This Problem Set will be marked out of 50. There are four problems.

Please type (or neatly handwrite) your solutions on standard 8.5×11 paper, with your name at the top of each solution. Ensure that you submit you solutions in one file PDF file on Gradescope. **each problem sets solution should be on in its own individual page, Gradescope will help ensure you submit each solution under its correct problem number**

While a solution must be absolutely perfect to receive full marks, I will be generous in awarding partial marks for incomplete solutions that demonstrate progress.

So that there is no ambiguity, there are two non-negotiable rules. A violation of either rule constitutes plagiarism and will result in you receiving an F for this course.

(a) If you meet with a classmate to discuss any of the Individual Problems, your submission must be an individual activity, done in your own words, away from others. The process of finding a solution might take 3 - 5 iterations or even more BUT you learn from all these attempts and your confidence grows with each iteration.

(b) These problem sets might seem hard on a first look. They are designed to be so. We learn by attempting problems, struggling through them and coming on top. I encourage you to make this learning exercise worth your while. What do I mean? Open the problem sets as early as you get them, then do not look at hints or answers any where (including on the internet and consulting other students for direct answers), give it the best shot you can. If you get stuck come to Professor or TA's office hour and we shall be glad to listen to your rationale and work with you till you are able to tackle the problem sets.

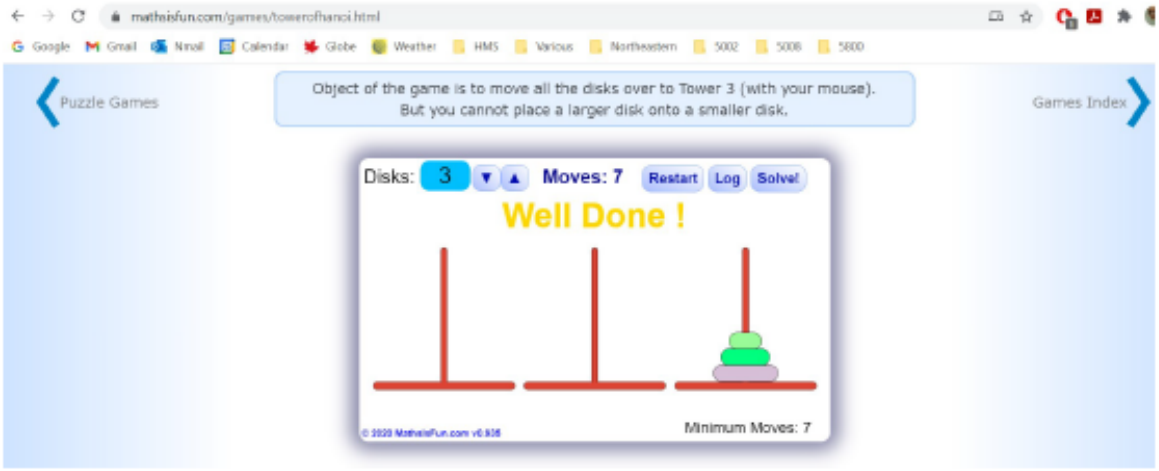
Problem #1 (10 points)

Play one or more games of "Towers of Hanoi", which you can do so on this website:

`https://www.mathsisfun.com/games/towerofhanoi.html`

There are three towers, and n disks. Your goal is to move all n disks from Tower 1 to Tower 3, but you may never place a larger disk on top of a smaller disk.

For example, when $n = 3$, the game can be solved in 7 moves, which is the optimal result.



(a) Attach a screenshot of you winning the $n = 4$ game in exactly 15 moves. (No proof or explanation is necessary – all you need to do is insert a .jpg image, just as I did above.)

(b) Let $T(n)$ be the minimum possible number of moves required to solve the game when there are n disks. For example, $T(2) = 3$ and $T(3) = 7$.

Clearly explain *why* $T(4) = 15$, showing it is possible to solve this game in exactly 15 moves and proving why it is impossible to solve this game in 14 (or fewer) moves.

