Final

- Do not open this quiz booklet until directed to do so. Read all the instructions on this page.
- When the quiz begins, write your name on the top of every page of this quiz booklet.
- You have 180 minutes to earn a maximum of 180 points. Do not spend too much time on any one problem. **Read through all problems first**, then solve them in an order that allows you to make the most progress.
- **You are allowed three double-sided letter-sized sheet with your own notes**. No calculators, cell phones, or other programmable or communication devices are permitted.
- Write your solutions in the space provided. Pages will be scanned and separated for grading. If you need more space, write "Continued on S1" (or S2, S3, S4, S5, S6) and continue your solution on the referenced scratch page at the end of the exam.
- Do not waste time and paper rederiving facts that we have studied in lecture, recitation, or problem sets. Simply cite them.
- When writing an algorithm, a **clear** description in English will suffice. Pseudo-code is not required. Be sure to **briefly** argue that your **algorithm is correct**, and analyze the **asymptotic running time of your algorithm**. Even if your algorithm does not meet a requested bound, you **may** receive partial credit for inefficient solutions that are correct.

Problem	Parts	Points
0: Information	2	2
1: Decision Problems	10	40
2: Debugging	3	9
3: Shelf Sort	3	12
4: 2D Priorities	1	16
5: Merged Anagrams	1	16
6: Positive Unicycle	1	15
7: Teleportation	1	15
8: Dinner Diplomacy	1	15
9: Sweet Tapas	1	20
10: Gokemon Po	1	20
Total		180

Name: _____

School Email: _____

Problem 0. [2 points] **Information** (2 parts)

(a) [1 point] Write your name and email address on the cover page.

(b) [1 point] Write your name at the top of each page.

Problem 1. [40 points] **Decision Problems** (10 parts)

For each of the following questions, circle either **T** (True) or **F** (False), and **briefly** justify your answer in the box provided (a single sentence or picture should be sufficient). Each problem is worth 4 points: 2 points for your answer and 2 points for your justification. **If you leave both answer and justification blank, you will receive 1 point**.

(a) **T** **F** If $f(c) = O(1)$ for a constant c and $f(n) = 4f(n/2) + \Theta(n^2)$ for $n > c$, and $g(c) = O(1)$ for a constant c and $g(n) = 8g(n/2) + \Theta(n)$ for $n > c$, then $f = \Omega(g)$.

(b) **T** **F** Given an array of n integers representing a binary min-heap, one can find and extract the maximum integer in the array in $O(\log n)$ time.

(c) **T** **F** Any binary search tree on n nodes can be transformed into an AVL tree using $O(\log n)$ rotations.

(d) T F Given an array of n key-value pairs, one can construct a hash table mapping keys to values in **expected** $O(n)$ time by inserting normally into the hash table one at a time, where collisions are resolved via chaining. If the pairs have unique keys, it is possible to construct the same hash table in **worst-case** $O(n)$ time.

(e) T F You implement a hash table using some specific hash function $h(k)$, and then insert a sequence of n integers into the hash table. Then looking up whether a queried integer is stored in this hash table will take expected $O(1)$ time.

(f) T F Given a graph where all edge weights are strictly greater than -3, a shortest path between vertices s and t can be found by adding 3 to the weight of each edge and running Dijkstra's algorithm from s.

(g) T F Given a directed acyclic graph having positive edge weights and only one source vertex s having no incoming edges, running Dijkstra's from s will remove vertices from the priority queue in a topological sort order.

(h) T F Given an unweighted directed graph having only one source vertex s having no incoming edges, a depth-first search from s will always generate a DFS tree containing a longest simple path in the graph.

(i) T F The problem of determining whether an undirected graph contains as a subgraph a complete graph on k vertices is in NP.

(j) T F If there is a polynomial time algorithm to determine whether an unweighted graph contains a Hamiltonian path, then there is also a polynomial time algorithm to determine whether a weighted graph contains a simple path with weight greater than or equal to k.

