

Midterm Assessment

9 Mar 2020

Time allowed: 2 hours

Instructions — please read carefully:

1. Do not open the midterm until you are directed to do so.
2. Read **all** the instructions first.
3. The midterm is closed book. You may bring one double-sided 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.
4. The **QUESTION SET** comprises **EIGHT (8) questions** and **FOURTEEN (14) pages**, and the **ANSWER SHEET** comprises of **FOURTEEN (14) pages**.
5. The time allowed for solving this test is **2 hours**.
6. The maximum score of this test is **100 marks**. The weight of each question is given in square brackets beside the question number.
7. All questions must be answered correctly for the maximum score to be attained.
8. All questions must be answered in the space provided in the **ANSWER SHEET**; no extra sheets will be accepted as answers.
9. You must submit only the **ANSWER SHEET** and no other documents. The question set may be used as scratch paper.
10. An excerpt of the question may be provided above the answer box. It is to aid you to answer in the correct box and is not the exact question. You should refer to the original question in the question booklet.
11. You are allowed to use pencils, ball-pens or fountain pens, as you like as long as it is legible (no red color, please).
12. 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!

This page is intentionally left blank.

It may be used as scratch paper.

Question 1: Sorting Jumble [12 marks]

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 that you have not seen in class. (Recursive algorithms recurse on the left half of the array before the right half.)

Unsorted	A	B	C	D	E	F	Sorted
Mary	Eddie	Eddie	Alice	Alice	Alice	Eddie	Alice
Harry	Fred	Gina	Bob	Bob	Bob	Gina	Bob
Patty	Gina	Harry	Carol	Carol	Carol	Harry	Carol
Eddie	Harry	Kelly	Eddie	Dave	Dave	Fred	Dave
Gina	Ina	Mary	Gina	Eddie	Eddie	Alice	Eddie
Kelly	Kelly	Patty	Kelly	Fred	Fred	Ina	Fred
Ina	Mary	Ina	Ina	Gina	Ina	Bob	Gina
Fred	Patty	Fred	Fred	Kelly	Harry	Kelly	Harry
Alice	Alice	Alice	Dave	Mary	Gina	Carol	Ina
Noah	Bob	Noah	John	Noah	Kelly	John	John
Bob	Linda	Bob	Harry	Harry	John	Dave	Kelly
Linda	Noah	Linda	Linda	Linda	Ophelia	Linda	Linda
Carol	Carol	Carol	Mary	Patty	Patty	Mary	Mary
John	Dave	John	Noah	John	Noah	Noah	Noah
Dave	John	Dave	Patty	Ina	Mary	Ophelia	Ophelia
Ophelia	Ophelia	Ophelia	Ophelia	Ophelia	Linda	Patty	Patty
Unsorted	A	B	C	D	E	F	Sorted

Identify, below, which column was (partially) sorted with which of the following algorithms:

1. BubbleSort
2. SelectionSort
3. InsertionSort
4. MergeSort
5. QuickSort (with first element pivot)
6. None of the above.

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.

- A.** What sort was used on Column A?
[2 marks]
- B.** What sort was used on Column B?
[2 marks]
- C.** What sort was used on Column C?
[2 marks]
- D.** What sort was used on Column D?
[2 marks]
- E.** What sort was used on Column E?
[2 marks]
- F.** What sort was used on Column F?
[2 marks]

Question 2: Algorithm Analysis [15 marks]

For each of the following, choose the best (tightest) asymptotic function from among the given options. Some of the following may appear more than once, and some may appear not at all. Write the letter indicating the proper bound in the answer box.

A. $\Theta(1)$	B. $\Theta(\log n)$	C. $\Theta(n)$	D. $\Theta(n \log n)$
E. $\Theta(n^2)$	F. $\Theta(n^3)$	G. $\Theta(2^n)$	H. None of the above.

A.
$$T(n) = \left(\frac{\sqrt{n}}{17}\right) \left(\frac{\sqrt{n}}{4}\right) + \frac{n \log n}{1000}$$

[3 marks]

B.
$$T(n) = (2^n)(2^n)$$

[3 marks]

C.

$$T(n) = 2T(n/2) + n$$

$$T(1) = 1$$

[3 marks]

D. The asymptotic running time of the following code, as a function of n :

```
public static int loopy(int n){
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < i; j++) {
            System.out.println("Hello.");
        }
    }
    return 17;
}
```

[3 marks]

E. The running time of the following code, as a function of n :

```
public static int recursivelooopy(int n){  
    for (int i=0; i<n; i++) {  
        for (int j=0; j<n; j++) {  
            System.out.println("Hello.");  
        }  
    }  
  
    if (n <= 2) {  
        return 1;  
    } else if (n % 2 == 0) {  
        return (recursivelooopy(n+1));  
    }  
    else {  
        return (recursivelooopy(n-2));  
    }  
}
```

[3 marks]

Question 3: How do they work? [15 marks]

Each of the questions in this part investigates how one of the algorithms from class works.

