CS1332 Campus Spring 2023 Exam 3 Version A2

Aaryan Vinay Potdar

TOTAL POINTS

92 / 100

QUESTION 1

Efficiency - Multiple Choice 14 pts

1.1 Part A 2 / 2

+ **0 pts** $$O(|V|)$$

+ **0 pts** $$O(|V||E|)$$

✓ **+ 2 pts** *Correct $$O(|V| + |E|)$$*

+ **0 pts** $$O(|V|^2)$$

+ **0 pts** $$O(|V| \log |V|)$$

+ **0 pts** blank

1.2 Part B 2 / 2

+ **0 pts** $$O(m)$$

+ **0 pts** $$O(n)$$

+ **0 pts** $$O(m + n)$$

✓ **+ 2 pts** *Correct $$O(n / m + m)$$*

+ **0 pts** $$O(mn)$$

+ **0 pts** blank

1.3 Part C 2 / 2

+ **0 pts** $$O(1)$$

+ **0 pts** $$O(\log n)$$

+ **0 pts** $$O(n)$$

+ **0 pts** $$O(n \log n)$$

✓ **+ 2 pts** *Correct $$O(n^2)$$*

1.4 Part D 2 / 2

✓ **+ 2 pts** *Correct $$O(m)$$*

+ **0 pts** $$O(n)$$

+ **0 pts** $$O(m + n)$$

+ **0 pts** $$O(n/m + m)$$

+ **0 pts** $$O(mn)$$

+ **0 pts** blank

1.5 Part E 2 / 2

+ **0 pts** $$O(1)$$

+ **0 pts** $$O(\log n)$$

✓ **+ 2 pts** *Correct $$O(n)$$*

+ **0 pts** $$O(n \log n)$$

+ **0 pts** $$O(n^2)$$

1.6 Part F 2 / 2

+ **0 pts** $$O(1)$$

+ **0 pts** $$O(\log n)$$

+ **0 pts** $$O(n)$$

+ **0 pts** $$O(n \log n)$$

✓ **+ 2 pts** *Correct $$O(n^2)$$*

1.7 Part G 2 / 2

+ **0 pts** $$O(1)$$

+ **0 pts** $$O(\log n)$$

✓ **+ 2 pts** *Correct $$O(n)$$*

+ **0 pts** $$O(n \log n)$$

+ **0 pts** $$O(n^2)$$

QUESTION 2

Code Matching - Multiple Choice 10 pts

2.1 Part A 2.5 / 2.5

✓ **+ 2.5 pts** *`j < pattern.length()`*

+ 0 pts `i < pattern.length()`

+ 0 pts `i <= pattern.length()`

+ 0 pts `j + i < pattern.length()`

2.2 Part B 2.5 / 2.5

+ 0 pts `failureTable[j] = i;`

✓ **+ 2.5 pts** *`failureTable[j] = i + 1;`*

+ 0 pts `failureTable[i] = j + 1;`

+ 0 pts `failureTable[i] = j;`

2.3 Part C 2.5 / 2.5

+ 0 pts `j - i == 1`

+ 0 pts `j - i == 0`

+ 0 pts `j == (0 | 1)`

✓ **+ 2.5 pts** *`i == 0`*

2.4 Part D 2.5 / 2.5

+ 0 pts `j = failureTable[j - 1];`

✓ **+ 2.5 pts** *`i = failureTable[i - 1];`*

+ 0 pts `j++ = failureTable[i - 1];`

+ 0 pts `i = failureTable[j - 1];`

QUESTION 3

Scenario Matching - Multiple Choice

15 pts

3.1 Part A 3 / 3

✓ **+ 3 pts** *Correct: Merge Sort*

+ 0 pts Incorrect

3.2 Part B 3 / 3

✓ **+ 3 pts** *Correct: Selection Sort*

+ 0 pts Incorrect

3.3 Part C 3 / 3

✓ **+ 3 pts** *Correct: Quicksort*

+ 0 pts Incorrect

3.4 Part D 3 / 3

✓ **+ 3 pts** *Correct: BFS*

+ 0 pts Incorrect

3.5 Part E 3 / 3

✓ **+ 3 pts** *Correct: KMP*

+ 0 pts Incorrect

QUESTION 4

4 LSD Radix Sort - Diagramming 12 / 12

✓ **+ 12 pts** *Fully correct answer*

![Screenshot_2023-04-20_at_12.48.43.png](/files/46fafddf-7800-4ea1-b10f-c71bddde63f6)

