CS1332 Campus Spring 2023 Exam 3 Version B1

Vidit Dharmendra Pokharna

TOTAL POINTS

88.1668 / 100

QUESTION 1

Efficiency - Multiple Choice 14 pts

1.1 Part A 2 / 2

+ **0 pts** $$O(n)$$
+ **0 pts** $$O(k)$$
+ **0 pts** $$O(n \log k)$$
+ **0 pts** $$O(n \log n)$$
✓ **+ 2 pts** *Correct $$O(n^2)$$*

1.2 Part B 2 / 2

+ **0 pts** $$O(|V|)$$
+ **0 pts** $$O(\log|V|)$$
+ **0 pts** $$O(|V| + |E|)$$
+ **0 pts** $$O(|V||E|\log |E|)$$
✓ **+ 2 pts** *Correct $$O(|E|\log|E|)$$*

1.3 Part C 2 / 2

+ **0 pts** $$O(1)$$
+ **0 pts** $$O(\log n)$$
✓ **+ 2 pts** *Correct $$O(n)$$*
+ **0 pts** $$O(n \log n)$$
+ **0 pts** $$O(n^2)$$

1.4 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** $$O(mn)$$

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** $$O(n^2)$$
+ **0 pts** No answer

1.6 Part F 2 / 2

+ **0 pts** $$O(m)$$
+ **0 pts** $$O(n)$$
+ **0 pts** $$O(m + n)$$
✓ **+ 2 pts** *Correct $$O(m + n/m)$$*
+ **0 pts** $$O(mn)$$
+ **0 pts** No answer

1.7 Part G 2 / 2

+ **0 pts** $$O(1)$$
+ **0 pts** $$O(\log n)$$
+ **0 pts** $$O(n)$$
✓ **+ 2 pts** *Correct $$O(n \log n)$$*
+ **0 pts** $$O(n^2)$$

QUESTION 2

Scenario Matching - Multiple Choice

15 pts

2.1 Part A 0 / 3

+ **0 pts** Quicksort
+ **3 pts** Insertion Sort
+ **0 pts** Merge Sort
✓ **+ 0 pts** *Heap Sort*
+ **0 pts** Blank

2.2 Part B 1.5 / 3

+ **1.5 pts** LSD radix sort
+ **0 pts** Merge Sort
✓ **+ 1.5 pts** *Bubble Sort*
+ **3 pts** Selection Sort
+ **0 pts** Blank

2.3 Part C 3 / 3

+ **0 pts** Dijkstra's
✓ **+ 3 pts** *Prims*
+ **0 pts** BFS
+ **0 pts** DFS

2.4 Part D 3 / 3

✓ **+ 3 pts** *KMP*
+ **3 pts** BM
+ **0 pts** RK
+ **0 pts** BF
+ **0 pts** BLANK

2.5 Part E 3 / 3

✓ **+ 3 pts** *BFS*
+ **0 pts** DFS
+ **0 pts** Kruskal's
+ **0 pts** Prim's

QUESTION 3

3 Boyer Moore - Diagramming 12 / 12

✓ **+ 12 pts** *![Screenshot_2023-04-18_at_10.05.22_PM.png](/files/caccbcec-2846-4d97-afc3-96a2382d0cf0)*

Fully Correct

Table (only green/red characters should be circled):

+ 1 pts _1st alignment_: **FULL MATCH** (all characters circled)

+ 1 pts _2nd alignment_: Shifts pattern over by $$1$$.

+ 1 pts _2nd alignment_: Correct comparison(s) for this alignment.

+ 1 pts _3rd alignment_: Correct shift based on previous alignment.

+ 1 pts _3rd alignment_: Correct comparison(s) for this alignment.

+ 1 pts _4th alignment_: Correct shift based on previous alignment.

+ 1 pts _4th alignment_: Correct comparison(s) for this alignment.

+ 1 pts _5th alignment_: Correct shift based on previous alignment.

+ 2 pts _5th alignment_: **Full match** (all characters circled) **OR** correct comparison(s) for this alignment.

