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. Skim them all first, and attack them in the 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, S7) 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 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.
- **Pay close attention to the instructions for each problem**. Depending on the problem, partial credit may be awarded for incomplete answers.

Problem	Parts	Points
0: Information	2	2
1: Decision Problems	10	40
2: Alternate Sorts	2	16
3: Biscuit Brute	1	15
4: Blue Friends	1	15
5: Trek Timing	1	20
6: Boxing Books	1	16
7: Parity Proximate	1	18
8: Minion Matching	1	18
9: Cache Cost	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 $T(n) = 2T(n/4) + \sqrt{n}\log n$ with $T(1) = \Theta(1)$, then $T(n) = \Theta(\sqrt{n}\log n)$.

(b) **T** **F** A $\Theta(n^{10})$-time algorithm will always take at least as long to run as a $\Theta(\log n)$-time algorithm on a problem with input size n.

(c) **T** **F** Given an array of n distinct comparable integers, we can identify and sort the $\frac{n}{\log n}$ smallest of them in $O(n)$ time using a heap.

(d) T F Given k distinct integer keys, there exists a binary search tree containing all k of them that satisfies the max heap property.

(e) T F Checking if a string of length k exists in a hash table takes worst-case $O(k)$ time.

(f) T F Using an AVL tree as Dijkstra's priority queue results in the same asymptotic running time as using a Fibonacci heap, when running Dijkstra's algorithm to compute single source shortest paths on a graph with n vertices and $O(n)$ edges.

(g) T F Depth-first search solves single-source shortest paths in an unweighted, directed graph $G = (V, E)$ in $O(|V| + |E|)$-time.

(h) T F Given a connected weighted directed graph having positive integer edge weights, where each edge weight is at most k, we can compute single source shortest paths in $O(k|E|)$ time.

(i) T F If **P** = **NP**, it follows that **NP-Hard** = **EXP**.

(j) T F Let A be a problem that cannot be solved in polynomial time, and let ALG be an algorithm with optimal running time $T(n)$ to solve a size n instance of another problem B. If there is an algorithm to solve a size m instance of problem A in time $O(m^c) + T(m^d)$ for some constants c and d by using ALG, then problem B also cannot be solved in polynomial time.

Problem 2. [16 points] **Alternate Sorts**

(a) [8 points] Given an array A containing n integers, where each integer is in the range $\{1, \ldots, n^m\}$, describe an $O(n \cdot \min\{m, \log n\})$-time algorithm to sort A.

(b) [8 points] Recall that an array of **distinct** integers is ***k-proximate*** if every integer of the array is at most k places away from its place in the array after being sorted[1]. Describe an efficient[2] **in-place** algorithm to sort a k-proximate array.

[1] If the ith integer of the unsorted input array is the jth largest integer contained in the array, then $|i - j| \leq k$.

[2] By "efficient", we mean that faster correct algorithms will receive more points than slower ones.

Problem 3. [15 points] **Biscuit Brute**

The Biscuit Brute is a sleepy monster who lives in the mountains and loves to eat biscuits. Fortuitously, the mountains are home to a community of n bakers who bake lots of biscuits. If the Brute ever visits the home of a baker, the baker will give the Brute exactly k biscuits to eat. Not wanting to take advantage of their generosity, after receiving biscuits from one baker, the Brute will always travel to another baker to beg for biscuits before returning.

The Brute has an integer **sleepiness**. Sleepiness increases by 1 each minute the Brute is awake, and decreases by 10 whenever the Brute eats a biscuit. For every ordered pair of bakers (b_i, b_j), the Brute knows how many minutes m_{ij} it will take to travel from baker b_i to baker b_j (assume it takes no time to receive or eat biscuits). Describe an efficient **polynomial-time** algorithm to determine the minimum sleepiness the Brute can have to reach the house of baker b_n, starting from the house of baker b_1 with sleepiness -10.

Problem 4. [15 points] **Blue Friends**

A **blue-red** graph is a **simple**[3] undirected graph where every edge is colored either red or blue. A blue-red graph is **blue friendly** if every pair of vertices is reachable from each other via a path of only blue edges. Given a (simple) blue-red graph $G = (V, E)$ that is not blue friendly, describe an $O(|V|^2)$-time algorithm to determine whether you can add blue edges to G to make a new (simple) blue-red graph G' that is blue friendly.

[3]Recall, a graph is simple if every edge in the graph connects two distinct vertices, and there is at most one edge connecting any vertex pair.

