CS 1332 Fall 2022 Studypalooza Worksheet

The final exam is cumulative and covers everything taught this semester! This document has many, many questions but does not touch on every topic, so we would recommend reviewing any missing topics.

Document outline:

1. Big-O Review
   a. Data Structures Table
   b. Sorting Algorithms Table
   c. Pattern Matching Table
   d. Graph Algorithms Table
   e. Multiple Choice Big O

2. Multiple Choice Questions

3. Diagramming Questions
   a. Tree Diagramming (BST, AVLs, 2-4 Trees)
   b. BFS + DFS
   c. Dijkstra's Algorithm
   d. Prim's Algorithm
   e. Kruskal's Algorithm
   f. LCS

4. Coding Questions
   a. Cocktail Shaker Sort
   b. Boyer Moore LOT
   c. Rabin Karp
   d. Dijkstra's Algorithm
   e. Kruskal's Algorithm
Data Structures Big-O Table

Note: Since these data structures all do different things, some of the categories may not apply (i.e. search for Stack) - write "N/A" if it does not apply. Some of the table is already filled out. We are assuming **worst-case time complexity with amortized analysis** (denoted with an asterisk). Feel free to copy this table and fill it out for average-case analysis.

<table>
<thead>
<tr>
<th></th>
<th>add</th>
<th>remove</th>
<th>search</th>
<th>resize</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>first</td>
<td>last</td>
<td>any</td>
<td></td>
</tr>
<tr>
<td></td>
<td>index</td>
<td>index</td>
<td>index</td>
<td>index</td>
</tr>
<tr>
<td></td>
<td>at</td>
<td>index</td>
<td>for specific</td>
<td></td>
</tr>
<tr>
<td></td>
<td>value</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ArrayLists</td>
<td></td>
<td></td>
<td>O(n)</td>
<td></td>
</tr>
<tr>
<td>SLL, no tail</td>
<td></td>
<td>O(1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SLL, tail</td>
<td></td>
<td></td>
<td>O(n)</td>
<td>O(n)</td>
</tr>
<tr>
<td>CSLL</td>
<td>O(1)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DLL</td>
<td></td>
<td>O(1)</td>
<td>O(n)</td>
<td></td>
</tr>
<tr>
<td>Stack</td>
<td></td>
<td>O(1)</td>
<td></td>
<td>N/A</td>
</tr>
<tr>
<td>Queue</td>
<td></td>
<td></td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>BST</td>
<td></td>
<td>O(n)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Heap</td>
<td></td>
<td></td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>Hash Map</td>
<td></td>
<td></td>
<td>N/A</td>
<td>O(n)</td>
</tr>
<tr>
<td>AVL</td>
<td></td>
<td></td>
<td>O(log n)</td>
<td></td>
</tr>
<tr>
<td>Skip List</td>
<td></td>
<td>O(n)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2-4 Tree</td>
<td></td>
<td></td>
<td></td>
<td>O(log n)</td>
</tr>
</tbody>
</table>
### Sorting Algorithms Table

<table>
<thead>
<tr>
<th>Sorting Algorithm</th>
<th>Big-O Worst Case</th>
<th>stable?</th>
<th>in-place?</th>
<th>adaptive?</th>
</tr>
</thead>
<tbody>
<tr>
<td>bubble sort</td>
<td></td>
<td>yes</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>cocktail shaker sort</td>
<td></td>
<td></td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>selection sort</td>
<td>$O(n^2)$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>insertion sort*</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>merge sort</td>
<td>$O(n \log n)$</td>
<td></td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>quick sort</td>
<td>$O(n^2)$</td>
<td>no</td>
<td></td>
<td></td>
</tr>
<tr>
<td>radix sort</td>
<td></td>
<td>no</td>
<td>no</td>
<td></td>
</tr>
</tbody>
</table>

### Pattern Matching Table

<table>
<thead>
<tr>
<th></th>
<th>Best Case</th>
<th>Worst Case</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>single occur.</td>
<td>all occur.</td>
</tr>
<tr>
<td>Brute Force</td>
<td>O(m)</td>
<td></td>
</tr>
<tr>
<td>Boyer-Moore (no Galil rule)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>KMP</td>
<td>O(m + n)</td>
<td></td>
</tr>
<tr>
<td>Rabin-Karp</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Graph Algorithms Table