+ 0 pts Incorrect / No Answer

- 1 pts Minor error (or extra alignments)

QUESTION 4

Code Matching - Multiple Choice 10 pts

4.1 Part A 2.5 / 2.5

+ **0 pts** Incorrect: `comp.compare(arr[i - 1], arr[i]) < 0`

+ **0 pts** Incorrect: `comp.compare(arr[i - 1], arr[i]) > 0`

+ **0 pts** Incorrect: `comp.compare(arr[i], arr[i + 1]) < 0`

✓ **+ 2.5 pts** *Correct: `comp.compare(arr[i], arr[i + 1]) > 0`*

4.2 Part B **2.5 / 2.5**

+ **0 pts** Incorrect: `swapped = end;`

✓ **+ 2.5 pts** *Correct: `end = swapped;`*

+ **0 pts** Incorrect: `if (swapped == start) { return; }`

+ **0 pts** Incorrect: `start = swapped;`

4.3 Part C **2.5 / 2.5**

+ **0 pts** Incorrect: `int i = end; i >= start; i--`

+ **0 pts** Incorrect: `int i = end - 1; i >= start; i--`

+ **0 pts** Incorrect: `int i = end - 1; i > start; i--`

✓ **+ 2.5 pts** *Correct: `int i = end; i > start; i--`*

4.4 Part D **2.5 / 2.5**

✓ **+ 2.5 pts** *Correct: `swapped = i`*

+ **0 pts** Incorrect: `swapped = i - 1`

+ **0 pts** Incorrect: `start = i`

+ **0 pts** Incorrect: `start = i - 1`

QUESTION 5

5 Quicksort - Diagramming **12 / 12**

✓ **+ 12 pts** *Correct*

![Screenshot_2023-04-18_at_1.23.39_PM.png](/files/9f8ad5e6-9729-4a5c-9b67-3165e40f41dd)

Partial Credit

+ **2.5 pts** **Row 1:** Swaps pivot somewhere

+ **1 pts** **Row 1:** Swaps 73 and 68a

+ **1 pts** **Row 2:** Swaps valid pair (arr[i] > arr[j] before swap)

+ **1 pts** **Row 2:** Swaps 66 and 76

+ **1 pts** **Row 3:** Swaps valid pair

+ **1 pts** **Row 3:** Swaps 68b and 83

+ **1 pts** **Row 4:** Swaps valid pair

+ **1 pts** **Row 4:** Swaps 68b and 73

+ **2.5 pts** Pivot ends in correct final position

+ **0 pts** Incorrect/No Answer

QUESTION 6

6 Dijkstra's - Diagramming **11 / 12**

+ **8 pts** Completely Correct Visited Set: (A, F, C, G, H, B, E, D)

✓ **+ 4 pts** *Correct Shortest Path: 25*

Partial Credit

✓ **+ 2 pts** *8 vertices in Visited Set*

✓ **+ 1 pts** *A is the first vertex visited*

✓ **+ 0.5 pts** *F is after A*

✓ **+ 0.5 pts** *C is after F*

+ **0.5 pts** G is after C

✓ **+ 0.5 pts** *H is after G*

✓ **+ 0.5 pts** *B is after H*

✓ **+ 0.5 pts** *E is after B*

✓ **+ 0.5 pts** *D is after E*

✓ **+ 1 pts** *D is the last vertex visited*

+ 2.5 pts Shortest path is less than or equal to 45

+ 0 pts No entry for Visited List/Incorrect

+ 0 pts No entry for shortest path length/Incorrect

- 1 pts Minor Error

QUESTION 7

7 External Chaining HashMap - Coding

8.6668 / 10

+ 0 pts ![Screenshot_2023-04-20_at_3.21.01_PM.png](/files/cdf30a2a-6817-4649-9a0a-c6b454985299)Key

✓ **+ 0.3333 pts** *Throws an exception anywhere*

✓ **+ 0.3333 pts** *Correctly throws an IllegalArgumentException if (length < size)*

✓ **+ 1 pts** *Attempts to create a new table*

✓ **+ 0.6667 pts** *Correctly creates a new table (array of HashMapEntries) with the passed in length*

✓ **+ 0.6667 pts** *Attempts to loop through old table*

✓ **+ 0.6667 pts** *Accesses all **indices** in the table where there are non-null chains - i.e. the code sees all indices that are necessary (the code can and should still terminate when seenEntries > size)*

