Final Exam

- Don't Panic.
- Write your name on every page.
- The final exam contains six problems. You have 120 minutes to earn 100 marks.
- The final exam contains 5 pages, including this one.
- The final exam is closed book. You may bring one double-sided handwritten "cheat sheet" of A4 paper to the quiz. (You may not bring any magnification equipment!) You may **not** use a calculator, your mobile phone, or any other electronic device.
- Write your solutions for each problem on separate pieces of paper. **Make sure your name is on every page.**
- Read through the problems before starting. Do not spend too much time on any one problem. **Difficulty is not necessarily correlated with number of marks.**
- Show your work. Partial credit will be given. You will be graded not only on the correctness of your answer, but also on the clarity with which you express it. Be neat.
- Good luck!

Problem #	**Name**	**Possible Points**	**Achieved Points**
1	Sorting jumble	12	
2	Independent subsystems of equations	15	
3	Tree partition	25	
4	Decrease-key	13	
5	Computerlandia	15	
6	Not one of them thought of a cow.	20	
Total:		100	

Name: _____ Student id.: _____

Problem 1. Sorting jumble. [12 points]

The first column in the table below contains an unsorted list of words. The last column contains a sorted list of words. Each intermediate column contains a partially sorted list.

Each intermediate column was constructed by beginning with the unsorted list at the left and running one of the sorting algorithms that we learned about in class, stopping at some point before it finishes. Each algorithm is executed exactly as described in the lecture notes. (One column has been sorted using a sorting algorithm not listed.)

Identify, below, which column was (partially) sorted with which algorithm. *Hint: Do not just execute each sorting algorithm, step-by-step, until it matches one of the columns. Instead, think about the invariants that are true at every step of the sorting algorithm.*

Unsorted	**A**	**B**	**C**	**D**	**E**	**F**	**Sorted**
Juliett	Alfa	Bravo	Bravo	Delta	Mike	Alfa	Alfa
Bravo	Bravo	Juliett	Alfa	Bravo	Lima	Bravo	Bravo
Kilo	Juliett	Kilo	Foxtrot	Echo	Kilo	Charlie	Charlie
Lima	Kilo	Lima	Charlie	Hotel	Juliett	Delta	Delta
Papa	Lima	Alfa	Juliett	Golf	Delta	Echo	Echo
Alfa	Papa	Charlie	Delta	Alfa	India	Juliett	Foxtrot
Foxtrot	Foxtrot	Foxtrot	Kilo	Foxtrot	Hotel	Foxtrot	Golf
Charlie	Charlie	Papa	India	Charlie	Charlie	Kilo	Hotel
Oscar	Oscar	Oscar	Golf	India	Foxtrot	Oscar	India
Delta	Delta	Delta	Hotel	Juliett	Echo	Lima	Juliett
November	November	November	Lima	November	Bravo	November	Kilo
India	India	India	Echo	Oscar	Alfa	India	Lima
Golf	Golf	Golf	Mike	Papa	Golf	Golf	Mike
Hotel	Hotel	Hotel	November	Lima	November	Hotel	November
Mike	Mike	Mike	Oscar	Mike	Oscar	Mike	Oscar
Echo	Echo	Echo	Papa	Kilo	Papa	Papa	Papa
Unsorted	**A**	**B**	**C**	**D**	**E**	**F**	**Sorted**

Please write the proper number in the blank space beside the letter:

A _ ☐ 1. BubbleSort

B _ ☐ 2. SelectionSort

C _ ☐ 3. InsertionSort

D _ ☐ 4. MergeSort (top down, sorts top half before bottom half)

E _ ☐ 5. QuickSort (with first element as the pivot)

F _ ☐ 6. None of the above

Problem 2. Independent subsystems of equations. [15 points]

A *system of equations* over some algebraic structure $(G, \oplus)$ is a list of equations, each of which is of the form $a \oplus b = c$. Each component (a, b, c) is either a constant $k \in G$ or a variable x, y, etc. For example, if G is the integers, and $\oplus$ is just the usual addition on integers, then the following is a system:

$$\begin{array}{rcrcr} x & + & 7 & = & y \\ 7 & + & z & = & y \\ x & + & z & = & 20 \\ u & + & w & = & 0 \\ v & + & 12 & = & u \end{array}$$

A *solution* to such a system is an assignment of elements of G to variables that makes each equation in the system hold. For example, the above system has the solution $\{u = 0,\ v = -12,\ w = 0,\ x = 10,\ y = 17,\ z = 10\}$. In general, depending on the exact nature of the set G and combining operation $\oplus$, these systems can be very difficult to solve. Accordingly, if one wishes to write a program that can solve such equations, it is useful to be able to split the initial problem into independent smaller subproblems. One way to do this is to separate the original system according to the variables used, with the observation being that subsystems of equations that contain no variables in common are independent. For example, the above system can be decomposed into the following two independent subsystems:

$$\begin{array}{rcrcr} x & + & 7 & = & y \\ 7 & + & z & = & y \\ x & + & z & = & 20 \end{array} \qquad \text{and} \qquad \begin{array}{rcrcr} u & + & w & = & 0 \\ v & + & 12 & = & u \end{array}$$

Since the subsystem on the left and the subsystem on the right contain no variables in common, any solution (*i.e.* assignment of values to variables) to the left-hand subsystem will not interfere with any solution to the right-hand subsystem. If—speaking hypothetically—we could solve a system with n equations in $O(n^5)$ time, then this decomposition reduces the time to solve the above system from $5^5 = 3,125$ time-units to $3^5 + 2^5 = 243 + 32 = 275$ time-units: a significant improvement!

