Please read the following instructions carefully before you begin.

- Write your name and Net ID on the answer sheets cover page and your Net ID on each additional page. Answer each of the six questions on the answer sheets provided. One sheet was intentionally left blank to provide you with scratch paper.
- Questions are not necessary given in order of difficulty, so read through them all before you begin writing!
- This exam is closed book. No notes or calculators are permitted.
- You have two hours and 45 minutes to take the exam.
- Please turn in these question sheets, your answer sheets, and scratch paper at the end of the exam period.
- Feel free to ask for clarification on any of the questions.
- You’re almost there!

1. Consider the following variant of MERGESORT that recursively sorts three subarrays before merging them. The procedure 3WAYMERGE(A[1 .. n], ℓ, m) sorts the array A[1 .. n] assuming subarrays A[1 .. ℓ], A[ℓ + 1 .. m], and A[m + 1 .. n] are already sorted.

```plaintext
3WAYMERGE(A[1 .. n]):
if n > 1
  ℓ ← ⌈n/3⌉
  m ← ⌈2n/3⌉
  3WAYMERGE(A[1 .. ℓ])
  3WAYMERGE(A[ℓ + 1 .. m])
  3WAYMERGE(A[m + 1 .. n])
  3WAYMERGE(A[1 .. n], ℓ, m)
```

(a) (4 out of 10) Describe how to implement 3WAYMERGE(A[1 .. n], ℓ, m). You may call the standard MERGE(B[1 .. p], q) procedure discussed in class which sorts an array B[1 .. p] in O(p) time assuming B[1 .. q] and B[q + 1 .. p] are already sorted. You do not need to analyze your implementation, but it should run in O(n) time.

[Hint: You should do one or more calls to MERGE as a subroutine. Otherwise, prepare to write a lot of cases into a for loop.]
(b) **(2 out of 10)** Let \( C(n) \) denote the total number of calls to 3WayMerge during a run of 3WayMergeSort(\( A[1..n] \)). We can find an asymptotic bound for \( C(n) \) by solving the following recurrence:

\[
C(n) = 3C(n/3) + 1
\]

State the number of calls to 3WayMerge using big-Oh notation by solving this recurrence.

(c) **(2 out of 10)** Let \( T(n) \) denote the worst-case running time of 3WayMergeSort(\( A[1..n] \)). Give a recurrence definition for \( T(n) \) that is suitable for analyzing the running time of 3WayMergeSort. You may assume 3WayMerge(\( A[1..n], \ell, m \)) runs in \( O(n) \) time.

(d) **(2 out of 10)** Using big-Oh notation, state the worst-case running time of 3WayMergeSort(\( A[1..n] \)) assuming 3WayMerge(\( A[1..n], \ell, m \)) takes \( O(n) \) time.  

*Hint: You probably want to solve your recurrence from part (c).*

2. Suppose you are given a stack of \( n \) pancakes of different sizes. You want to sort the pancakes so that smaller pancakes are on top of larger pancakes. The only operation you can perform is a **flip**—insert a spatula under the top \( k \) pancakes, for some integer \( k \) of your choice between 1 and \( n \), and flip those \( k \) pancakes over.

![Figure 1. Flipping the top four pancakes.](image)

Please note, this problem has nothing to do with the Tower of Hanoi.

(a) **(4 out of 10)** Describe an algorithm that given an arbitrary stack of \( n \) pancakes, moves the largest pancake to the bottom of the stack using at most 2 flips. You do not need to consider the time taken for any other operations.

(b) **(6 out of 10)** Describe an algorithm to sort an arbitrary stack of \( n \) pancakes using \( O(n) \) flips. You do not need to consider the time taken for any other operations. You may assume your solution for part (a) is correct.

*Hint: Use recursion.*
3. In Homework 3, you described a backtracking recursive algorithm that given an array \( A[1..n] \) of characters, computed the number of partitions of \( A \) into words. For example, given the string ARTISTOIL, your algorithm should have returned 2 for the partitions ARTIST · OIL and ART · IS · TOIL.

Today, we’ll turn that slow backtracking algorithm into an efficient dynamic programming algorithm. We’ll assume that we have a subroutine \text{IsWord} \) that takes an array of characters as input and returns \text{True} if and only if that string is a “word”.

Let \( \text{NumPartitions}(i) \) denote the number of ways to partition the array \( A[i..n] \) into words. Recall, \([P] \) evaluates to 1 if the proposition \( P \) is true, and it evaluates to 0 otherwise. \( \text{NumPartitions}(i) \) can be defined recursively as follows:

\[
\text{NumPartitions}(i) = \begin{cases} 
1 & \text{if } i > n \\
\sum_{j=i}^{n} \left( [\text{IsWord}(i, j)] \cdot \text{NumPartitions}(j + 1) \right) & \text{otherwise}
\end{cases}
\]

(a) (2 out of 10) In what kind of memoization data structure should we store the solutions to all subproblems \( \text{NumPartitions}(i) \)? If you’re using a (multidimensional) array, be sure to state the indices we use.

