Weight: 20 points

Directions: Do not discuss this exam with those who have not taken it. You have two hours to complete this exam. The exam is open book, open notes, in paper form only. You may not use any electronic devices or information provided by other students. Do not open the exam until instructed to do so.

In your answers, you may directly refer to the lectures, textbooks, homework solutions, or readers. Do not write down any extraneous, impertinent, or incorrect information – you may be penalized if you do. Please write your answers on the exam itself, using the back of each page if necessary. You may use any standard library routines in your code except where prohibited by the problem statement. You are advised to read the entire exam and begin with questions that you find to be easy.

Be sure that your name and login are on every sheet of this exam.

Your Name: _______________________

Your Login: _______________________

Your TA’s Name: _______________________

1. _______ / 7
2. _______ / 5
3. _______ / 4
4. _______ / 4

TOT _______ / 20
Throughout this exam, assume the following declarations are available:

    /* Represents a user login or logout at a certain time */
    class Event {
        String username;
        int time;
    }

    /* An Event Binary Search Tree */
    class BST {
        Event data;
        BST left, right;

        /* Constructs a BST */
        BST (Event data, BST left, BST right) {
            this.data = data;
            this.left = left;
            this.right = right;
        }

        other methods not shown
    }

    /* A Vector */
    class Vector {

        /* Constructs an empty Vector */
        Vector() { ... }

        /* Adds the given element to the Vector */
        void add(Object o) { ... }

        /* Returns the item at the given index in the Vector */
        Object get(int index) { ... }

        /* Returns the number of elements in the Vector */
        int size() { ... }

        /* Removes the item at the given index from the Vector */
        void remove(int index) { ... }

        /* Returns whether the Vector contains the given element (as given by equals()) */
        boolean contains(Object o) { ... }

        other methods not shown
    }
1. **(User Logins)** Recently, on a local multiuser computer system, the security system was cracked allowing sensitive data to be leaked. The system’s administrators are able to pinpoint approximately when the security breach occurred, within a certain range of times. They ask you to determine who was logged into the system to help find the culprit.

As part of the logging mechanism, the system automatically records each login and logout of a user. One BST keeps information about user logins – each time a user logs into the system, a new item is inserted into the BST with the time as the key. Similarly, when a user logs out of the system, a new item is inserted into a separate BST that records logouts. The BSTs store these values as instances of `Event`. A user may not be logged in more than once at the same time.

a. Given the definitions on the previous page (including partial definitions of `BST`, `Event`, and `Vector`), fill in the following method `userLogins` that returns a `Vector` of the usernames that were logged in between the given start time and end time. Times are stored as `int` values (the number of seconds since the system was turned on). Also, a user is considered to be “logged in” during a range if she did not log out before the start time or log in after the end time. You may assume that no events occur simultaneously.

```cpp
/* Computes and returns a Vector of the usernames that were logged into the system between start and end. logins is a BST that stores Event instances indicating user logins. logouts is a BST that stores Event instances indicating user logouts. */
static Vector userLogins(int start, int end, BST logins, BST logouts) {
    // FILL IN
}
```

Continued on next page.
b. After using your code for some time, the system’s administrators return to you with a complaint – it works great, but it’s far too slow. However, the real performance issue lies in the way the system is storing information about logins and logouts. What’s bad about the way the system is storing this information? Carefully describe its problems and give a tight asymptotic bound on the running time of your implementation of `userLogins`.

c. Propose an optimal means of storage of the information about logins and logouts so that storage is efficient (the system can store quickly and also not waste disk space/memory space) and information retrieval is fast (for operations such as `userLogins`). Include in your description the data structure you would use and how you would store information about logins and logouts in such a data structure. Remember that login/logout information is not all inserted at once, but rather, is stored as users log in and log out over time.

d. Describe how your `userLogins` method could be rewritten to take advantage of your new, efficient logging data structure.
2. (Synthesis) Answer the questions below with a short paragraph for each.

a. Some cellular phones keep a list of N frequently dialed numbers. When a call is made from the phone’s phone book, the phone increments the number’s call count in the frequently dialed numbers list, potentially moving the number up in the list (indicating that the number is “more frequently dialed”). Suppose the user of such a cell phone looks at this (long) list from the most frequently dialed to least frequently dialed, and often doesn’t look through the entire list.