A. The maximum number of rotations necessary to rebalance an AVL tree containing n elements during the insertion of a new item is:

- | | | |
|------|------|---------------------|
| A. 0 | C. 2 | E. $\Theta(\log n)$ |
| B. 1 | D. 3 | F. $\Theta(n)$ |

[3 marks]

B. The maximum number of rotations necessary to rebalance an AVL tree containing n elements during the deletion of an existing item is:

- | | | |
|------|------|---------------------|
| A. 0 | C. 2 | E. $\Theta(\log n)$ |
| B. 1 | D. 3 | F. $\Theta(n)$ |

[3 marks]

C. Assume a standard implementation of binary search that stops as soon as it finds the item being queried, as in the following (pseudo)code:

```
int search(int key, int[] A, int low, int high)
    if (low > high) return NOT_FOUND;
    mid = (low+high)/2;
    if (key == A[mid]) return mid;
    else if (key < A[mid]) return search(key, A, low, mid-1);
    else if (key > A[mid]) return search(key, A, mid+1, high);
```

We know (from class) that if you are searching for a key x in a sorted array of size n , and if the key appears in the array, then it will be found in $\Theta(\log n)$ time (in the worst case).

What if the key x appears in the (sorted) array $m \geq 1$ times? As a function of n and m , what is the worst-case running time of binary search for key x ?

- | | | |
|------------------------------|------------------------------|----------------------|
| A. $\Theta(\log(m))$ | D. $\Theta(\log(m)/\log(n))$ | G. None of the above |
| B. $\Theta(\log(n/m))$ | E. $\Theta(n/m)$ | |
| C. $\Theta(\log(n)/\log(m))$ | F. $\Theta(n)$ | |

[3 marks]

D. Assume that this array was just partitioned by a QuickSort partitioning algorithm:

[19, 7, 8, 1, 16, 25, 62, 47, 80]

Of the following options, which is a possible pivot?

- | | | |
|-------|-------|----------------------|
| A. 19 | C. 16 | E. 47 |
| B. 8 | D. 25 | F. None of the above |

[3 marks]

E. Assume that comparing two strings of length k_1 and k_2 takes $\min(k_1, k_2)$ time. The worst-case running time for inserting a string of length L into an AVL tree of size n where all the keys in the tree have length L is:

- | | | |
|-----------|------------------|--------------------|
| A. $O(1)$ | C. $O(\log n)$ | E. $O(\log n + L)$ |
| B. $O(L)$ | D. $O(L \log n)$ | F. $O(nL)$ |

[3 marks]

Question 4: Which is best? [15 marks]

We have seen several different data structures that support a similar set of operations, e.g., insertion, deletion, search, successor, predecessor, etc. For each of the following questions, choose one of the following three options to indicate which data structure is fastest. Assume each contains the same number of keys. (Note that for a randomized algorithm, by “worst-case” we mean the worst case cost for the worst possible random choices.)

- A. AVL tree
- B. Skip List
- C. Hash table (with chaining, load factor 1)

A. Which data structure has the fastest worst-case insertion time? [3 marks]

B. Which data structure has the fastest expected insertion time? [3 marks]

C. Which data structure has the fastest worst-case search time? [3 marks]

D. Which data structure has the fastest expected search time? [3 marks]

E. Which data structure has the fastest worst-case successor query time? [3 marks]

Question 5: Invariants [12 marks]

One of the best ways to understand how an algorithm works is to think about the invariants that it guarantees! In this problem, you will identify the invariants for a few different algorithms and data structures.

A. InsertionSort. Recall the following (pseudo)code for InsertionSort (where `swap(a, i, j)` swaps the items at index i and j in array a):

```
void InsertionSort(int[] array)
    int size = array.length;
    for (int i=1; i<size; i++)
        for (int j=i; j>0; j--)
            if (array[j-1] > array[j])
                swap(array, j-1, j);
            else break;
```

Which of the following is a good loop invariant for the outer loop in InsertionSort (where the loop invariant is evaluated at the end of the loop). (Choose one.)

- A. For all k such that $k < i$: $A[k] \leq A[k+1]$.
- B. For all k such that $j < k < i$: $A[k] \leq A[i]$.
- C. The subarray $A[0..i]$ contains the $i+1$ smallest elements in the array.
- D. The subarray $A[0..i-1]$ contains the i smallest elements in the array.
- E. None of the above.

[4 marks]

B. Which of the following are invariants for an AVL tree (evaluated at the end of every operation)? (Select all that apply.) Assume height is defined as in class, where a leaf has height 0.