✓ **+ 1 pts** *Attempts to loop through each chain accessed - if the code only sees certain indices but accesses the entire **chain** on each seen index, give them this point*

+ 0.6667 pts Correctly accesses all elements in each chain seen

✓ **+ 1 pts** *Attempts to recalculate the index for every entry seen*

+ 0.6667 pts Correctly recalculates the index for each entry seen

✓ **+ 1 pts** *Attempts to add entry to new table*

✓ **+ 0.6667 pts** *Correctly adds entry to the front of **a** chain - Doesn't have to be the correct chain, just adds to front of a chain without deleting anything*

✓ **+ 0.6667 pts** *Time efficient: keeps track of number of seen elements and terminates all loops once seen > size*

✓ **+ 0.6667 pts** *Sets table to newly resized table*

- 0.6667 pts Syntax

- 0.6667 pts Minor error

+ 0 pts Incorrect / No Answer

QUESTION 8

8 Rabin-Karp - Coding 9 / 15

+ 0 pts Correct

![Screenshot_2023-04-20_at_2.05.55_AM.png](/files/6be5d9e3-f1f3-475d-a5db-3632f52d9fb0)

✓ **+ 1 pts** *checks if patternHash == textHash*

✓ **+ 1 pts** *reset j to the start of the pattern*

✓ **+ 1 pts** *attempts to compare characters in pattern to the text*

+ 2 pts correctly compares characters in pattern to the text

✓ **+ 1 pts** *attempts to check if there is a match and adds it to the list*

+ 2 pts correctly checks if there is a match and adds it to the list

+ 1 pts checks if we can shift i before updating textHash

✓ **+ 1 pts** *attempts to update textHash to roll the hash over*

✓ **+ 2 pts** *Rolling hash: correctly subtracts old character*

✓ **+ 1 pts** *Rolling hash: multiplies by BASE*

+ 1 pts Rolling hash: correctly adds new character

✓ **+ 1 pts** *increments i*

- 1 pts Efficiency

- 1 pts Syntax

- 1 pts Minor Error

+ 0 pts Incorrect/No answer

QUESTION 9

9 Bonus - Chef **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 B1
Spring Semester 2023 - April 19, 2023

Name (print clearly including your first and last name): Vidit Pokharna

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

Signature: viditdpokha

GT account username (msmith3, etc): vpokharna3

GT account number (903000000, etc): 903772087

- 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.

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

ciency - Multiple Choice [14 points]

. or 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 sorting n numbers with LSD Radix Sort where $k = n$?

○ O(n) ○ O(k) ○ O(n log k) ○ O(n log n) ● $O(n^2)$

B) What is the runtime of Dijkstra's algorithm on a graph with |V| vertices and |E| edges?

○ O(|V|) ○ O(log|V|) ○ O(|V| + |E|) ○ O((|V||E|) log|E|) ● O(|E| log |E|)

C) What is the worst case time complexity of insertion sort on an already sorted array?

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

D) What is the average case time complexity of building a failure table for KMP?

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

E) What is the time complexity of the partition procedure on an array of length n in quick select?

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

F) What is the best case time complexity of finding all occurrences of a pattern using Boyer-Moore?

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

G) What is the best case time complexity of performing merge sort?

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

2) Scenario Matching - Multiple Choice [15 points]

For the following scenarios, select the algorithm that fits the best.

A) Anh is now the intern for a very large company with a large set of data. The company has already sorted 10,000 users in ascending order based on their age. Anh has received a small batch of 150 users to sort into this database. What sorting algorithm should Anh use?

- ○ Quicksort
- ● Insertion sort
- ○ Merge sort
- ● Heap sort

B) Jack is working on sorting data in a very old system. Jack is not concerned with the amount of time it takes to sort the data. Jack has minimal space to work with and the system crashes every time too many comparisons are made. What sorting algorithm should he use?

- ○ LSD radix sort X
- ○ Merge sort X
- ● Bubble sort
- ○ Selection sort

C) Harrison was just hired by Mint Mobile. His job is to link all large cities with one high-speed wifi cable. The cable cost $10,000 per inch, so Harrison wants to use as little cable as possible. What algorithm should he use to wire all of the large cities together?