(b) (2 out of 10) What is a good evaluation order for solving the subproblems so each subproblem is solved after the ones it is dependent upon?

(c) (2 out of 10) What will be the final space and time complexity of the dynamic programming algorithm?

(d) (4 out of 10) Write an iterative algorithm that computes the number of partitions of \( A[1..n] \) into words.

4. Through a surprising sequence of career moves, you’ve been tasked with designing the walkways for the new University of Northwestern Middle Texarkana. An initial collection of building sites have already been determined, and your job is to connect all of them via sidewalks. Unfortunately, very little money has been allocated for your project. You need to minimize the total length of side walks that will be built subject to the following two conditions: 1) Each sidewalk you choose to build has to directly connect a pair of buildings, and 2) for each pair of buildings \( i \) and \( j \), there exists some sequence of complete sidewalks one can follow to travel between buildings \( i \) and \( j \). (For simplicity, we’ll assume you cannot save money or create shortcuts by having sidewalks cross one another. The guaranteed route between any pair of buildings must take each sidewalk in the sequence from one end to the other.)

You are given a set of \( n \) building sites as a pair of arrays \( X[1..n], Y[1..n] \) such that each building \( i \) will be placed at location \((X[i], Y[i])\). You also have access to a procedure \text{Distance}(x_1, y_1, x_2, y_2) \) that returns the distance between locations \((x_1, y_1)\) and \((x_2, y_2)\) in constant time. We will pick which sidewalks to build by reducing to the standard minimum spanning tree problem.

(a) (2 out of 10) What set should we use for the vertices of the graph? (Either define the set plainly in English, or provide a short algorithm for creating a useful vertex set.)
(b) **(2 out of 10)** What set should we use for the edges of the graph? (Either define the set of edges plainly in English, or provide a short algorithm for creating a useful edge set.) Are the edges directed?

(c) **(2 out of 10)** What weights should we give the edges?

(d) **(2 out of 10)** How many vertices are there, and how many edges are there? Recall, there are \( n \) buildings in total.

(e) **(2 out of 10)** In terms of \( n \), how long will it take to build the graph and run a minimum spanning tree algorithm to pick which sidewalks to build?

5. Consider the directed graph \( G = (V, E) \) below with non-negative capacities \( c : E \to \mathbb{R}_{\geq 0} \) and an \((s, t)\)-flow \( f : E \to \mathbb{R}_{\geq 0} \) that is feasible with respect to \( c \). Each edge is labeled with its flow/capacity.

![Figure 2. An \((s,t)\)-flow \( f \). Each edge is labeled with its flow/capacity.](image)

(a) **(2.5 out of 10)** Draw the residual graph \( G_f = (V, E_f) \) for flow \( f \). Be sure to label every edge of \( G_f \) with its residual capacity.

(b) **(2.5 out of 10)** There should be one or more augmenting paths \( s = v_0 \to v_1 \to \ldots \to v_r = t \) in \( G_f \). Let \( F = \min_i c_f(v_i \to v_{i+1}) \) and let \( f' : E \to \mathbb{R}_{\geq 0} \) be the flow obtained from \( f \) by pushing \( F \) units through your augmenting path. In a new copy of \( G \), label its edges with the flow values for \( f' \). *You do not need to write the edge capacities.*

(c) **(2.5 out of 10)** Your new flow should have maximum value. What is the value of the maximum flow/capacity of the minimum cut in the given graph?

(d) **(2.5 out of 10)** List the vertices lying on the \( s \) side of a minimum \((s, t)\)-cut.
6. Recall the edit distance problem: We’re given two strings $A[1..m]$ and $B[1..n]$, and we’re asked for the minimum number of insertions, deletions, and substitutions needed to transform $A$ into $B$.

Consider a variant of the problem where different operations have different costs: inserting a character into $A$ has cost 3, deleting a character from $A$ has cost 5, and substituting one character of $A$ for a different character from $B$ has cost 1.

(a) (5 out of 10) Let $EditNew(i, j)$ be the minimum total cost of any set of insertions, deletions, and substitutions that transform $A[1..i]$ into $B[1..j]$ according to the new costs given above. Fill in the blanks to complete the following recursive definition of $EditNew(i, j)$. There is no need to justify your answer.

$$EditNew(i, j) = \begin{cases} 
5i & \text{if } j = 0 \\
if i = 0 \\
\min \left\{ 
EditNew(i, j-1) + \right. \\
\left. EditNew(i-1, j) + \right. \\
\left. EditNew(i-1, j-1) + \left[ A[i] \neq B[j] \right] \right\} & \text{otherwise}
\end{cases}$$

(b) (5 out of 10) Describe and analyze an efficient dynamic programming algorithm to compute the modified edit distance between two strings $A[1..m]$ and $B[1..n]$ as defined above. You do not need to justify correctness of your algorithm, but you should go through the standard memoization steps anyway to make sure your algorithm is correct. Don’t forget to explain your running time. You may assume your answer to part (a) is correct when designing your algorithm.