**CS 5050 Final 99 Points**

This is an open book exam. This means you can use the internet, books, software, and personal notes to produce the answers. Think of it like a set of problems you must solve at work. There are limits though. The work you submit must be your own, so no collaboration with any other person (including but not limited to people in the class). No copy and pasting of any internet content. By submitting the completed exam, you are affirming that you have followed these rules.

You must submit your completed exam as one pdf file. You may enter your answers by editing this .doc file or printing it, writing on it and then scanning in. Or some combination of the two. Some of the answers require writing some code. For these, the best way would be to do a screen snip that shows the new code inserted into any existing code and/or as new functions. Add a few comments so I can understand what you did. The advantage of the snip is that the formatting will be preserved so the code will be easier to read.

For this last “assignment” I will not accept late work.

Please keep your answers concise. There is no need to give me an essay as an answer. There are 9 questions in this exam and they are each worth 11 points.

**Q1)** Use the meta algorithms to design a dynamic programming solution to the following problem:

You have solar panels that are mounted on a pole in your front yard. They have three settings that tip the panels at different angles with respect to the horizontal. As the seasons change, the settings allow you to capture more solar energy by better aligning the tipping angle of the panels with the position of the sun in the sky. Let these settings be called 0, 1, and 2.

You are given the following:

A table that maps the day-of-year (1 to 366) and the setting (0, 1, 2) to an estimate of the energy you could collect. Let this table be **E[d, s], 0 < d <= 366**, and **0 <= s < 3**. So **E[d, s]** will return the amount of energy you could collect if the panels are in setting **s** on day-of-year **d**.

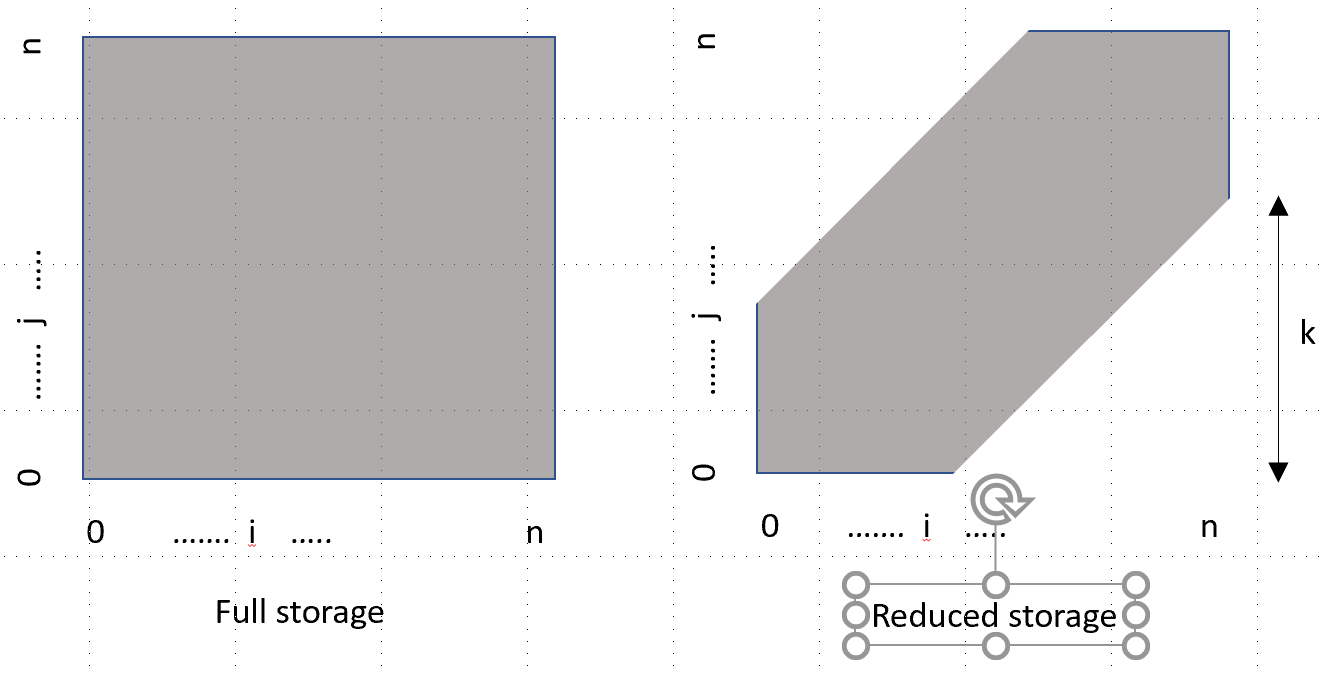
Design an algorithm that will work out the days you should change from one setting to another over the year in order to maximize your solar energy collection. Note: You can only make a maximum of four changes in settings in a year.