- ○ Dijkstra's
- ● Prim's
- ○ BFS
- ○ DFS

D) Jon is trying to learn all of the Roman Numerals. Roman Numerals are always composed of letters I, V, X, L, C, D, and M. Jon is trying to search for a specific Roman Numeral "XVIXVI" in a document of 100,000 Roman Numerals. What algorithm should he use?

- ● KMP FT
- ○ Boyer-Moore Last occurrence
- ○ Rabin Karp
- ○ Brute force

E) 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

...yer-Moore Diagramming [10 points]

...iven the following text, pattern, and last occurrence table, perform the **Boyer-Moore algorithm without Galil Rule** to find **all** occurrences of the pattern in the text. Starting at the first row of the table, align the pattern with the text, then **circle each character in the pattern when it is compared with the text**. Rewrite the pattern on the next row of the table each time the pattern is shifted.

Text: **ABCADABBABCA**
Pattern: **ABCA**

Last Occurrence Table:

A	B	C	*
3	1	2	-1

Row #	A	B	C	A	D	A	B	B	A	B	C	A
1	(A)	(B)	(C)	(A)								
2		A	B	C	(A)							
3						A	B	(C)	(A)			
4							A	B	C	(A)		
5									(A)	(B)	(C)	(A)
6												
7												

4) Code Matching - Multiple Choice [10 points]

Given the implementation of the Cocktail Shaker sorting algorithm, fill the missing parts **from the options** provided below.

```
/**
  * Performs the Cocktail Shaker sorting algorithm
  *
  * @param arr        the array you are sorting
  * @param comp       the comparator used to compare the data in arr
*/
public static <T> void cocktailSort(T[] arr, Comparator<T> comp) {

      int start = 0;
      int end = arr.length - 1;
      int swapped;

      while (end > start) {
          swapped = start;
          for (int i = start; i < end; i++) {
              if (BLANK 1 - CHOOSE NEXT PAGE) {
                  T temp = arr[i];
                  arr[i] = arr[i + 1];
                  arr[i + 1] = temp;
                  swapped = i;
              }
          }
          // BLANK 2 - CHOOSE NEXT PAGE
          for (BLANK 3 - CHOOSE NEXT PAGE) {
              if (comp.compare(arr[i], arr[i - 1]) < 0) {
                  T temp = arr[i];
                  arr[i] = arr[i - 1];
                  arr[i - 1] = temp;
                  // BLANK 4 - CHOOSE NEXT PAGE
              }
          }
          start = swapped;
      }
}
```

- ○ `comp.compare(arr[i - 1], arr[i]) < 0`
- ○ `comp.compare(arr[i - 1], arr[i]) > 0`
- ○ `comp.compare(arr[i], arr[i + 1] < 0`
- ● `comp.compare(arr[i], arr[i + 1]) > 0`

Blank 2:

- ○ `swapped = end;`
- ● `end = swapped;`
- ○ `if (swapped == start) { return; }`
- ○ `start = swapped;`

Blank 3:

- ○ `int i = end; i >= start; i--`
- ○ `int i = end - 1; i >= start; i--`
- ○ `int i = end - 1; i > start; i--`
- ● `int i = end; i > start; i--`

Blank 4:

- ● `swapped = i;`
- ○ `swapped = i - 1`
- ○ `start = i`
- ○ `start = i - 1`

5) Quicksort - Diagramming [12 points]

Perform **ONE** iteration of **Quicksort** on the following array, placing the pivot in its final, sorted position. The randomly selected pivot index for this array is **index 4**. Duplicate values are denoted with letters (i.e. 10a, 10b) based on their original ordering in the unsorted array. **After every swap**, rewrite the changed array on a new line, and **circle the values which have swapped** during that step (exactly two elements should be circled on each row).

There is no need to show *i* and *j* pointers; simply rewrite the array when the array changes. Do not make any recursive calls -- just perform a **single** partition step. You **may not** need to use all the rows provided.