Iteration 1 buckets

+ 3 pts Completely correct

+ 1 pts **at most 2 buckets** incorrect

+ 0 pts Incorrect

Array after iteration 1

+ 3 pts Completely correct

+ 2 pts Fall through correct based on previous

Iteration 2 buckets

+ 3 pts Completely correct

+ 2 pts Fall through correct based on previous

+ 1 pts **at most 2 buckets** incorrect based

on previous

Array after iteration 2

+ 3 pts Completely correct

+ 2 pts Fall through correct based on previous

+ 0 pts Incorrect/No answer

QUESTION 5

5 Pattern Matching Tables - Diagramming 12 / 12

A. Failure Table

✓ **+ 4 pts** *Correct: 01001223*

+ 3 pts 1 wrong entry

+ 2 pts 2-3 wrong entries

+ 1 pts 4-5 wrong entries

+ 0 pts incorrect

B. Inverse Failure Table - ALLOW D in pattern :D

✓ **+ 4 pts** *Correct: ABCABCBA or similar*

+ 3 pts 1 wrong entry

+ 2 pts 2-3 wrong entries

+ 1 pts 4-5 wrong entries

+ 0 pts incorrect

C. Last Occurrence Table

✓ **+ 4 pts** *Correct:*

![Screenshot_2023-04-20_at_6.36.11_PM.png](/files/f20e831d-0729-4be1-ab6a-f08ca1611ab6)

+ 3 pts 1 incorrect

+ 2 pts 2 incorrect

+ 1 pts 3-4 incorrect

+ 0 pts incorrect

QUESTION 6

6 MST - Diagramming 12 / 12

✓ **+ 4 pts** *Completely Correct Visited Set (C, H, D, B, F, G, A, E)*

✓ **+ 8 pts** *Completely Correct MST (CH, HD, DB, CF, BG, GA, AE)*

Partial Credit

+ 1 pts 8 vertices in Visited Set

+ 1 pts Partially Correct Visited Set (At least 5 or more vertices are in their correct spots like C before H and H before D but they need to be next to each other)

+ 1 pts 7 edges in MST

+ 1 pts CH is in the MST

+ 1 pts HD is in the MST

+ 1 pts DB is in the MST

+ 1 pts CF is in the MST

+ 1 pts BG is in the MST

+ 1 pts GA is in the MST

+ 1 pts AE is in the MST

+ 0 pts No entry for Visited Set/ Incorrect entry

+ 0 pts No entry for MST edges/Incorrect entry

- 2 pts Added 9 or more edges to MST

- 1 pts Minor Error

QUESTION 7

7 AVL - Coding 6 / 15

+ 0 pts

![Screenshot_2023-04-18_at_10.24.17_PM.png](/files/12035c3d-b008-43dc-868b-26ef67e7a76c)

Correct Implementation

✓ **+ 1.5 pts** *Has a helper method*

✓ **+ 1 pts** *Calls helper method in main wrapper method*

✓ **+ 1.5 pts** *attempts to use balance factor/children height to only traverse left/right*

+ 2 pts correctly uses balance factor/children height (IF USING CHILDREN HEIGHT, MUST ACCOUNT FOR potential NullPointerExceptions)

+ 2 pts correctly handles case when BF == 0 (recurses left)

✓ **+ 1 pts** *attempts to use height for base case/checks if node is a leaf/checks if both children are null.*

+ 2 pts correctly uses height/checks if node is a leaf

+ 2 pts in case of leaf node, return curr data

✓ **+ 2 pts** *returns something*

- 1 pts Efficiency

✓ **- 1 pts** *Minor Error*

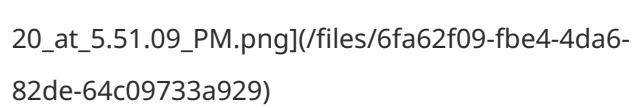
- 1 pts Syntax Error

+ 0 pts Incorrect/No answer

QUESTION 8

8 Sorting - Coding **10 / 10**

+ 0 pts Correct



Loop bounds (if there are any unexpected out of bounds exceptions inside the loops, add a minor error)