- A. If node u and v are siblings, then $|\text{height}(u) - \text{height}(v)| < 2$.
- B. If node u is the parent of node v , then $|\text{height}(u) - \text{height}(v)| < 2$.
- C. If node u is the parent of node v , then $|\text{height}(u) - \text{height}(v)| > 0$.
- D. If node u has height h , then the number of nodes in the subtree rooted at u is at most 2^h .
- E. If node u has height h , then the number of nodes in the subtree rooted at u is at least 2^{h-2} .

[4 marks]

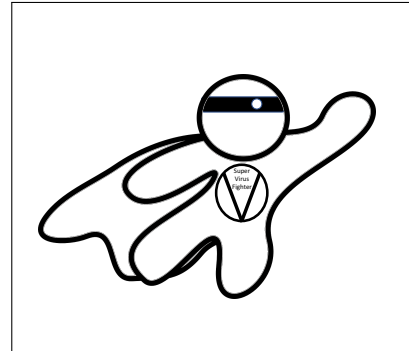
C. Which of the following are invariants for an (a, b) -tree (i.e., properties that are true at the end of every operation)? (Select all that apply.) Assume height is as defined as in an AVL tree, where a leaf has height 0. Define $\text{deg}(u)$ to be the number of children that node u has.

- A. If node u and v are siblings, then $|\text{deg}(u) - \text{deg}(v)| \leq b$.
- B. If node u and v are siblings, then $|\text{deg}(u) - \text{deg}(v)| \geq a$.
- C. If node u and v are siblings, then $|\text{height}(u) - \text{height}(v)| < 1$.
- D. If node u and v are siblings, then $|\text{height}(u) - \text{height}(v)| < 2$.
- E. If node u has height h , then the subtree rooted at u contains at least a^h nodes.
- F. If node u has height h , then the subtree rooted at u contains at least h^a nodes.

[4 marks]

Question 6: Super Virus Fighter [16 marks]

To pass the time while we are staying home due to pandemic viruses, you decide to design a brand new game: Super Virus Fighter. The game is a massive multiplayer game in which players from around the world play the heros of our world: the virus fighters! Throughout the game, when a player accomplishes a particular difficult feat, they may temporarily acquire certain special skills, e.g., immunity, cure, quarantine, sleep, etc. And if they ever acquire at least two (2) of these special skills, then (for a time) they become Super Virus Fighter.



To build this game, you are going to need a data structure. Your data structure needs to store the identities of the players, along with their current special skills. For the purpose of this question, we will identify players by their names (i.e., a string), and you can assume that all names are unique. There are exactly k special skills, and we will refer to the skills by (integer) number, e.g., $1, 2, \dots, k$.

The data structure should support two types of search operations. First, it should allow you to search for a player by name, returning their current set of skills. Second, it should support searching for a player with at least two of the special skills, i.e., it can find a Super Virus Fighter.

Your data structure should support the following operations:

- `insert(String name)`: adds a player to the data structure with the specified name.
- `addSkill(String name, int skill)`: updates the player with the specified name to have the specified skill.
- `deleteSkill(String name, int skill)`: updates the player with the specified name to not have the specified skill.
- `search(String name)`: returns the set of skills for the specified player.
- `searchSuper()`: returns the name of a player that has at least two skills (or null, if none have more than one skill).

Give a solution that involves a single data structure, which is augmented. Each operation should be implemented as efficiently as possible. (Less efficient solutions, if correct, will get some limited partial credit.) You may assume that you can compare two strings in $O(1)$ time (so the length of a name does not matter).

When describing your solution, you do not need to reproduce/restate an algorithm covered in class. You only need to describe how it needs to be changed to solve your problem. When asked for an algorithm, you may give pseudocode or describe the algorithm in words. (Java code is not needed.) You are encouraged to draw pictures to illustrate how your algorithm works.

Note: the marks given depend on the entire solution, not just the individual part.

[16 marks]

A. To solve the problem, which existing data structure will you start with? (E.g., array, linked list, AVL tree, hash table, etc.).

B. Describe and explain what you will do. What is stored in each cell/entry/node/bucket of your data structure? What additional information will you store in each cell/entry/node/bucket of your structure?

C. Draw a picture that illustrates how your data structure works.

D. How will you insert a new user? (Be sure to include how to maintain the additional information listed above.)

E. How will you implement `addSkill` and `deleteSkill`? (Be sure to include how to maintain the additional information listed above.)

F. Let N be the total number of items and k the total number of skills. What is the asymptotic worst-case running time of `insert`?

G. Let N be the total number of items and k the total number of skills. What is the asymptotic worst-case running time of `addSkill`, if the user being updated for has s skills?

