, we have T (5 k) = 5 k \* T (1) + k \* 5 k = 5 k \* (T (1) + k) = 5 k \* (5 + k).

Therefore, the exact solution to T(n) = 5 \* T(n/5) + n with T (1) = 5, when n is a power of 5, is T(n) = n \* (5 + log5(n)).

***(b)***

Following the same process, we did in part (a), we can write the recurrence relation as: T(5k) = 5 \* T (5 k-1) + 6 \* 5 k

Then we can substitute k-1 for k in the second term so we get:

T (5 k) = 5 \* T (5 k-1) + 6 \* 5 k = 5 \* (5 \* T (5 k-2)) + 6 \* 5 k-1 + 6 \* 5 k = 5 2 \* T (5 k-2) + 2 \* 6 \* 5 k

T (5 k) = 5 i \* T (5 k-i) + i \* 6 \* 5 k

When i = k, we have T (5 k) = 5 k \* T (1) + k \* 6 \* 5 k = 5 k \* (T (1) + 6k) = 5 k \* (5 + 6log5(n)).

Therefore, the exact solution to T(n) = 5 \* T(n/5) + 6n with T (1) = 5, when n is a power of 5, is T(n) = n \* (5 + 6log5 (n)).

**2.**

***(a)*** Using the Master Theorem, we have a = 7, b = 9, and f(n) = sqrt(n). Since logb(a) = log9 (7) < 1/2 and f(n) = sqrt(n) is greater than n^0.5, we can apply case 3 of the Master Theorem. Therefore, the solution to the recurrence relation is T(n) = O (n^ (log9 (7))).

***(b)*** Using substitution method, we can assume here that T(n) = O (n 2) and prove it by induction. For the base case, T (6) = 21 which is a constant, so our assumption holds. Assuming T(k) <= ck 2 for all k < n, we have:

T(n) = T(n-6) + n <= c(n-6) 2 + n

= cn 2 - 12cn + 36c + n

<= cn 2

where the last inequality holds for sufficiently large c. Therefore, T(n) = O (n 2).

***(c)*** Using the Master Theorem, we have a = 8, b = 6, and f(n) = n 2. Since logb(a) = log6(8) > 1 and f(n) = n 2 is greater than n 1, we can apply case 1 of the Master Theorem. Therefore, the solution to the recurrence relation is T(n) = O (n^ (log6 (8))).

***(d)*** Using the Master Theorem, we have a = 7, b = 4, and f(n) = n 3. Since logb(a) = log4 (7) > 3/2 and f(n) = n 3 is greater than n (3/2) we can apply case 1 of the Master Theorem. Therefore, the solution to the recurrence relation is T(n) = O (n^ (log4 (7))).

**3.**

The algorithm printStuff takes an array A as input and prints its elements. It first checks if the size of A is less than or equal to 15, in which case it returns without doing anything. Otherwise, it loops through the entire array which takes O(n) time, where n is the size of A. Then, it makes two recursive calls on the first and last 20% of the array. The recursion stops when the size of the array is less than or equal to 15.

The running time of printStuff is O(nlog(n)) because the algorithm traverses the array of size n and makes two recursive calls on subarrays of size at most 4n/5. The recurrence relation can be written as T(n) = 2T(4n/5) + O(n), and using the Master Theorem, we can solve it to get T(n) = O(nlog(n)).

**4.**

***(a)*** The standard algorithm for multiplying two n by n matrices involves computing the dot product of each row in the first matrix with each column in the second matrix. This requires n^2 operations for each entry in the resulting n by n matrix, giving a total of n^3 operations. Therefore, the running time of the standard algorithm is O(n^3).

***(b)*** The divide and conquer strategy involve dividing each matrix into four n/2 by n/2 sub-matrices, and recursively multiplying these sub-matrices to obtain the final result. The recurrence relation for this algorithm is T(n) = 8T(n/2) + O(n^2), since each recursive call involves multiplying two n/2 by n/2 matrices, and there are 8 such recursive calls. Using the Master Theorem, we can see that the running time of this algorithm is O(n^3), the same as the standard algorithm.

***(c)*** We can solve the original problem XY by combining them in the following way:

AE + BG = P5 + P4 - P2 + P6

AF + BH = P1 + P2

CE + DG = P3 + P4

CF + DH = P1 + P5 - P3 - P7

The recurrence relation for this approach is T(n) = 7T(n/2) + O(n2), which can be solved using the Master Theorem to get T(n) = O(nlog2(7)). This is a faster runtime than the naive algorithm and the previous divide-and-conquer approach.

***(d)*** The error in the reasoning is that increasing the number of subproblems by a constant factor at each subsequent level does not necessarily mean that the running time per level can only increase by a constant factor. In fact, the size of the subproblems decreases by a constant factor at each level, which means that the running time per level may increase by a larger factor than the number of subproblems. Therefore, the running time per level is not necessarily O(n^2) for all levels, and the total running time may not be O(n^2log(n)).

**5.Optional**

***(a)*** A straightforward algorithm to find all trustworthy tarsiers would be to pair each tarsier with every other tarsier and have them evaluate each other. For each tarsier, count how many times they were evaluated as trustworthy. If a tarsier was evaluated as trustworthy by more than n/2 other tarsiers, then they are a trustworthy tarsier.

***(b)*** To reduce the problem to the same problem with less than half the size, we can pair up tarsiers and have them evaluate each other. For each pair, if they both say the other is trustworthy, then we keep one of them. If one says the other is tricky, then we discard both of them. If they both say the other is tricky, then we discard both of them. We repeat this process until we have less than half the original population, and there are still strictly more than m/2 trustworthy tarsiers in the population.