✓ **+ 3 pts** *Outer: 0 <= i < arr.length - 1*
Inner: i + 1 <= j < arr.length

+ 2 pts Outer: 0 <= i < arr.length - 1
Inner: i <= j < arr.length

+ 3 pts Outer: 0 <= i < arr.length
Inner: i + 1 <= j < arr.length

+ 2 pts Outer: 0 <= i < arr.length
Inner: i <= j < arr.length

✓ **+ 1 pts** *Inside the outer loop but before inner loop, minIndex = i*

✓ **+ 1 pts** *Attempts to compare some elements*

✓ **+ 1 pts** *Compares some elements using comparator in the inner loop*

✓ **+ 2 pts** *Set minIndex to the j if arr[j] is less than minIndex in the inner loop*

✓ **+ 1 pts** *Attempts to swap arr[minIndex] and arr[i]*

✓ **+ 1 pts** *Correctly swaps arr[minIndex] and arr[i] in the outer loop*

- 1 pts Syntax

- 1 pts Minor Error

+ 0 pts Incorrect/No Answer

QUESTION 9

9 Bonus - Picture **1 / 0**

✓ **+ 1 pts** *Writes something*

+ 0 pts Incorrect/No answer

QUESTION 10

10 Writing after time **0 / 0**

✓ **+ 0 pts** *Unmarked*

- 5 pts Marked

CS 1332 Exam 3 - Version A2
Spring Semester 2023 - April 19, 2023

Name (print clearly including your first and last name): AARYAN POTDAR

Section (8:25 am - A, 9:30 am - B, 2:00 pm D, 3:30 pm - C, Online - O): B

Signature: AP

GT account username (msmith3, etc): apotdar31

GT account number (903000000, etc): 903795148

- You must have your BuzzCard or other form of identification on the table in front of you during the exam. It is your responsibility to have your ID prior to beginning the exam.
- You are not allowed to leave the exam room and return. If you leave the room for any reason, then you must turn in your exam as complete.
- Signing and/or taking this exam signifies you are aware of and in accordance with the Academic Honor Code of Georgia Tech and the Georgia Tech Code of Conduct.
- Notes, books, calculators, phones, laptops, smart watches, headphones, etc. are not allowed. Extra paper is not allowed. If you have exhausted all space on this test, talk with your instructor. There are extra blank pages in the exam for extra space.
- If you plan on using ear plugs (foam or silicone, NOT AirPods) during the exam, you must show them to the instructor for approval.
- Pens/pencils and erasers are allowed. Do not share.
- If you brought a duck with you to the exam, you may silently consult with it at any time.
- All work entered on this exam, whether code, diagrams or multiple choice, must be implemented as was presented in lecture and recitation.
- All code must be in Java.
- Efficiency matters. For example, if you code something that uses O(n) time when there is a way to do it in O(1) time, your solution may lose credit. If your code traverses data 5 times when once would be sufficient, this also is considered poor efficiency even though both are O(n).
- Comments are not required unless a question explicitly asks for them.

Do not write below this line. It may not be scanned, therefore it will not be graded.

Table of Contents

FOR INSTRUCTOR USE ONLY - Write your name here if the student was writing after time was called.

1) Efficiency - Multiple Choice [14 points]

For each of the operations listed below, determine the time complexity of the operation as it pertains to the data structure. Select the bubble corresponding to your choice in the space provided, and completely fill in the bubble. Unless otherwise stated, assume the ***worst-case*** time complexity. However, make sure you choose the tightest Big-O upper bound possible for the operation. **Do not use an amortized analysis for these operations *unless* otherwise specified.**

A) What is the runtime of DFS on a graph with |V| vertices and |E| edges?

○ O(|V|) ○ O(|V||E|) ● O(|V| + |E|) ○ O($|V|^2$) ○ O(|V| log |V|)

B) What is the best case time complexity of finding all occurrences of a pattern in a text using Boyer Moore with Galil Rule?

○ O(m) ○ O(n) ○ O(m + n) ● O(n/m + m) ○ O(mn)

C) What is the best case runtime of selection sort on an already sorted array?

○ O(1) ○ O(log n) ○ O(n) ○ O(n log n) ● O(n^2)

D) What is the best case time complexity of finding the first occurrence of a pattern in a text using brute force?