Problem 2. [9 points] **Debugging** (3 parts)

Given a binary max-heap stored in Python list `A`, the following code is supposed to extract and return the maximum element from the heap while maintaining the max-heap property, but does not always work; the code contains a bug.

```
1  def extract_max(A):
2      # Given array A satisfying the max-heap property,
3      # extracts and returns the maximum element from A,
4      # maintaining the max-heap property on the remainder of A
5      i, n = 0, len(A)
6      A[0], A[n - 1] = A[n - 1], A[0]                  # swap first and last
7      while True:
8          l = (2 * i + 1) if (2 * i + 1 < n) else i    # left child or i
9          r = (2 * i + 2) if (2 * i + 2 < n) else i    # right child or i
10         c = l if A[r] > A[l] else r                  # larger child
11         if A[i] >= A[c]:                             # satisfies max-heap!
12             break
13         A[i], A[c] = A[c], A[i]                      # swap down
14         i = c
15     return A.pop()                                   # return maximum
```

(a) [2 points] State the running time of the code above.

(b) [3 points] Fix the bug in the code by changing a single line.

(c) [4 points] Give an array A of four integers satisfying the max-heap property for which the original code does not behave as desired.

Problem 3. [12 points] **Shelf Sort** (3 parts)

You have n books in your room, and you want to store them on a single shelf. Each book has the same thickness. **Briefly** describe and justify an algorithm that is **best suited** to the following scenarios (in one or two sentences).

(a) [4 points] You want to store your books on a single shelf sorted by color. The color of each book is specified by tuple of red, green, and blue values, each in the range 1 to 16. Place your books on the shelf sorted first by red, then green, then blue value.

(b) [4 points] You are sick of the colorful ordering of your books, and instead want to sort your books by height. The only way to compare the height of two books is to take them off the shelf and compare them. You would like to sort your books by only removing two books from the shelf at a time, then putting them back in the same or swapped order. Sort your books by height using swaps.

(c) [4 points] You've bought three new books and want to add them to your bookshelf currently sorted by height, but you only want to move any book currently on the shelf at most once. Place the new books on the shelf, assuming you may take a book off the shelf, compare it with a new book, and place it back on the shelf wherever you like.

Problem 4. [16 points] **2D Priorities** (1 part)

A **2D priority queue** maintains a set of ordered integer pairs (x, y), and supports four operations:

- `insert(x,y)`: insert ordered pair (x, y) into the queue
- `extract(x,y)`: remove ordered pair (x, y) from the queue, or return `None` if queue is empty
- `extract_min_x()`: remove and return an ordered pair from the queue having smallest x coordinate, or return `None` if queue is empty
- `extract_min_y()`: remove and return an ordered pair from the queue having smallest y coordinate, or return `None` if queue is empty.

Describe a data structure that supports all four 2D priority queue operations, each in **worst-case** $O(\log n)$ time, where n is the number of pairs in the queue at the time of the operation. You may assume that each pair stored in the queue is unique.

Problem 5. [16 points] **Merged Anagrams** (1 part)

A list of words contains a **merged anagram triple** (a, b, c) if words a and b from the list can be concatenated to form an anagram of another list word c. For example, the list of words (`grail`, `happy`, `word`, `test`, `moths`, `algorithms`) contains the merged anagram triple (`grail`, `moths`, `algorithms`). Given a list of n words, where each word contains at least one and at most n English characters, describe an $O(n^2)$ time algorithm to determine whether the list contains a merged anagram triple. State whether your algorithm achieves a **worst-case**, **expected**, and/or **amortized** bound.

Problem 6. [15 points] **Positive Unicycle** (1 part)

Given source vertex s from a weighted directed graph $G = (V, E)$ having both positive and negative edge weights containing **exactly one positive weight cycle** (though possibly having many cycles of negative weight), describe an $O(|V|^3)$ algorithm to return a list of all vertices having a longest path from s traversing fewer than $|V|$ edges.

Problem 7. [15 points] **Teleportation** (1 part)

A **teleportation graph** is a graph where each vertex has been assigned a color: either black, red, or blue. A **teleportation path** in a teleportation graph is a sequence of vertices $v_1, \ldots, v_k$ such that every adjacent pair of vertices (v_i, v_{i+1}) is either an edge of the graph, or v_i is red and v_{i+1} is blue, i.e. the path may **teleport** from any red vertex to any blue vertex. The **cost** of a teleportation path is the sum of weights of all edges traversed by the path; teleportation contributes zero cost. Given a weighted directed teleportation graph $G = (V, E)$ having only positive edge weights, where each vertex has **at most four** out-going edges, describe an $O(|V| \log |V|)$ time algorithm to find the minimum cost teleportation path from vertex s to t that teleports **exactly two times** (if such a path exists). Significant partial credit will be given for a correct $O(|V|^2)$ time algorithm.