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Time (or Space) Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adjacency Matrix (space complexity)</td>
<td>( O(</td>
</tr>
<tr>
<td>Adjacency List (space complexity)</td>
<td></td>
</tr>
<tr>
<td>Edge List (space complexity)</td>
<td>( O(</td>
</tr>
<tr>
<td>Depth-First Search</td>
<td></td>
</tr>
<tr>
<td>Breadth-First Search</td>
<td></td>
</tr>
<tr>
<td>Dijkstra's Algorithm</td>
<td>( O((</td>
</tr>
<tr>
<td>Prim's MST Algorithm</td>
<td>( O((</td>
</tr>
<tr>
<td>Kruskal's MST Algorithm</td>
<td></td>
</tr>
</tbody>
</table>
Big O Practice:

1. What is the worst-case time complexity of running Boyer Moore where the text and pattern share no characters in common?
   a. $O(m)$
   b. $O(n + m)$
   c. $O(n/m + m)$
   d. $O(n*m + m)$

2. Both quick sort and insertion sort share the following properties (select all that apply):
   a. Stable
   b. Not stable
   c. In-place
   d. Out-of-place
   e. Adaptable
   f. Not adaptable

3. Both LSD Radix Sort and Merge Sort share the following properties (select all that apply):
   a. Stable
   b. Not stable
   c. In-place
   d. Out-of-place
   e. Adaptable
   f. Not adaptable

4. What is the worst case time complexity of adding to a hashmap with the following properties:
   - Uses linear probing to handle collisions
   - Has a hash function $h(key) = k \% \text{array.length}$ (updates when capacity updates)
   - Will exceed the LF upon adding the next element
   a. $O(\log n)$
   b. $O(n)$
   c. $O(n \log n)$
   d. $O(n^2)$

5. What is the average case time complexity of searching through a skiplist with a coin toss that is always HHHT?
   a. $O(1)$
   b. $O(\log n)$
c. \( O(n) \)

d. \( O(n^2) \)

6. What is the worst case time complexity of performing \( k \) left-right rotations in an AVL of size \( n \)?
   
a. \( O(1) \)
   
b. \( O(\log n) \)
   
c. \( O(\log k) \)
   
d. \( O(k) \)

7. What is the time complexity of building a heap from \( n \) items by calling the `add()` method on each item?
   
a. \( O(1) \)
   
b. \( O(\log n) \)
   
c. \( O(n) \)
   
d. \( O(n \log n) \)
   
e. \( O(n^2) \)

8. You are creating a SLL of the \( k \) largest data in a binary search tree of size \( n \) by calling the linked list's `addToBack(...)` on each data. Assume that the SLL has a tail pointer and that \( k \) can range from \( (0, n) \). What is the worst case time complexity of the above construction? There are no space complexity constraints, so consider how this construction can be done as efficiently as possible.
   
a. \( O(\log n) \)
   
b. \( O(k \log n) \)
   
c. \( O(n) \)
   
d. \( O(k * n) \)
Multiple Choice Questions
1. Given the following tree, select all options the tree could be. **You may need to select more than one answer.**

![Tree Diagram]

- Binary Tree
- Binary Search Tree
- Heap
- AVL

2. Assume you have an empty deque backed by an array with initial capacity 7. What is the resulting array after the following operations:
   - addFirst(5)
   - addLast(7)
   - addFirst(8)
   - addLast(1)

   a. 
   
   |   |   |   |   | 8 | 5 |
   |
   b. 
   
   |   |   |   |   | 1 | 7 |
   |
   c. 
   
   | 5 | 8 |   |   | 7 | 1 |
   |
   d. 
   
   | 8 | 5 | 7 | 1 |   |   |
3. Given the MinHeap below, what is the resulting array after removing 6 from the heap?

null  6  10  44  23  15  60

a. null 10  15  23  44  60

b. null  10  44  23  15  60

c. null 10  15  44  23  60

d. null 10  44  23  15  60

4. Suppose you have the HashMap below with the collision strategy of **quadratic probing**. Each <key, value> pair is <data, data>. Assume the hashcode is the same as the key (e.g. hash(10) = 10). The hashcode is then compressed to fit within the bounds of the HashMap. Suppose you add the <14, 14>. Which index should the data be added to?