● O(m) ○ O(n) ○ O(m + n) ○ O(n/m + m) ○ O(mn)

E) What is the runtime of generating a sorted list of data from an AVL tree? inorder traversal

○ O(1) ○ O(log n) ● O(n) ○ O(n log n) ○ O(n^2)

F) What is the worst case runtime of finding the largest element in an array using quick select?

○ O(1) ○ O(log n) ○ O(n) ○ O(n log n) ● O(n^2)

G) What is the runtime of the build heap algorithm?

○ O(1) ○ O(log n) ● O(n) ○ O(n log n) ○ O(n^2)

build + downheap calls

2) Code Matching - Multiple Choice [10 points]

Given the implementation of the KMP Build Failure Table algorithm, fill the missing parts **from the options** provided below.

```
/**
  * Builds a Failure Table for a given pattern
  *
  * @param pattern    the pattern you are building failure table for
  * @param comparator you MUST use this for checking character
  *                   equality
  * @return integer array holding the failure table
  * @throws java.lang.IllegalArgumentException if pattern or
  *                                            comparator is null
*/
public static int[] buildFailureTable(CharSequence pattern,
                                      CharacterComparator comparator) {
  if (pattern == null) {
    throw new IllegalArgumentException("Pattern cannot be null");
  } else if (comparator == null) {
    throw new IllegalArgumentException("Comparator cannot be null");
  }
  int[] failureTable = new int[pattern.length()];
  if (pattern.length() > 0) {
    int i = 0;
    int j = 1;
    failureTable[0] = 0;
    while (BLANK 1 - CHOOSE NEXT PAGE) {
      if (comparator.compare(pattern.charAt(i), pattern.charAt(j))
            == 0) {
        // BLANK 2 - CHOOSE NEXT PAGE
        i++;
        j++;
      } else {
        if (BLANK 3 - CHOOSE NEXT PAGE) {
          failureTable[j] = 0;
          j++;
        } else {
          // BLANK 4 - CHOOSE NEXT PAGE
        }
      }
    }
  }
  return failureTable;
}
```

Blank 1:

- [x] `j < pattern.length()`
- [ ] `i < pattern.length()`
- [ ] `i <= pattern.length()`
- [ ] `j + i < pattern.length()`

Blank 2:

- [ ] `failureTable[j] = i;`
- [x] `failureTable[j] = i + 1;`
- [ ] `failureTable[i] = j + 1;`
- [ ] `failureTable[i] = j;`

Blank 3:

- [ ] `j - i == 1`
- [ ] `j - i == 0`
- [ ] `j == (0 | 1)`
- [x] `i == 0`

Blank 4:

- [ ] `j = failureTable[j - 1];`
- [x] `i = failureTable[i - 1];`
- [ ] `j++ = failureTable[i - 1];`
- [ ] `i = failureTable[j - 1];`

---

3) Scenario Matching - Multiple Choice [15 points]

For the following scenarios, select the algorithm that fits best using implementations taught in class.

A) Sunidhi is developing a new social media app. In this app, posts are displayed and ordered on the user's newsfeed based on a relevance score. In order to achieve real time performance, posts should be sorted as fast as possible even in the worst case.

- ○ quicksort
- ○ selection sort
- ○ cocktail sort
- ● merge sort

B) Hannah wants to buy herself a new car. She has compiled a list of cars and wants to sort them by price. Unfortunately, her computer is old, meaning writing to memory is very expensive and she wants to minimize the number of swaps necessary.

- ● selection sort
- ○ bubble sort
- ○ insertion sort
- ○ heap sort

C) Kunal needs to sort files on his computer by name. He has a habit of naming files randomly, and his computer has limited memory available.

- ○ heap sort
- ● quicksort
- ○ merge sort
- ○ LSD radix sort

D) Mert wants to try out more restaurants in Atlanta. He wants to start with those nearby. Which graph algorithm should Mert use to find restaurants one street away from his house, then two streets away from his house, and so on?

- ● breadth first search
- ○ depth first search
- ○ Kruskal's
- ○ Prim's

E) Wendy is working on a research project to find diseased DNA sequences. DNA patterns are made up of letters A, C, G, T and include many repetitions. Wendy wants to choose an algorithm that exploits this fact.