(c) Find a recurrence relation for $T(n)$ and clearly and carefully explain why that recurrence relation holds. Then solve the recurrence relation using any method of your choice to determine a formula for $T(n)$ that is true for all integers $n \geq 1$.

(d) Substitute $n = \log(m)$ into your recurrence relation for $T(n)$ above, and use the Master Theorem to prove that $T(n) = \Theta(2^n)$. Briefly explain how and why your formula in part (c) is indeed $\Theta(2^n)$.

Problem #2 (10 points)

Quicksort is a powerful divide-and-conquer sorting algorithm that can be described in just four lines of pseudocode.

```
QUICKSORT(A, p, r)
1  if p < r
2      q = PARTITION(A, p, r)
3      QUICKSORT(A, p, q − 1)
4      QUICKSORT(A, q + 1, r)
```

The key to Quicksort is the PARTITION(A, p, r) procedure, which inputs elements p to r of array A, and chooses the final element $x = A[r]$ as the pivot element. The output is an array where all elements to the left of x are less than x, and all elements to the right of x are greater than x.

In class, we saw that there are very many techniques to partition the array. One nice technique covered in the book was invented by Nico Lomuto (See page 171 of the class textbook). You can also watch a quick youtube video here: `https://www.youtube.com/watch?v=86WSheyr8cM`. In this question, we will use the Lomuto Partition Method. Please assume that the pivot is *always* the last (right-most) element of the input array.

For example, if $A = [2, 8, 7, 1, 3, 5, 6, \mathbf{4}]$, then the pivot element is $x = A[8] = \mathbf{4}$, and PARTITION$(A, 1, 8)$ returns the array $[2, 1, 3, \mathbf{4}, 7, 5, 6, 8]$. We then run PARTITION on the sub-arrays $[2, 1, 3]$ and $[7, 5, 6, 8]$.

(a) Demonstrate the Quicksort algorithm on the input array $A = [3, 1, 5, 7, 6, 2, 4]$, showing how eventually the algorithm outputs the sorted array $[1, 2, 3, 4, 5, 6, 7]$. Clearly show all of your steps.

(b) When PARTITION is called on an array with n elements, we require $n - 1$ comparisons, since we must compare the pivot element to each of the other $n - 1$ elements.

If the input array is $A = [1, 2, 3, 4, 5, 6, 7]$, show that Quicksort requires a total of 21 comparisons.

(c) Determine an input array with 7 elements for which Quicksort requires the *minimum* number of total comparisons.

Clearly demonstrate why your input array achieves the minimum number of comparisons, and explain why there cannot exist a 7-element array requiring fewer comparisons than your array.

(d) Let A be an array with $n = 2^k - 1$ elements, where k is some positive integer. Determine a formula (in terms of n) for the *minimum* possible number of total comparisons required by Quicksort, as well as a formula for the *maximum* possible number of total comparisons required by Quicksort.

Use your formulas to show that the running time of Quicksort is $O(n \log n)$ in the best case and $O(n^2)$ in the worst case.

Problem #3 (10 points)

In this question, you will prove the Master Theorem in the special (and most important) situation when $f(n) = n^z$ for some real number z.

This result enables us to determine tight asymptotic bounds for various recurrence relations, which will help us tremendously in algorithm design and algorithm analysis.

If you can reproduce the proof of this result, then you will understand how the Master Theorem works in all situations – e.g. when $f(n) = 4n^3 + 2n^2 \log n + 5n + 100 \log n + 777$. But for the purposes of this problem, we will assume $f(n) = n^z$.

Let x, y, z be real numbers for which $T(1) = 1$ and $T(n) = xT(n/y) + n^z$.

(a) If $z < log_y(x)$, prove that $T(n) = \Theta(n^{log_y(x)})$.

(b) If $z = log_y(x)$, prove that $T(n) = \Theta(n^{log_y(x)} \log_y n)$.

(c) If $z > log_y(x)$, prove that $T(n) = \Theta(n^z)$.