Problem 5. [20 points] **Trek Timing**

Steryl Chrayed wants to visit Old Loyal, the geyser in Bluerock National Park which **erupts** exactly on the hour every 60 minutes. Steryl has a map showing the m two-way trails connecting n hiking junctions in the park. Each trail t_i is marked with its positive integer **length** ℓ_i measured in hundreds of meters, and its positive integer **difficulty** d_i. Steryl will leave at precisely 6:00 a.m. from the parking lot at junction p, and hike to Old Loyal at junction g, but wants to do so by hiking at a constant pace of exactly one hundred meters per minute along trails without stopping[4], so that Old Loyal erupts precisely when she first arrives at g. Given Steryl's map, describe an efficient **polynomial-time** algorithm to determine whether it is possible for Sheryl to do this, and if so, return such a hiking path that minimizes the sum of trail difficulties hiked.

[4]Steryl will always continue at the same pace in the same direction along a trail until reaching a junction.

Problem 6. [16 points] **Boxing Books**

The semester has ended, and Tim the Beaver needs to pack up the books in his dorm room. All of Tim's books have the same height and width, but each book has a different thickness proportional to the positive number of pages in the book. Tim has **three** identical boxes that can each fit at most m pages of books. Given a list $P = (p_1, \ldots, p_n)$ of the page length of each of his n books, describe an $O(nm^2)$-time dynamic programming algorithm to determine whether Tim can fit all of his books into his boxes. (Each book must fit into one of the boxes; do not break Tim's books!)

Problem 7. [18 points] **Parity Proximate**

A sequence of integers is **parity proximate** if every even integer in the sequence is adjacent to another even integer, and every odd integer in the sequence is adjacent to another odd integer. Given a sequence of integers $A = (a_1, \ldots, a_n)$, describe an $O(n)$-time dynamic programming algorithm to determine the maximum sum of any parity proximate subsequence of A. For example, if $A = (4, 3, 6, 5, 7, 2, 1, 4)$, then the subsequence $(4, 6, 5, 7, 2, 4)$ is parity proximate and has sum 28, which is maximum over all parity proximate subsequences of A.

Problem 8. [18 points] **Minion Matching**

Supervillain Shrunk leads a team of social minions. Each minion has a unique **name**: a string of at most k lowercase English letters. Each pair of minions has at most two **buddy names**, formed by the concatenations of their names in either order (note it is possible for a pair of minions to have only one buddy name if both concatenations are the same). Four minions are **buddy buddies** if a buddy name of two of them is identical to a buddy name of the other two. For example, minions `'alex'`, `'isabel'`, `'alexis'`, and `'abel'` are buddy buddies.

Given a list $A = (a_1, \ldots, a_n)$ containing the names of Shrunk's n minions, describe an $O(kn^2)$-time algorithm to determine whether any four of them are buddy buddies. State whether your algorithm's running time is worst-case, amortized, and/or expected.

Problem 9. [20 points] **Cache Cost**

A computer program accesses n variables $V = (v_1, \ldots, v_n)$ over time. When a program accesses a variable v, we would like to know the **cache cost** of the access: specifically, the number of **distinct** variables accessed since variable v was last accessed (if v has not been accessed before, its access cost is zero). For example, if a program accesses variables $V = (a, b, c, d)$ in the order (a, b, c, d, b, a, b), then the cache cost of the accesses are:

access #	1	2	3	4	5	6	7
variable	a	b	c	d	b	a	b
cache cost	0	0	0	0	2	3	1

Accesses 1-4 are first accesses so each have cost 0; access 5 has cost 2 because c and d are accessed between the first two accesses of b; access 6 has cost 3 because b, c, and d are accessed between accesses of a; and access 7 has cost 1 because a is accessed between the last two accesses of b.

Describe a dynamic data structure to store variable accesses over time from the start of a program, supporting one operation, `access(`v`)`, which accesses variable v and returns the cost of the access. The operation should run in $O(\log n)$ time where n is the number of variables accessed by the program. State whether your operation running time is worst-case, amortized, and/or expected.

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 write "Continued on S1" on the problem statement's page.

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 write "Continued on S2" on the problem statement's page.

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 write "Continued on S3" on the problem statement's page.

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 write "Continued on S4" on the problem statement's page.

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 write "Continued on S5" on the problem statement's page.

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 write "Continued on S6" on the problem statement's page.

SCRATCH PAPER 7. 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 write "Continued on S7" on the problem statement's page.