- ○ Rabin-Karp
- ● KMP
- ○ Boyer Moore
- ○ Brute Force

4) LSD Radix Sort - Diagramming [12 points]

Perform the first **two** iterations of LSD radix sort on the provided array. You must show the contents of each bucket during each iteration. When writing the contents of a bucket, write the items in the order you add them, from **top to bottom**. Assume the algorithm accepts only non-negative numbers (and uses ten buckets).

Initial *array*:

i	0	1	2	3	4	5	6
a[i]	81	7	10	818	42	567	307

Iteration 1 *buckets*:

0	1	2	3	4	5	6	7	8	9
10	81	42					7 567 307	818	

***Array* after iteration 1:**

i	0	1	2	3	4	5	6
a[i]	10	81	42	7	567	307	818

Iteration 2 *buckets*:

0	1	2	3	4	5	6	7	8	9
7 307	10 818			42		567		81	

***Array* after iteration 2:**

i	0	1	2	3	4	5	6
a[i]	7	307	10	818	42	567	81

5) Pattern Matching Tables - Diagramming [12 points]

Fill in the missing rows for the following Failure Tables and Last Occurrence Tables.

For part B, use the following characters only: {'A', 'B', 'C'}, though these may not all be used.
There may also be multiple correct answers.
For part C, the mapping for characters not present in the pattern is filled in for you.

A. Failure Table:

index	0	1	2	3	4	5	6	7
F[index]	0	1	0	0	1	2	2	3
pattern	B	B	C	A	B	B	B	C

B. Failure Table:

index	0	1	2	3	4	5	6	7
F[index]	0	0	0	1	2	3	0	1
pattern	A	B	C	A	B	C	B	A

C. Last Occurrence Table:
Pattern: **ADAGIETTO**
0 1 2 3 4 5 6 7 8

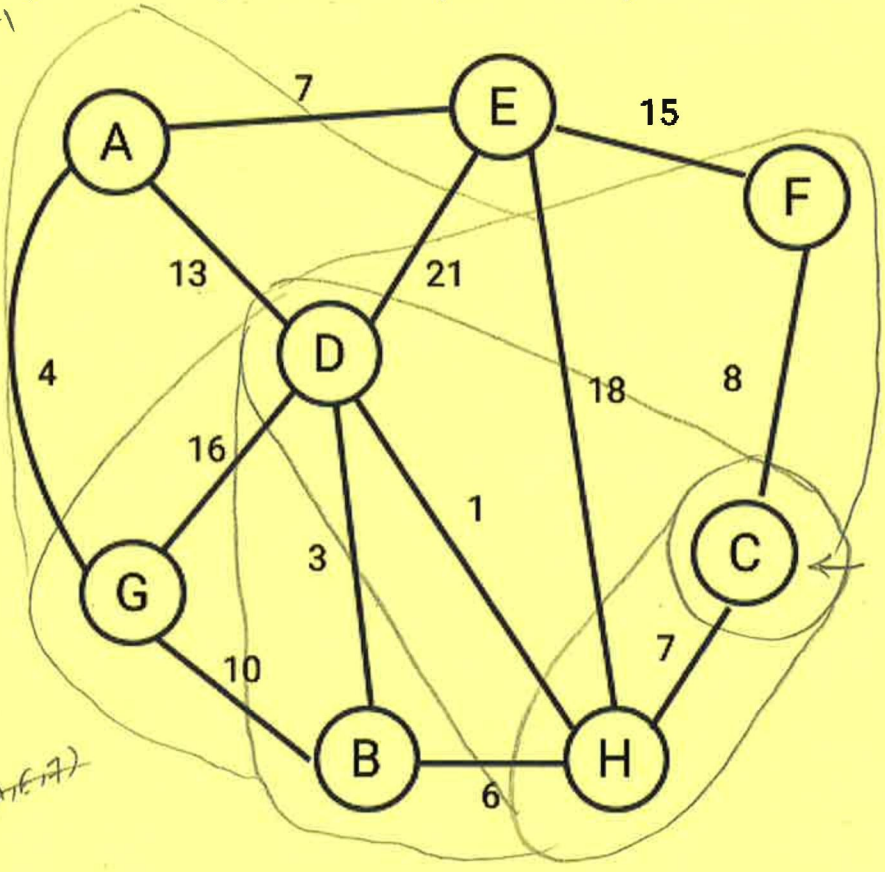
char	A	D	G	I	E	T	O	*
LOT[char]	2	1	3	4	5	7	8	-1

6) MST - Diagramming [12 points]

Given the following Graph, **perform Prim's algorithm to find the MST** of the given graph.