Design the recursive solution then convert it to a DP and write the traceback routine that will return the days (from 1 to 366) when an adjustment should be made and the type of adjustment (“on day 57 go to setting 2” for example)

**Q2)** The DNA alignment algorithm (and the minimum edit distance problem) are memory intensive. Answer the following questions.

Given two strings, both of size **n**, what is the memory required to perform an alignment?

One idea to save memory is to “cut off” the corners as illustrated in this figure, where only the grey area is allocated and used during the DP execution. How much memory would this save?



For the simple edit distance problem (where one edit step is scored as 1), how would this corner cut off change the algorithm’s behavior. Be precise.

For the DNA alignment problem, how would this memory reduction influence the result? If you were using this result to study evolutionary relationships, how would the distance in the “tree of life” relate to an appropriate setting for **k**?

What factor (as a function of **k**) would this change the run time?

**Q3)** Here is a variant of one of the best known google interview questions: Given a dictionary **D** which contains a set of words (strings) and a string **S**, determine ***how many ways*** **S** can be made from concatenating any subset of words contained within **D**.

If **D** = {“cat”, “dog”, “jim”, “fred”, “jimmy”, “my”, ”ed”, ”ment”, “em”, “body”, “embodiment”, “ i”}

And **S** = “jimmy”, your code would return 2.

Write the recursive and DP algorithms to solve this problem.

**Q4)** Write the simple minimum edit distance algorithm where a deletion, insertion or substitution is scored as 1 and the goal is to find the minimum number of edit steps. Answer the following questions:

Many kinds of typing errors (and mutations in DNA) involve switching the order of two adjacent letters. For instance, a classic when typing is to produce “hte” rather than “the”

Modify your recursive algorithm to include this kind of edit step and count it as 1.

Convert your code to a DP algorithm and show it works on some simple problems.

**Q5)** We have seen a few examples where we can identify two algorithms that solve the same problem: A simple or naïve algorithm that solve small problems quickly, but becomes impractical as the problems get large; and a clever algorithm that takes more time on smaller problems, but has a much slower growth rate in run time as problems get large. Answer the following questions:

Draw or show a graph that illustrates this pattern for a problem we solved in class. Make sure it shows the crossing point of run times of the algorithms.

Write some simple code that switches between the two algorithms based on these empirical results in the graph. What would be the run time of this algorithm? Mark it on the graph.

Considering the polynomial multiplication problem and two algorithms that solve this problem: “school” and the “three subproblem”, design an algorithm that uses each, but is faster than the simple switch algorithm. Show the algorithm below. No need to code and run this, you can just type up or write the algorithm.

**Q6)** Consider algorithm evaluation methods, both analytical and empirical.

* 1. For a recursive algorithm that generates 7 sub-problems of ¼ size with an overhead of *O(n)* each call, write the recurrence relation.
  2. Solve the recurrence relation using the master “cook book” method
  3. Briefly describe an empirical study you would perform to evaluate this algorithm and draw a graph of expected performance. Describe how you would solve for the actual function mapping problem size to run time given some timing results.
  4. For a recursive algorithm that generates 3 sub-problems of *n-1, n-2* and *n-3* problem sizes, write the recurrence relation
  5. Estimate the solution to the recurrence relation in the “big O” notation.
  6. Briefly describe an empirical study you would perform to evaluate this algorithm and draw a graph of expected performance. Describe how you would solve for the actual function mapping problem size to run time.