Give a data structure and means of search and update of phone numbers in the list that optimize for speed of operation, based on the user behavior described. Give approximate running times (in terms of N) using asymptotic notation for search and update of phone numbers using your data structure.

b. Suppose a stack of N books of different sizes and a sheet of paper are on a table. You are asked to sort the books from smallest (physically) to largest so that the stack doesn’t fall over easily. Since the books are heavy and N is large, the only thing you can do is pull out one book from anywhere in the stack and place it at the top, but since you have quick eyes, you can immediately determine which book is the smallest or largest in some part of the stack (separated by the paper). You may insert the sheet of paper above or below any book in the stack.

Give a set of steps to quickly sort the stack of books (and say where you put the paper, if anywhere). Which sorting algorithm is this similar to? Why? Why is this “quick” for this problem?

c. A certain program reads input from a file by reading 1 kilobyte of the input at a time. Within the program, an array of bytes holds the already-read input such that every time another 1 kilobyte is read, an array 1 kilobyte larger than the current one is created, the old contents of the array are copied over, and the new kilobyte is added to the end. How could this method of storage of input data be improved to radically decrease the time spent reading input?
3. (Data Structures / Algorithms) In this question you are to answer short questions about data structures or algorithms.

a. Algorithm A runs in $O(n \log n)$ and algorithm B runs in $\Omega(n^2)$. Therefore, given the same input, A always takes less time to run than B. True or False? If true, explain why. If false, explain and give a counterexample.

b. Given the adjacency matrix representation of a directed weighted graph G below, draw G, labelling the nodes alphabetically from A to E (corresponding with indices 0 to 4 in the matrix). Then run breadth-first search starting at A (when there is a choice of next node, choose the node that is lower alphabetically), and write the order nodes are visited in during the search.

\[
\begin{pmatrix}
1 & 3 \\
1 & 2 & 2 \\
4 & 1 & 3 & 0 \\
& & & & 10
\end{pmatrix}
\]

c. Draw a legal minimum height B-Tree of order 5 that contains the following key values with 5 in the root node: 15, 30, 0, 10, 5, 35, 40, 45, 60, 25. Afterwards, show the resulting B-Tree upon removal of the key 5. Be sure to explain how you reorganize the tree during removal operations.

d. Suppose an input list of length 6 is sorted with merge sort. The following calls to merge are made (with the linked lists shown as arguments to merge) during the merge sort: merge(2, 7), merge(2 -> 7, 6), merge(4, 1), merge(1 -> 4, 3), merge(2 -> 6 -> 7, 1 -> 3 -> 4). After these merge calls, the list is sorted. What was the original (input) list?
4. (Cheap Cheese-Food) You are hired as a consultant by a cheese-food manufacturer, and are asked to help decide between several cheese-food manufacturing processes. Below is a sample graph of cheese-food manufacturing processes, in which each node represents a stage in the process and edges represent costs (in dollars per pound of starting ingredient) to ready the cheese-food-product for the next stage. For example, 1 pound of milk, 1 pound of Yellow #5, and 1 pound of glue can produce 3 pounds of Velveeta for $0.79 + $2.56 + $1.23, which then added to 1 pound of culture costs $3 \times 1.09 + $0.78 to turn into cheese-food. Note that several distinct starting ingredients are necessary for different portions of the manufacturing process.

Given such a graph, determine the cheapest way to manufacture cheese-food from its required starting ingredients. Your solution should be detailed steps (in pseudo-code) that determine the initial necessary starting ingredients, since you aren’t told which they are (in the example below, the starting ingredients happen to be milk, culture, Yellow #5, and glue), and the total minimum cost to produce cheese-food from them. Your solution should work not only on the graph below, but on any possible cheese-food manufacturing process. If you use any known algorithms in your solution, simply state their names, their input, expected output, and how you use the expected output.