To see why this works, suppose there are k trustworthy tarsiers in the original population, where k > n/2. If we pair up the tarsiers randomly, then the probability that a trustworthy tarsier is paired with another trustworthy tarsier is (k/n) \* ((k-1)/(n-1)) > 1/4. Therefore, on average, we will keep more than 1/4 of the trustworthy tarsiers in each round. After log2(n) rounds, we will have less than half the original population, and there are still strictly more than m/2 trustworthy tarsiers in the population.

***(c)*** For odd n, we can use the same procedure as in part (b), but we need to handle the case where we have one tarsier left at the end. In this case, we can simply return that tarsier as a trustworthy tarsier if it was evaluated as trustworthy by more than (n-1)/2 other tarsiers.

***(d)*** To find a single trustworthy tarsier using O(n) tarsier-to-tarsier comparisons, we can use the following recursive algorithm:

1.If there is only one tarsier, return that tarsier as the trustworthy one.

2.Divide the n tarsiers into two equal-sized groups (or one group of size n/2 and one group of size n/2 + 1 if n is odd).

3.Recursively find the trustworthy tarsier in each group using the same algorithm.

4.If both groups return the same tarsier as the trustworthy one, return that tarsier.

5.Otherwise, compare the trustworthiness of the two candidates by checking the number of tarsiers that trust each candidate. Return the candidate with more trust.

This algorithm uses O(n) tarsier-to-tarsier comparisons because each level of the recursion splits the tarsiers into two equal-sized groups and each group is compared to find the trustworthy tarsier.

***(e)*** We prove the correctness of the algorithm by induction on n.

Base case: If n = 1, then there is only one tarsier, which must be the trustworthy one. The algorithm correctly returns this tarsier.

Inductive step: Assume that the algorithm correctly finds the trustworthy tarsier for any input of size k < n. We need to show that it also correctly finds the trustworthy tarsier for input of size n.

The algorithm splits the n tarsiers into two groups and recursively finds the trustworthy tarsier in each group. By the inductive hypothesis, each recursive call correctly finds the trustworthy tarsier for the corresponding group.

If both groups return the same tarsier as the trustworthy one, then this tarsier is the trustworthy one for the entire set of n tarsiers. The algorithm correctly returns this tarsier.

If the two groups return different tarsiers as the trustworthy one, then we need to compare their trustworthiness. Let T1 and T2 be the trustworthy tarsiers for the two groups, respectively. Let count (Ti) be the number of tarsiers that trust Ti.

Since the two groups are equal-sized, the total number of tarsiers that trust T1 is count(T1) + count(T2)/2. Similarly, the total number of tarsiers that trust T2 is count(T2) + count(T1)/2. Therefore, we can compare count(T1) + count(T2)/2 to count(T2) + count(T1)/2 to determine which candidate has more trust. The algorithm correctly returns the candidate with more trust. Therefore, the algorithm correctly finds the trustworthy tarsier for any input of size n.

***(f)*** We will prove that the running time of our procedure in part (d) uses O(n) tarsier-to-tarsier comparisons using mathematical induction.

Base Case: When n = 2, our procedure uses only one tarsier-to-tarsier comparison to determine the trustworthy tarsier. Therefore, the running time is O (1), which is certainly O(n).

Inductive Hypothesis: Assume that for some positive integer k, our procedure uses at most k tarsier-to-tarsier comparisons to determine the trustworthy tarsier when there are 2k tarsiers.

Inductive Step: We need to show that our procedure uses at most k+1 tarsier-to-tarsier comparisons when there are 2k+2 tarsiers. By using the procedure in part (b), we can reduce the problem to a population of 2k tarsiers with strictly more than k trustworthy tarsiers. By our inductive hypothesis, our procedure uses at most k tarsier-to-tarsier comparisons to determine the trustworthy tarsier in this population.

Next, we can use the procedure in part (c) to reduce the problem to a population of m tarsiers, where 0 < m ≤ k+1 and there are strictly more than m/2 trustworthy tarsiers. By the constraint of part (c), we have used no more than [(2k+2)/2] = k+1 tarsier-to-tarsier comparisons to get to this point.

Now, we can use our procedure recursively on the population of m tarsiers to determine the trustworthy tarsiers. By our inductive hypothesis, this will take at most m tarsier-to-tarsier comparisons. Therefore, the total number of tarsier-to-tarsier comparisons used by our procedure is at most k + 1 + m ≤ k + 1 + (k+1) = 2(k+1), which is O(n).

Conclusion: By mathematical induction, our procedure uses O(n) tarsier-to-tarsier comparisons to determine the trustworthy tarsier when there are n tarsiers.

***(g)*** To find all trustworthy tarsiers using O(n) tarsier-to-tarsier comparisons, we can modify our procedure in part (d) as follows:

Use the procedure in part (b) to reduce the problem to a population of n/2 tarsiers with strictly more than n/4 trustworthy tarsiers.

Use the procedure in part (c) to reduce the problem to a population of m tarsiers, where 0 < m ≤ n/2 and there are strictly more than m/2 trustworthy tarsiers.

Recursively apply step 1 and step 2 to each population of tarsiers that contains more than one tarsier until we reach populations of one tarsier.

The tarsiers that are determined to be trustworthy in each population are trustworthy tarsiers in the original population.

Since we are using the procedures from parts (b) and (c) which use O(n) tarsier-to-tarsier comparisons, and we are only applying them to populations of size n/2 or less, the total number of tarsier-to-tarsier comparisons used by our procedure is O(n). Therefore, we can find all trustworthy tarsiers using O(n) tarsier-to-tarsier comparisons.