Problem 8. [15 points] **Dinner Diplomacy** (1 part)

Meresa Thay is organizing a state dinner for a large number of diplomats. Each diplomat has a **nemesis**: another diplomat who they despise the most. If a diplomat is served dinner after the diplomat's nemesis, the diplomat will file a complaint with the state department. A complaint from diplomat d_i will result in s_i units of scandal. Given a list containing the nemesis and amount of scandal associated with each of the n diplomats attending the dinner, describe an $O(n)$ time algorithm to determine whether there exists an order to serve dinner to the diplomats one-by-one resulting in no more than S total units of scandal. For example, if only two diplomats d_1 and d_2 attend, with $s_2 < s_1$, they would be nemeses of each other. There will be a serving order resulting in no more than S total units of scandal if $s_2 \leq S$, by serving dinner to d_1 first, then to d_2.

Problem 9. [20 points] **Sweet Tapas** (1 part)

Obert Ratkins is having dinner at an upscale tapas bar, where he will order many small plates which will comprise his meal. There are n plates of food on the menu, with each plate p_i having integer volume v_i, integer calories c_i, and may or may not be sweet. Obert is on a diet: he wants to eat no more than C calories during his meal, but wants to fill his stomach as much as he can. He also wants to order exactly s sweet plates, for some $s \in [1, n]$, without purchasing the same dish twice. Describe a dynamic programming algorithm to find a set of plates that maximizes the volume of food Obert can eat given his constraints.

Be sure to define a set of subproblems, relate the subproblems recursively, argue the relation is acyclic, provide base cases, construct a solution from the subproblems, and analyze running time. Correct but inefficient dynamic programs will be awarded significant partial credit.

Problem 10. [20 points] **Gokemon Po** (1 part)

Kash Etchum wants to play a new augmented reality game, Gokemon Po, where the goal is to catch a set of n monsters who reside at specific locations in her town. Monsters must be caught in a specified linear order: before Kash can catch monster m_i, she must have already caught all monsters m_j occurring before m_i in the order (for $j < i$). To catch monster m_i, Kash may either purchase the monster in-game for c_i dollars, or she may catch m_i for free from that monster's location. If Kash is not at the monster's location, she will have to pay a ride share service to drive her there. The **minimum possible cost** of transporting from monster m_i to monster m_j via ride sharing is denoted $s(i, j)$. Given a list of prices of in-game purchases and ride sharing trips between all pairs of monsters, describe a dynamic programming algorithm to determine the minimum amount of money Kash must spend in order to catch all n monsters, assuming that she starts at the location of monster m_1.

Be sure to define a set of subproblems, relate the subproblems recursively, argue the relation is acyclic, provide base cases, construct a solution from the subproblems, and analyze running time. Correct but inefficient dynamic programs will be awarded significant partial credit.

SCRATCH PAPER 1. DO NOT REMOVE FROM THE EXAM.

You can use this paper to write a longer solution if you run out of space, but be sure to refer to the scratch paper next to the question itself.

SCRATCH PAPER 2. DO NOT REMOVE FROM THE EXAM.

You can use this paper to write a longer solution if you run out of space, but be sure to refer to the scratch paper next to the question itself.

SCRATCH PAPER 3. DO NOT REMOVE FROM THE EXAM.

You can use this paper to write a longer solution if you run out of space, but be sure to refer to the scratch paper next to the question itself.

SCRATCH PAPER 4. DO NOT REMOVE FROM THE EXAM.

You can use this paper to write a longer solution if you run out of space, but be sure to refer to the scratch paper next to the question itself.

SCRATCH PAPER 5. DO NOT REMOVE FROM THE EXAM.

You can use this paper to write a longer solution if you run out of space, but be sure to refer to the scratch paper next to the question itself.

SCRATCH PAPER 6. DO NOT REMOVE FROM THE EXAM.

You can use this paper to write a longer solution if you run out of space, but be sure to refer to the scratch paper next to the question itself.