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Entry</td>
<td>12</td>
<td>25</td>
<td>4</td>
<td></td>
<td>18</td>
<td></td>
<td>20</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

a. 0  
b. 2  
c. 5  
d. 6  
e. 8  

5. You are given the starting array [3, 14, 25, 26, 37, 19] and perform an unknown sorting algorithm. After 1 iteration, the array becomes [3, 14, 25, 26, 37, 19]. Which sorting algorithm could have produced the array after 1 iteration? Select all that apply.
   a. Bubble Sort
   b. Selection Sort (selecting for minimum element)
   c. Cocktail Shaker Sort
   d. QuickSort
   e. LSD Radix Sort

6. Perform 2 iterations of LSD Radix Sort on the initial array below. What is the resulting array?

   160 35 49 9 222 85 100

   a. 

   160 100 222 35 85 49 9

   b. 

   9 35 49 85 100 160 222

   c. 

   35 85 49 9 100 222 160

   d. 

   100 9 222 35 49 160 85

7. Suppose you have the text “rubixcube” and you are searching for the pattern “yolo”. Which pattern matching algorithm below would minimize the number of comparisons?
   a. Brute Force
   b. Boyer Moore
   c. KMP
   d. Rabin Karp (assume BASE is a large prime number and each character is mapped to its ascii value)
8. Given the following SkipList, select the path taken to determine whether 46 exists. Only nodes that are traversed over should be included in the path. The nodes in the path are written as (level number, data).

a. (4, -∞), (4, 5), (4, 47), (3, 47), (2, 47), (1, 47), (0, 47)
b. (4, -∞), (4, 5), (3, 5), (3, 33)
c. (4, -∞), (4, 5), (3, 5), (3, 33), (2, 33), (2, 44), (1, 44), (0, 44)
d. (4, -∞), (3, -∞), (2, -∞), (1, -∞), (0, -∞), (0, 5), (0, 15), (0, 33), (0, 44)

9. Give the following scenario, select the sorting algorithm that would perform best. Assume that the integers are being sorted in ascending order:

Efficiency is prioritized and no memory constraints exist. The relative order of duplicates must be maintained. The input data has 20 elements, with the largest element being 1234567890 and the smallest element being 0.

a. QuickSort
b. MergeSort
c. LSD Radix Sort
d. Bubble Sort
10. Given the following AVL and the add operation, describe the sequence of rotation operations that are performed after the add operation.
Add(24)

```
   30
  /   \
22     45
 / \
26   
```

a. Right-Left rotation  
b. Left-Right rotation  
c. Right rotation  
d. Left rotation  
e. No rotations needed

**Diagramming Questions:**

**Tree Diagramming:**

1. BST traversals (Challenge!)
   a. Construct a BST so that the Pre-Order traversal is [20, 16, 33, 45, 40]  
   b. Construct a BST so that the In-Order traversal is [13, 21, 27, 31, 49]  
   c. Construct a BST so that the Level-Order traversal is [6, 3, 10, 1, 5, 13]  
   d. Construct a BST so that the Post-Order traversal is [20, 18, 27, 31, 39, 22]

```
   11
  /   \
 3     14 20
 / |  \
2  5 8 12 15 17
```

Questions 2-4 apply to the 2-4 tree above. If needed for any operations, use the predecessor node. You should perform the operation based on the result of the previous part (for example, in Q3, you should remove 5 from the result in Q2)

2. Remove 11 from the following 2-4 tree  
3. Now, remove 5 and determine the resulting 2-4 tree.  
4. Now, remove 20 and determine the resulting 2-4 tree.
Questions 5-8 apply to the tree above. If necessary for any operations, use the **successor** node.

