CS 3510 — Algorithms, Spring 2022

Practice Exam 1 Solutions

Problem 1 Indicate whether the following staetments are true or false.

a.) $4^{\log_2(n)} = \Theta(n^2)$ where $4^{\log_2(n)}$ can be written as $2^{2\log_2(n)} = n^2$: $\boxed{true}$.

b.) $n^2 = \Omega(n^{\log_3 11})$ but $\log_3 11$ is grater than 2 because $3^2 = 9 < 11$, but n^2 is not lower bounded by a polynomial with a higher degree: $\boxed{false}$.

c.) $n^n = \Omega(n!)$ where n^n is $\underbrace{n \cdots n}_{n \text{ times}}$ while $n!$ is $n(n-1)\cdots(2)(1)$ clearing making n^n grow larger. Hence, we can lower bound n^n by $n!$: $\boxed{true}$.

d.) $n! = \Omega(2^n)$ where n is the product of numbers that have an average greater than 2, while 2^n is the product of 2 a total of n times. This makes $n!$ grow faster than 2^n and therefore this statement is $\boxed{true}$.

e.) $2^{\log(n)^2} = \Theta(n)$. We can rewrite n as $2^{\log_2(n)}$ and compare exponents noticing that $\log_2(n)$ is not the same order as $\log_2^2(n)$ making this statement $\boxed{false}$.

f.) $\log(n^3) = O(\log(n))$ where we can rewrite $\log(n^3)$ as $3\log(n)$ which is $O(\log(n))$ making this statement $\boxed{true}$.

Problem 2 Solve the following number theory problems:

a.) Calculate $13^{13} \mod 10$

Solution: By using repeated squaring, we can solve this by noticing $(13)_{10} = (1101)_2$ meaning we can compute $13, 13^2, 13^4, 13^8$ to get our final answer.

$$\begin{aligned} 13 &\equiv 3 \mod 10 \\ 13^2 &\equiv 9 \mod 10 \\ 13^4 \equiv 81 &\equiv 1 \mod 10 \\ 13^8 &\equiv 1 \mod 10 \end{aligned}$$

and since $13^{13} = 13^8 13^4 13 \equiv 1 \times 1 \times 3 \equiv \boxed{3 \mod 10}$

b.) Calcuate the multiplicative inverse of $3 \mod 121$

Solution: First, we need to confirm that 3 and 121 are relatively prime, which we can do by computing $gcd(121, 3)$ which is given by:

$$121 = 3(40) + 1$$

We can already rewrite this equation in terms of $1 = 121x + 3y$ by letting $x = 1$ and $y = -40$ such that we have $121 - 40(3) = 1$. This lets us claim -40 is the multiplicative inverse, but we can take the mod of this by 121 to get 81 ($-40 + 121 = 81$). The multiplicative inverse of $3 \mod 121$ is $\boxed{81}$.

c.) True or False. If a has a multiplicative inverse mod b, then b has a multiplicative inverse mod a. Remark: let $a, b \geq 2$

Solution: The existence of a multiplicative inverse a mod b is given by if $gcd(a, b) = 1$, but by the symmetric relation of gcd, we know that if $gcd(a, b) = 1$, then $gcd(b, a) = 1$. Further, the existence of some equation $ax + by = 1$ in one direction (the assumption) means we can take mod b of the equation to get another multiplicative inverse.

Note: this does not work for the case of when a or $b = 1$ as there is no multiplicative inverse for anything mod 1 as $\forall n \in \mathbb{Z} : n \mod (1) = 0$.

d.) Let p be a prime number, and let $c \equiv c' \mod (p-1)$. Show that, for any $a < p$ $a^c \equiv a^{c'}$

Solution: Since c' is equivelant to c mod $p-1$ then we can write $c' = (p-1)k + c$. Hence, we can check if $a^c \equiv a^{(p-1)k+c} \mod (p)$

By splitting up $a^{(p-1)k+c}$ into $a^{(p-1)k}a^c$ and then rearrange $\left(a^k\right)^{p-1}$, now we see:

$$\begin{aligned} a^c &\equiv a^{c'} \mod p \\ &\equiv \left(a^k\right)^{(p-1)} a^c \mod p \\ &\equiv 1a^c \mod p \end{aligned}$$

Hence, it has been proved. □

Problem 3 Given an input $a_1, \ldots, a_n$, we are interested in finding the sum of the elements of the subarray whose elements sum to the maximum value out of all the possible subarrays.

I. A naive brute force approach would simply loop through all the possible subarrays, calculating the sum each time. What is the runtime of this?