(d) Some of you have taken a course in Linear Algebra, a core course in an undergraduate mathematics curriculum. In this course, students learn how to multiply two n by n matrices, and one can easily design an algorithm to perform matrix multiplication, running in $\Theta(n^3)$ time.

In 1969, Volker Strassen developed a recursive method to perform matrix multiplication, where the running time $T(n)$ can be given by the recurrence relation $T(n) = 7T(n/2) + n^2$.

Using any of the results in (a), (b), or (c), determine the running time of Strassen's Algorithm and show that it is faster than the standard algorithm that runs in $\Theta(n^3)$ time.

Note: if you are unable to prove (a), (b), (c) in its general form, you are welcome to instead solve the following simplified version of the problem, where you let $y = 2$ and $x = 16$, and prove the result for $z = 3$ in part (a), $z = 4$ in part (b), and $z = 5$ in part (c). If your proof is correct, you will be given 10 marks for solving the simplified version of the problem.

Problem #4 (20 points)

Let A and B be two *sorted* arrays, each with n elements. For the purposes of this problem, assume that all $2n$ elements are *distinct* integers, and assume $n = 2^k + 1$ for some integer $k \geq 1$.

In this problem, we wish to compute the *median* of the $2n$ elements, which will be the average of the n^{th} and $(n+1)^{\text{th}}$ smallest elements in the (combined) array.

For example, suppose $A = [1, 2, 4, 8, 16, 32, 64, 128, 256]$ and $B = [30, 40, 50, 60, 70, 80, 90, 100, 110]$. If we merge the arrays and sort it, we get $C = [1, 2, 4, 8, 16, 30, 32, 40, \mathbf{50}, \mathbf{60}, 64, 70, 80, 90, 100, 110, 128, 256]$. From this we see that the middle two elements are 50 and 60 and so the median is 55.

(a) Consider the following algorithm: merge the two arrays and sort it to produce a new array C with $2n$ elements. Then the median must be $(C[n] + C[n+1])/2$.

Determine whether this algorithm is *linear* or *linearithmic*. Clearly justify your answer.

(b) In our example above, we see that $A[5] < B[5]$ since $A[5] = 16$ and $B[5] = 70$.

Without knowing anything about the other numbers in these two 9-element arrays, clearly explain why $A[5] < B[5]$ implies that the correct answer *must* be equal to the median of the combined sub-arrays $A[5:9]$ and $B[1:5]$.

NOTE: $A[5:9]$ denotes elements 5 to 9 of array A, and $B[1:5]$ denotes elements 1 to 5 of array B. In our example above, $A[5:9] = [16, 32, 64, 128, 256]$ and $B[1:5] = [30, 40, 50, 60, 70]$. Indeed we see that the median of the combined sub-arrays is 55 since it is the average of the two middle numbers: $[16, 30, 32, 40, \mathbf{50}, \mathbf{60}, 64, 70, 128, 256]$. And this matches our correct answer of 55.

(c) Let $n = 2^k + 1$ for some $k \geq 1$.

If $A[(n+1)/2] > B[(n+1)/2]$, explain why the answer must be the median of the combined sub-arrays $A[1 : (n+1)/2]$ and $B[(n+1)/2 : n]$.

If $A[(n+1)/2] < B[(n+1)/2]$, explain why the answer must be the median of the combined sub-arrays $A[(n+1)/2 : n]$ and $B[1 : (n+1)/2]$.

(d) Using the result in part (c), design an algorithm that will determine the median of A and B. Clearly explain why your algorithm is guaranteed to return the correct output whenever $n = 2^k + 1$ for some $k \geq 1$. Finally, show that the running time of your algorithm is $O(\log(n+n)) = O(\log n)$. In other words, your algorithm is *logarithmic*, which is so much faster than linear or linearithmic!

(e) Using a programming language of your choice, solve Leetcode Problem #4 (Median of Two Sorted Arrays). For this part you shall need to provide a screenshot of your completed and accepted solution and an analysis of your solving the problem. Please follow leetcode directive and provide Space and Time complexity and a written detail of your problem solving approach.