5. Remove 100 treating the tree as an AVL.

6. (Hard) Using the same tree in the image above and treating it as an AVL (without removing 100), remove 60 instead.

7. Using the same tree in the image above, remove 60 treating it as a BST instead.

8. Using the same tree in the image above, remove 29 treating it as a BST.

**For the DFS, BFS, and Dijkstra’s problems below, if there exists a tie on the next vertex to traverse, always choose the vertex that comes first alphabetically.**

9. Run BFS on the graph below beginning at vertex B and determine the order in which the vertices are visited.
10. Run DFS on the graph below beginning at vertex H and determine the order in which the vertices are visited.

11. Run Dijkstra’s on the graph below beginning at vertex G and determine
   a. The order in which the vertices are visited.
   b. The distance map to all other vertices
12. Prim's Algorithm
If you need to break ties, always add the edge alphabetically based on which vertex you have added to the visited set and the vertex you are about to add. For example, if you have already visited vertices R and S and need to break a tie between RG and SA, note that RG comes before SA alphabetically. So you would choose RG to traverse first.

Run Prim's Algorithm on the following graph and determine
a. The order in which the vertices are visited.
   b. The MST produced
Do NOT continue adding edges once an MST has been formed. If you need a start vertex, use vertex F.
13. Kruskal’s Algorithm
If you need to break ties, compare edges alphabetically after sorting the edges alphabetically. For example, if you need to break a tie between edges YA and GB, first sort the edges alphabetically to AY and BG. Note that AY comes before BG, so AY should be selected.

Run Kruskal’s Algorithm on the following graph and determine
   a. The order in which the edges are visited
   b. The final MST produced

Do NOT continue adding edges once an MST has been formed. If you need a start vertex, use vertex A.

(For more graph practice, check out the [CSVisTool](#))

14. Longest Common Subsequence
   a. List 2 LCS for the two strings below. You must produce the table and fill in the entries.
      FROZONE
      BRONZE
   b. (For Fun!) Find 4 LCS for the two strings below. You must produce the table and fill in the entries.
      GCGAGTCTAT
      CAGGATTTCAC
Coding Questions

Potential Questions:
- HashMaps Open Addressing: Linear Probing (for practice on this, see the Exam 3 Review worksheet in Canvas under Resources > PLUS Session Worksheets)
- Insertion Sort
- Cocktail Shaker Sort
- QuickSort
- HeapSort
- Boyer-Moore (without Galil-Rule)
- KMP
- Rabin-Karp
- BFS
- DFS (Recursive version)
- Dijkstra's Shortest Path
- Kruskal's MST

1. Cocktail Shaker Sort
   Goal: Write the Cocktail Shaker Sort algorithm as efficiently as possible with last swap optimization.

   Requirements: You can assume a valid non-null input array and comparator. Modify the input array and sort the elements in ascending order. See the Javadocs for more details. Some variables are initialized for you, but you must fill in the rest of the code.

   ```java
   public class Sorting {

      /**
       * Implement cocktail shaker sort (with last swap optimization).
       *
       *
       * Modify the array in place and sort the data in ascending order. You must use the comparator's .compare method to compare two data values in the array!
       *
       * @param arr the array to sort
       * @param comparator comparator the Comparator used to compare
   ```
2. Boyer Moore LOT
Goal: In the following PatternMatching class, implement the buildLOT(...) method that determines the last occurrence of each unique character in the pattern.

Requirements: Return a Map with keys as characters and values corresponding to the last index the character appears in the pattern. If the pattern is null, throw an IllegalArgumentException.

```java
public class PatternMatching {

    /**
     * Builds the LOT for the pattern.
     *
     * @param pattern a pattern you are building the LOT for
     * @return a Map with keys of all of the characters in the pattern mapping to their last occurrence in the pattern
     * @throws java.lang.IllegalArgumentException if the pattern is null
     */
    public Map<Character, Integer> buildLOT(CharSequence pattern) {
        //YOUR CODE HERE
    }
}
```
3. Rabin Karp
Goal: In the following PatternMatching class, implement the calculateInitialHash(...) method that calculates the hash of the given pattern.