Row #	0	1	2	3	4	5	6	7	8	9
0	68a	65	76	69	73	83	68b	85	77	66
1	73	65	76	69	68a	83	68b	85	77	66
2	73	65	66	69	68a	83	68b	85	77	76
3	73	65	66	69	68a	68b	83	85	77	76
4	68b	65	66	69	68a	73	83	85	77	76
5										
6										

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

...jkstra's - Diagramming [12 points]

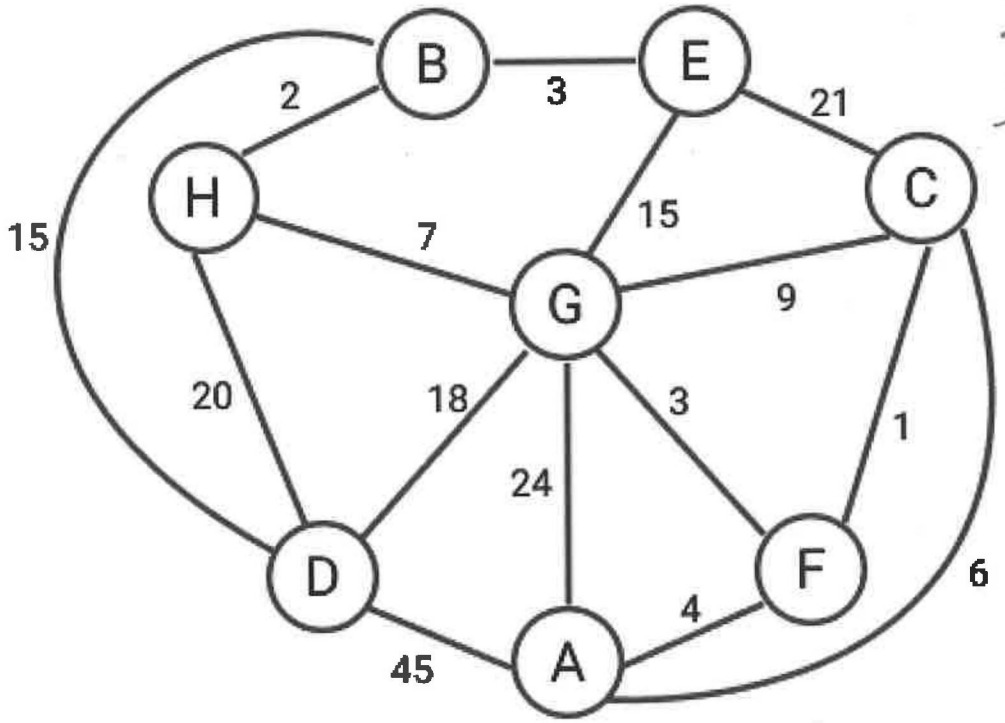
...oal: Given the following weighted graph where the vertices are lettered and the edges are weighted, **perform Dijkstra's algorithm to find the shortest path from Starting Vertex *A* to Vertex *D* in the graph** (i.e. perform Dijkstra's algorithm only until you've found the shortest path to vertex D, you do not need to perform Dijkstra's on the entire graph).

Requirements: While performing the algorithm, list the vertices as they would be pulled from the priority queue where it says **Visited List** below (don't include duplicates). Then, write the total cost of the **shortest path** from A to D where it says Shortest Path. When breaking ties between vertices, use the first vertex in alphabetical order.

Visited List:

A, F, G, C, H, B, E, D

Length of Shortest Path from A to D: 25



AD 45
~~AF 4~~
AG 24
~~AC 6~~
~~EG 3~~
~~FC 9~~
~~GD 18~~
~~GC 9~~
~~GE 15~~
~~GH 7~~
~~CE 21~~
~~HB 2~~
HD 20
~~BE 3~~

7) External Chaining Hashmap - Coding [10 points]

Goal: Write the resizeBackingTable method for a HashMap that uses external chaining. This method takes in a parameter called length and resizes the backing table to a new table of that length and correctly calculates the new indices for each entry in the table.