H. How will you implement `search`?

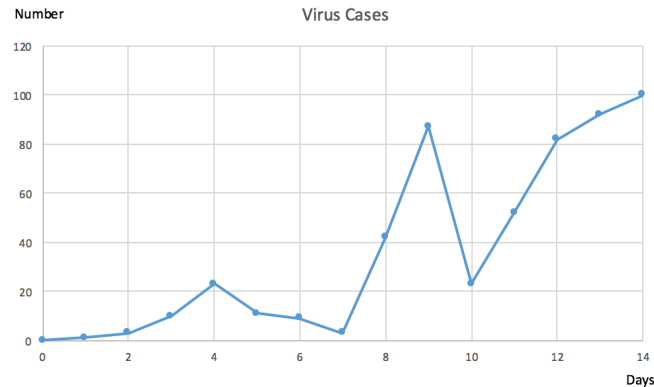
I. Let N be the total number of items and k the total number of skills. What is the asymptotic worst-case running time of `search`, if the user being searched for has s skills?

J. How will you implement `searchSuper`?

K. Let N be the total number of items and k the total number of skills. What is the asymptotic worst-case running time of `searchSuper`, if the user being searched for has s skills?

Question 7: What Goes Up Must Come Down? [15 marks]

The goal of your game Super Virus Hunters is, of course, to decrease the number of sick people in your simulated world. Therefore, as the game runs, you keep careful track of the number of currently sick patients. But the game is hard! Sometimes the levels go up, sometimes they go down. Whenever the number of sick patients drops below a previously seen level, that is good! We call that occurrence a recovery period. For example, consider the following data from two weeks of the game:



`dataArray = [0, 1, 3, 10, 23, 11, 9, 3, 42, 87, 23, 52, 82, 92, 100]`

There is one recover period from day 3 to day 6 (where it starts at 10 and ends at 9); another recover period is from day 4 to day 7 (from 23 to 3), and yet another recovery period from day 3 to day 7 (from 10 to 3). In fact, the latter is the longest recover period in this two week interval. (Notice that a recovery period can continue even as the levels drop; it does not stop as soon as it drops below its previous level.) In case you are still confused, only the start and end days matter. You can ignore what happens in between.

As the game gets harder, the length of the recovery periods seems to be increasing. Your goal in this problem is to find the longest recovery period in your data.

More specifically, you are given an array `data[1...n]` where each `data[i]` is the number of sick patients on day i of the game. We want to find a pair of indices $(low, high)$ where `data[low] > data[high]` that maximizes $(high - low)$. We will do this in three steps, below.

For the purpose of this problem, you may not use a hash table. You may only compare the values in the table.

For each of the following parts, give the most efficient algorithm you can.

A. First, give an algorithm for computing a prefix max-array M . That is, given an array of integers A , compute $M[j] = \max_{i=1}^j A[i]$. For example, if $A = [5, 2, 7, 1]$, then $M = [5, 5, 7, 7]$.

[3 marks]

B. Next, assume you are given a prefix max-array as described in the previous part. Give the pseudocode for the function `findFirst(M, key)` that finds the smallest index j in M where $M[j] > \text{key}$. [4 marks]

C. Now give an algorithm for computing the maximum length recovery period, i.e., given `data`, find `begin` and `end` such that `data[begin] > data[end]` where $(\text{end} - \text{begin})$ is maximum. You can use the algorithms from the previous parts as black-box functions, even if you were not able to write the pseudocode. [3 marks]

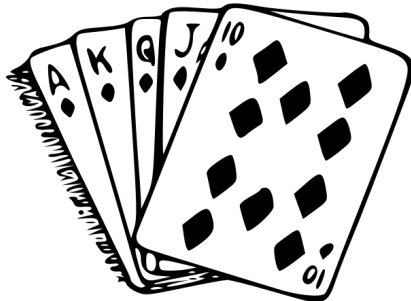
D. Explain how algorithm works (in words) and why it is correct. (You might want to give an example execution.) [3 marks]

E. What is the asymptotic worst-case running time of the complete algorithm for finding the maximum recovery period? [2 marks]

Question 8: Just for Fun [0 marks]

You are in a completely dark room. You can't see anything, not even your hands when you wave them in front of your face. You are sitting in a chair, and you can feel a table in front of you. On that table your hands run across what feels like a deck of cards. A voice overhead announces, "Before you is a standard deck of 52 cards. Thirteen of the cards are face up, the rest are face down. You will also find two boxes on the table. You must place one stack of cards in each of the two boxes (and each stack must contain at least one card). You will win if both boxes contain the same number of face up cards. Otherwise, you lose." You do not want to know what will happen if you lose.

What is your best strategy for winning? [0 mark]



— END OF PAPER —