Your task.

(a) [10 points] Please explain an algorithm that can decompose/separate an input system of equations Σ into a list of independent subsystems $\Sigma_1,\ \Sigma_2,\ \ldots,\ \Sigma_k$ by variables. To get full marks, your algorithm should be efficient.

(b) [5 points] Please give a good tight asymptotic (big-O) for your algorithm, in terms of n as the number of equations in the input system Σ. Defend your answer.

You can assume that the data comes to you in some convenient way, but should carefully explain any such assumptions. If you want to use an algorithm and/or data structure we have covered in class as part of your solution, then you don't have to explain how it works as long as you're using it without any modifications.

Problem 3. Tree partition [25 points]

Suppose you have a binary search tree (BST) τ, each of whose nodes are implemented in Java as per the following class (this code is identical to the BST code given out as part of lecture 10):

```
class BSTVertex {
  BSTVertex(int v) { key = v; parent = left = right = null; height = 0; }
  public BSTVertex parent, left, right;
  public int key;
  public int height; // unused for BST, needed for bBST
  public int size;   // unused for BST, needed for bBST
}
```

Now suppose someone gives you a partition value p, and asks that you split τ into two BST, the first of which τ_l contains all elements $\leq p$ and the second of which τ_r contains all of the elements $> p$. Note that both τ_l and τ_r need to satisfy the BST property.

Your task.

(a) [10 points] Please explain clearly how this can be done. Maximum marks require an **efficient** solution. If you find it helpful to draw a diagram as part of your explanation, please feel free.

(b) [5 points] Please give a good asymptotic (big-O) bound for your strategy from (a), regardless of whether it was efficient or not.

(c) [5 points] Write a Java function `partition` that does this partitioning, taking a `BSTVertex` root and `int` partition value as arguments and returning a size-2 array of `BSTVertex`es that points to τ_l (at index 0) and τ_r (at index 1).

(d) [5 points] Suppose that τ was actually a balanced binary search tree (bBST), in particular an AVL tree. What can you say about the balanced-ness of τ_l and τ_r, given your partition strategy?

Problem 4. Decrease-key [13 points]

Recall from lecture note 16 that Dijkstra's algorithm requires a more advanced form of heap: one that supports a *decrease-key* operation.

Your task.

(a) [3 points] What is a decrease-key operation and why does Dijkstra's algorithm need it?

(b) [5 points] Please give an explanation for how such a heap can be implemented.

(c) [5 points] Your heap strategy in (b) should take no more than $O(\log n)$ time per insert/remove-min/decrease-key operation. This implies that you can get full marks in (b) even if your heap takes more than $O(\log n)$ time per operation. **Hint.** If you are having a hard time thinking about a strategy for (b), I encourage you to think about relatively simple solutions that "get the job done," even if it means that you won't get marks for (c).

Problem 5. Computerlandia [15 points]

You have been put in charge of building the power lines in the fast-rising country of Computerlandia. As you might imagine, everyone in this country needs electrical power to go about their daily tasks. However, since the country is rather new, and is spending a significant fraction of its GDP on Netflix subscriptions, the power grid needs to be developed as cheaply as possible. You are given a list of coordinates of each of the cities in (latitude, longitude) form.

Your task.

(a) [10 points] Assuming that the distance between cities is calculated using the Pythagorean method (i.e. that Computerlandia is far enough away from the poles that the curvature of the Earth has an immaterial effect on this problem), how can you determine the optimum placements for the electrical wires, so that each city is connected to each other city (not necessarily directly connected: indirect connections via other cities are fine).

(b) [5 points] What is the asymptotic (big-O) running time of your algorithm? Explain.

Problem 6. Not one of them thought of a cow. [20 points]

> The cheese-mites asked how the cheese got there,
> And warmly debated the matter;
> The Orthodox said that it came from the air,
> And the Heretics said from the platter.
> They argued it long and they argued it strong,
> And I hear they are arguing now;
> But of all the choice spirits who lived in the cheese,
> Not one of them thought of a cow.
>
> – Sir Arthur Conan Doyle, 1898

Two cheese mites, Taran and Eilonwy, find themselves wandering in a large block of Swiss cheese. Such a cheese has many holes of a variety of sizes; a "basic" hole is a sphere, and so can be specified by a 4-tuple (x, y, z, r) giving the 3D coordinates and radius of the hole. When basic holes intersect, the resulting shape is called a compound hole; these can be quite complex if enough basic holes are involved. Taran and Eilonwy are initially located in different holes. Since they wish to debate philosophy, they are determined to find each other. Being cheese mites, they can eat their way through the cheese. However, being somewhat vain, they wish to eat the minimum amount of cheese necessary. Thus, they must travel from hole to hole, eating when the way forward is blocked by cheese (to travel from one basic hole in a compound hole to another basic hole in that same compound hole doesn't require them to gain any weight: they just walk on the inner surfaces). You should also assume that the "outer surface" of the cheese is sufficiently far away that it doesn't affect the optimum route (*i.e.*, that the cheese is infinitely large).

Your task.

(a) [15 points] Design an algorithm that, given a list of the basic holes in the cheese, as well as the starting holes for our mites, computes the optimal path for Taran and Eilonwy.

(b) [5 points] What is the asymptotic (big-O) running time of your algorithm? Explain.

END OF EXAM