**Q7)** Read up on the traveling salesperson problem. Some simple code that solves the problem is included as part of this assignment (see canvas). The code enumerates all possible tours and keeps track of the best so far. When it returns, the global variable bestSolution will contain the shortest tour. Answer the following questions:

Write the recurrence relation for this algorithm that counts the number of base cases.

Write the solution to this recurrence relation.

Run the code for some simple problems going from 3 to 11 (don’t go too much higher) and time the computation.

Determine the appropriate graph for illustrating run time as a function of problem size and plot it.

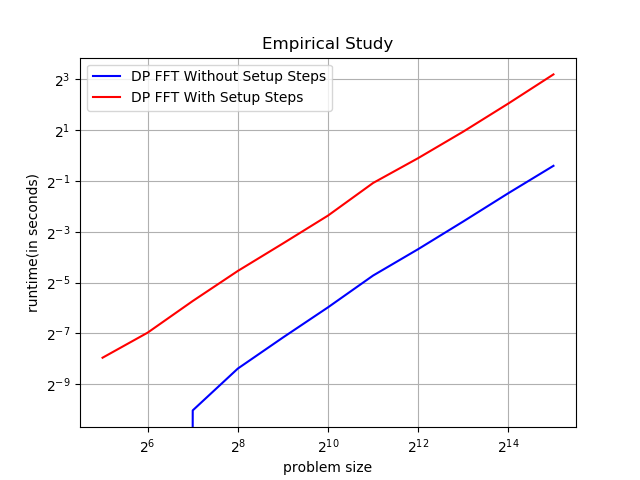
Consider the following observations:

1. The recursive algorithm keeps track of the length of the tour so far
2. The distances between cities are always positive so tours can only get longer as they are extended
3. The algorithm has a value for the best so far solution at every function call

Add an additional termination condition to the recursive algorithm that will exploit these observations to avoid unnecessary computation. Repeat the calculation above and plot the graph of this new (potentially quicker algorithm).

**Q8)** Many students were surprised to find that the iterative version of the FFT was not clearly faster than the recursive FFT. One reason is that the following steps are repeated each time: a) allocating the solution array, computing the omega tables, and computing the reverse-bit-shuffle to fill in the base cases.

In general, these steps may be done once only and used every time from there on. This is called amortization, where the time taken is spread out over all future runs of the algorithm. Do the following:

Rerun your iterative code, not counting these three setup steps and compare the results. Show a graph of the two versions of the code, one counting the setup steps, the other without. What is the speedup?

**As shown in the graph, without the set up steps the dp algorithm runs about 11 times faster.**

**Q9)** COVID-19 is a new virus that has mutated from previously observed viruses like SARS. COVID-19 continues to mutate as it spreads through the population. One of our assignments was to determine the mutation “distance” between two DNA sequences and identify the most likely mutations that occurred to change one sequence to the other. Do some research and identify and briefly describe three uses of the DNA alignment algorithm that is helping scientists understand COVID-19 and help find treatments to save lives.

* ChemRxiv uses DNA sequence alignment algorithm to compare the sequences and homology modeling of COVID-19 with other viruses Coronaviridae family. This study allows them to analyze the receptor binging sites of the samples to check for possible bonding with certain ligands with anti-viral properties. Knowing this will help with creating a vaccine.
* Because COVID-19 is an RNA virus, like the flu, it is more prone to mutation which makes it difficult to make a vaccine Dr. John Rose from the department of pathology at Yale Medicine is using the DNA Alignment algorithm to track how different the current virus spreading in the US from the original samples being studied from China. He has found that COVID-19 is mutating very slowly and when it does mutate the changes are very small.
* Nextstrain uses the DNA alignment algorithm to track how quickly COVID-19 is mutating as well as where in the genome the mutations occur. This helps with drug development so that treatments for the illness will not have adverse effects.