Requirements: Since HashMapEntry is an internal class, you should access their data directly. This method should be implemented as efficiently as possible.

```
public class HashMap<K, V> {
    private HashMapEntry<K, V>[] table;
    private int size;

    public HashMap(int capacity) {
        // HINT: this is how to properly initialize the
           backing array
        table = (HashMapEntry<K, V>[])
                new HashMapEntry[capacity];
    }

    class HashMapEntry<K, V> {
        private K key;
        private V value;
        private HashMapEntry<K, V> next;

        HashMapEntry(K key, V value,HashMapEntry<K, V> next) {
            this.key = key;
            this.value = value;
            this.next = next;
        }
    }
    /**
     * Resizes the existing backing table to the new passed in length.
     * @param length new length of the backing table
     * @throws java.lang.IllegalArgumentException if length is less
     *                                            than the number of
     *                                            items in hashmap
     */
    public void resizeBackingTable(int length) {
```

```
if( length < size){
    throw new IllegalArgumentException("Requested length is too small");
}
HashMapEntry<K,V>[] newTable = (HashMap<K,V>[]) new HashMapEntry[length];
int count = 0;
int index= 0;
while (count < size){
    HashMapEntry<K,V> temp = table[index];

    while ( temp != null){
        int hash = temp.key.getHashCode() % length;
        if( newTable[hash] == null){
            newTable[hash] = new HashMapEntry(temp.key, temp.value,
                                              null);

        } else{
            HashMapEntry<K,V> curr = new HashMapEntry(temp.key,
                                                      temp.value,
                                                      newTable[hash])

            newTable[hash] = curr;
        }
        count++;
    }
    index++;
```

```
    } // END OF METHOD
} // END OF CLASS
```

8) Rabin-Karp Coding [15 points]

Goal: Code the remaining portion of the **Rabin-Karp algorithm** as taught in this class. **You are already given the code that generates the initial hashes for the pattern and text and the outer loop of the algorithm.**

Requirements: You may **NOT** import any additional classes, create additional helper methods or instance variables. **Do not use Math.pow() or write your own equivalent.** You must use the comparator provided to compare characters. **Your code should be as efficient as possible.**

```
import java.util.ArrayList;
import java.util.List;

public class PatternMatching {
    private static final int BASE = 91;

    /**
     * You may assume that all inputs are not null and that the
     * pattern has length >= 1. You can also assume that
     * pattern.length <= text.length()
     *
     * You are already given the code that
     * generates the initial hashes for the pattern and text.
     *
     * @param pattern the pattern you are searching for
     * @param text the body of text where you are searching for the
     *             pattern
     * @param comparator the comparator you MUST use to check equality
     *             of characters
     * @return list containing the starting index for each match found
     */
    public static List<Integer> rabinKarp(CharSequence pattern,
                                          CharSequence text,
                                          CharacterComparator
                                              comparator) {
        int expBase = 1;
        int patternHash = 0;
        int textHash = 0;
        for (int i = 0; i < pattern.length(); i++) {
            patternHash += pattern.charAt(pattern.length() - i - 1)
                           * expBase;
            textHash += text.charAt(pattern.length() - i - 1)
                           * expBase;
            if (i < pattern.length() - 1) {
                expBase *= BASE;
            }
        }
```

```
List<Integer> matches = new ArrayList<>();
int i = 0;
while (i <= text.length() - pattern.length()) {
    // YOUR CODE HERE
    boolean flag = true;
    if(textHash == patternHash){
        while(flag){
            if(comparator.compare(pattern.charAt(i), text.charAt(i)) != 0){
                flag = false;
            }
            i++;
        }
        if(i == pattern.length()){
            matches.add(i-pattern.length());
        }
    } else {
        int newTextHash = (textHash - text.charAt(0)*expBase)*BASE
                          + text.charAt(i+1);
    }
}
```

linked
linkedQ

// Continue code on next page

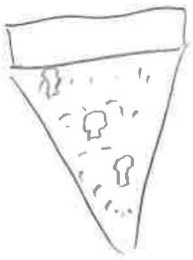
```
        }

        return matches;

    } // END OF RABIN-KARP METHOD
} // END OF PATTERN MATCHING CLASS
```

...nus - Chef [1 point]

...or 1 extra point, please **draw** or **describe** a dish you had recently that you particularly enjoyed. It can be anything you made or tried outside (inappropriate content will not receive credit):



mushroom pizza