Solution: The number of possible subarrays is given by n^2 and each subarray requires $O(n)$ time to compute the sum of the subarray, giving a total runtime of $O(n) \times n^2 = O(n^3)$.

II. Suppose are given a list of integers $a_1, \ldots, a_n$ which are guaranteed to be such that maximum sum subarray spans the middle of the list. Explain how to solve the problem in $O(n)$ time:

Description Given a list of integers, we can pick out the middle element of the array by taking the size of the array and dividing it by 2.

After picking out the middle element, we know that the maximum sum subarray that crosses the middle is equal to the maximum sum subarray of the left (from the middle) and of the right (from the middle).

To find the maximum sum subarray starting at the middle and extending to the left, we build a table and compute subarray sums from i to $n/2$ where $i < n/2$. By saving these values in a table, we can pick the value i such that the value in the table (the sum of the subarray) is maximal.

We reproduce the above without loss of generality for the right side to get the sum of the maximum sum subarray from the middle to the left and add it to the maximum sum subarray from the middle (+1) to the right.

This gives us the sum of the maximum sum subarray that crosses the middle.

Justification: Under the assumption that the maximum sum subarray crosses the middle, we know that the combination of the left and right halves (that include the middle) is a correct answer.

For the left portion of this crossed maximum sum subarray, we fix the right-most element and check every possible value of i such that $i < n/2$. By enumerating over each possible value i, we are guaranteed to have the correct left portion of the answer.

The same logic as above applies for the right portion of the answer giving us the final answer.

Runtime: The runtime of this algorithm is $O(n)$ as we are simply iterating from $n/2 \to 0$ and $n/2 \to n$ and saving these values in a table and picking the largest i such that it maximizes $T[i]$ (where T is our table).

Hence, this algorithm runs in $O(n)$ time.

III. Give a divide and conquer algorithm for Maximum Sum Subarray with runtime $O(n \log n)$

Description: We know that the maximum sum subarray exists either in the left half (exclusively), the right half (exclusively), or crosses the middle.

Hence, we can create an algorithm as such. Let us take an input $a_1, \ldots, a_n$ of which we want to find our maximum sum subarray (MSA). If there is only one element, we return that element (if it is positive). If there are no elements, we return 0.

We compute three candidate solutions:

(a) MSA that spans the middle

(b) MSA that is exclusively on left

(c) MSA that is exclusively on right

Using the algorithm from above, we can get the MSA that spans the middle which is 1 or 3 total candidate solutions. We call this algorithm recursively on the left and the right halves which will return the other 2 candidate solutions.

We simply return the max of these 3 candidate solutions.

Justification: From above, we know that the MSA can only exist in 3 total areas which are either in the middle or on the left and the right. From the previous part, we know that the $O(n)$ algorithm will return the MSA that spans the middle. Hence, we only need to justify correctness for the left and right halves.

We proceed with strong induction. Let the base case be when the size of the array has size 1, then the algorithm works correctly as we can simply return this element if it is element and otherwise return 0 which makes us return the maximal value.

Now, we assume the inductive hypothesis that this algorithm returns the MSA for an array input of size $1 \to k - 1$. For an input size of k, we know that we can use the previous $O(n)$ algorithm to find the candidate solution that spans the middle.

For the other candidate solutions, we look in the left and right halves of the algorithm which return the correct result by the inductive hypothesis.

Using these three candidate solutions, we simply return the maximal solution. Hence, this algorithm is correct. $\square$

Remark: This level of rigor is not needed for exams or homework. A less rigorous justification is valid if it is correct.

Runtime: This algorithm makes two recursive calls where each recursive call has $n/2$ size and performs $O(n)$ non-recursive work, therefore the recurrence relation is:

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n)$$

and by Master Theorem, this gives a runtime of $O(n \log n)$.

Problem 4 Answer the following problems which analyze functions. Let n be a power of 4, suppose we have the provided function (see original PDF for function):

I. Let $H(n)$ denote the number of times `Hello` is printed. What is the recurrence relation for $H(n)$?

Solution: In each iteration, we print `Hello` a total of n^2 times where n is the input. We also make 3 recursive calls of size $n/4$, therefore the recurrence relation is:

$$H(n) = 3H\left(\frac{n}{4}\right) + O(n^2)$$

II. What is the tight upper bound for $H(n)$?

Solution: By using the Master Theorem on the above recurrence relation with $a = 3, b = 4, d = 2$, we get a tight upper bound given by:

$$T(n) = \Theta(n^2)$$