Requirements: You must use the hash formula specified in the Javadocs. Use the BASE variable in your method (not the literal value). You cannot use Math.pow and your code must perform exponentiation efficiently. You can assume that pattern is not null.

```java
public class PatternMatching {
    public static final BASE = 117;

    /**
     * Computes the hash of a string using to the formula
     * from the Rabin-Karp algorithm. The formula is:
     * hash(s) = s[0]*BASE^(m-1) + s[1]*BASE^(m-2) + ... + s[m-1]*BASE^0
     * @param pattern the pattern to compute the hash of
     * @return the calculated hash value of pattern
     */
    public int calculateInitialHash(CharSequence pattern) {
        // YOUR CODE HERE.
    }
}
```
4. Dijkstra’s Algorithm

Goal: In the following Graph class, implement Dijkstra’s algorithm to find the map of the shortest paths to all vertices from the provided start vertex.

Requirements: Return a Map of <Vertex<T>, Integer> denoting the shortest distance required to reach each vertex from the provided start vertex. You can assume that the start vertex is always in the graph. Any classes in java.util.* may be used in your implementation. The Vertex, Edge, and VertexDistance classes are private inner classes, meaning they can be referenced directly.

```java
import java.util.*;

public class Graph<T> {
    private class Vertex<T> {
        T data;

        @Override
        public boolean equals(Object obj) {
            //Implementation Omitted
        }
    }

    private class Edge<T> implements Comparable<Edge<? super T>> {
        Vertex<T> u;
        Vertex<T> v;
        int weight;

        //Constructor and other methods omitted
    }

    private class VertexDistance<T> {
        Vertex<T> vertex;
        int weight;
```

```java
```
private Map<Vertex<T>, List<VertexDistance<T>>> adjList;
private Set<Vertex<T>> vertices;

/**
 * Perform Dijkstra's algorithm on the graph beginning at the start vertex
 * Return the map of the shortest distances from start to every other vertex in the graph
 * @param start the vertex to begin Dijkstra's algorithm on
 * @return the shortest distance map
 */
public Map<Vertex<T>, Integer> dijkstras(Vertex<T> start) {
    //YOUR CODE HERE
}
5. Kruskal’s Coding

Below you are given a bank of unordered code lines from A to P where some of the code lines from the code bank complete Kruskal’s algorithm as implemented in the homework.

Write the letter corresponding to each line in the order they are to be called within the method. Separate individual letters in the list using spaces. For example, your answer should be in the format of: “R S T U V W X Y”

**NOTE: You will not need to use every step. No step is used more than once.**

A few points to note:
- Your ordering should be based on the homework implementation
- Each edge stores an edge weight and a pair of vertices (u, v).
- Assume that the edge set output is initialized for both Kruskal’s.
- Assume the disjoint set has already been initialized for Kruskal’s.
- For Kruskal’s, recall that two sets are disjoint if they have no element in common. This is equivalent to checking if the representative of the sets are different (using the find() method).

A. Create a while loop that terminates when the MST has 2* (|V| - 1) or the data structure that stores edges is empty
B. For all edges adjacent to u, add to the data structure that stores edges only if the corresponding pair v has not already been visited
C. Return the edge set output if the size is 2 * (|V| - 1), else return null
D. Create a priority queue and add all edges adjacent to the start vertex to the priority queue
E. Union each vertex in the edge added to the MST
F. Traverse back to the beginning of the while loop and recheck the terminating condition
G. Create a priority queue and add all edges in the graph to the priority queue
H. Return the edge set output
I. Add edge and reverse edge to MST edge set output if the vertex u and vertex v are in disjoint sets
J. Create a while loop that terminates when the data structure that stores edges is empty
K. Remove the first edge in the data structure that stores edges
L. Add edge and reverse edge to MST edge set output if the vertex v has not been visited before (you have already visited vertex u)
M. Create a queue and add all edges in the graph to the queue
N. Add edge and reverse edge to MST edge set output if the vertex u and vertex v are not in disjoint sets
O. Initialize a visited set and add the start vertex
P. Add the edge vertex v to the visited set (you have already visited vertex u)