Answer Format: In the **answer box**, list the edges added to the MST **in the order** in which they are added in a comma separated list. For example, if you want to add the edge between D and E, write DE (or ED). Do **not** add the inverse of each edge. While performing the algorithm, list the vertices in the order they are visited when edges are pulled from the priority queue where it says **Visited List** below (don't include duplicates).

Do **not** continue the algorithm once you have a complete MST. If you need a start vertex, use vertex **C**.



VS PQ
C
H (C,H,7)
D (C,F,8)
B (H,B,6)
F (H,D,1)
G (H,E,18)
A (D,A,5)
E (D,G,16)
(D,A,13)
(D,E,21)
(B,G,10)
(F,E,15)
(A,E,7)
(G,A,4)

CH
HD
DB
CF
BG
GA
AE

Visited List: C, H, D, B, F, G, A, E

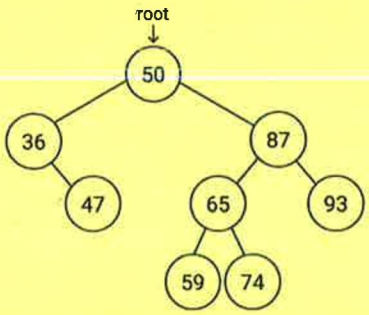
Answer Box:

CH, HD, DB, CF, BG, GA, AE

7) AVL - Coding [15 points]

Goal: Implement the following `minDeepestNode()` method, which returns the minimum data element among the deepest nodes. If there is more than one node with the same depth, return the leftmost (i.e. smallest) node with the deepest depth.

Requirements: Because **Node** is a private inner class, you must access its fields directly (i.e. use `curr.left` to retrieve the left node instead of `curr.getLeft()`). This method **must** follow a recursive implementation. This method should be implemented as efficiently as possible. You may assume **T** extends the **Comparable** interface meaning the **compareTo()** method should be used for any comparisons if necessary.



Example: on the tree above

- minDeepestNode() returns 59

```
public class AVL<T> {

  private class Node<T> { // methods & implementations omitted
    T data;
    int height;
    int bf; // balance factor
    Node<T> left;
    Node<T> right;
  }

  private Node<T> root;
}

// continued on next page
```

```
/**
* Finds the smallest deepest data of the given AVL
* You may assume the AVL is valid and not empty (i.e. root != null)
*
* @return the data of the min deepest node of the AVL tree
*/

public T minDeepestNode() {
        return getDeep(root);
}

private T getDeep(Node<T> curr) {
        if (curr == null) {
                return;
        }
        T value = curr.value;
        if (curr.left != null) {
                get deep(curr.left);
                if (curr.left.value.compareTo(value)<=0) {
                value = curr.left.value;
                }
        } else if(curr.right!=null) {
                get deep(curr.right);
                if (curr.right.value.compareTo(value)<0){
                        value = curr.right.value;
                }
        return value;
} // End of AVL class
```

8) Sorting - Coding [10 points]

Goal: Code the **selectionSort algorithm** as taught in this class. Please select the **min value** each iteration.

Requirements: Use the comparator object passed in to decide the order of merging. Assume inputs are non-null.

```
/**
* Implement selection sort.
*
* @param arr        array to sort
* @param comparator comparator object to compare elements
*/
public static <T> void selectionSort(T[] arr,
                                     Comparator<T> comparator) {
    // YOUR CODE HERE

    for (int i = 0; i < arr.length-1; i++) {
        int minIdx = i;
        for (int j = arr.length-1; j > i; j--) {
            if( comparator.compare(arr[j], arr[minIdx]) < 0) {
                minIdx = j;
            }
        }
        T temp = arr[i];
        arr[i] = arr[minIdx];
        arr[minIdx] = temp;
    }

} // END OF METHOD
```

---

9) Bonus - Picture [1 point]

For 1 extra point, please **describe** what you see below and try to come up with a **name** for it (inappropriate content will not receive credit):

A cool fish with goggles!

Name: Debbie