1. **What is your name? What have you learned in this course? (1pt)**

My name is Paksh Patel. In this course I learned a lot of new things about different data structures and the way to sort, move, and grab data from within them. I also learned algorithms like bubble sort, selection sort, and binary search.

1. **Looking at the below code snippet, is this code error free and if so, what will it print to the console? If the code does have errors, then describe all syntax, run-time, and logic errors and how they may be fixed. (3pts)**

**Text

Description automatically generated**

The code is error free and will print:  
1.0

2.0

3.0

4.0

5.0

2.0

4.0

6.0

8.0

10.0

1. **Looking at the below code snippet, it is supposed to read a tab-delimited file, parse the information, and process the data. The method “process” works correctly, but there is something wrong with how the file is being read. It seems to read it word-by-word instead of line-by-line, so what could be causing this problem and how can it be fixed? (3pts)**

**Text

Description automatically generated**

The issue with the code is that the file is being read word-by-word instead of line-by-line. This is because the Scanner method next() reads the input up to the next whitespace character, which includes not only spaces but also tabs, newlines, and other whitespace characters. Therefore, when the split() method is called on fileLine, it is only splitting the first word at the first tab character it encounters, and the remaining words are not being processed correctly.

To fix this, the code should use the Scanner method nextLine() instead of next(). This method reads the input up to the next newline character, which is what is needed to process the file line-by-line.

Here's the corrected code:

try {

Scanner fileScanner = new Scanner(new File("./SomeFile.txt"));

while (fileScanner.hasNextLine()) {

String fileLine = fileScanner.nextLine();

String[] splitLine = fileLine.split("\t");

if (splitLine.length != 3) {

continue;

}

String str00 = splitLine[0];

int int01 = Integer.parseInt(splitLine[1]);

double double02 = Double.parseDouble(splitLine[2]);

process(str00, int01, double02);

}

fileScanner.close();

} catch (Exception e) {

e.printStackTrace();

}

**4.       You may assume the following Integer linked list code is provided.**

**Graphical user interface, text

Description automatically generated**

**Using the above provided code and assuming the values 1, 2, 3, 4, 5 are currently in the linked list, the following code snippet’s purpose is print out all even values. However, it never prints anything. Why is this and how can the code be modified to fix this? (3pts)**

**Text

Description automatically generated**The code never prints anything because the ListNode "temp" is not updated to the next node in the list inside the while loop. This means that if the current node's data is not even, the code will loop infinitely since the "temp" pointer never moves forward. To fix this, the code needs to update the "temp" pointer to the next node in the list by setting it equal to its link field inside the while loop. Here's the corrected code:

public void printAllEvenValues()

{

ListNode temp = head;

while(temp != null)

{

If(temp.data % 2 == 0)

{

System.out.println(temp.data);

}

temp = temp.link;

}

}

**5.       You may assume the following String linked list code is provided.**

**Graphical user interface, text

Description automatically generated**

**Using the above provided code and assuming the linked list already has the data “Abc”, “efG”, “HIJ”, “kLm”, “noP”, the following code snippet’s purpose is to print all the values in the String linked list. Does this method work as described and if so, what does it print to the console? If the method does not work as described, then detail all syntax, run-time, and logic errors and how they may be fixed. (3pts)**

**Text

Description automatically generated**

There are a few issues with the provided code.

Firstly, the print statement in the while loop is trying to print the entire node, not just the data.

Secondly, the while loop condition is checking if temp.link is not null, but it should be checking if temp is not null, since temp.link will be null for the last node in the list.

To fix these issues, the code should be modified as follows:

public void printAllValues() {

ListNode temp = head;

while(temp != null) {

System.out.println(temp.data);

temp = temp.link;

}

}

1. **Assuming the following is a Queue of Strings implemented as a circular array, show the resulting array after the following actions. The answer must clearly show the “head” and “tail” values as demonstrated below as an array, where head is the first value and tail is the last value. (3pts)**

**String Queue Circular Array**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Ref** |  | **Tail** |  |  |  |  | **Head** |  |  |
| **Index** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** |
| **Value** | **“d”** | **“e”** | **NULL** | **NULL** | **NULL** | **NULL** | **“a”** | **“b”** | **“c”** |

**Actions**

**ENQUEUE Values “j”, “k”**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Ref |  | Tail |  |  |  |  | Head |  |  |
| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| Value | “d” | “e” | “j” | “k” | NULL | NULL | “a” | “b” | “c” |

**DEQUEUE 2 Times**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Ref |  | Tail |  |  |  |  | Head |  |  |
| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| Value | NULL | NULL | “j” | “k” | NULL | NULL | “a” | “b” | “c” |

**ENQUEUE Values “m”, “n”, “o”**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Ref |  | Tail |  |  |  |  | Head |  |  |
| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| Value | “m” | “n” | “j” | “k” | “o” | NULL | “a” | “b” | “c” |

**DEQUEUE 3 Times**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Ref |  | Tail |  |  |  |  | Head |  |  |
| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| Value | NULL | NULL | NULL | NULL | NULL | NULL | “a” | “b” | “c” |

**7.       What are the Big O Time Complexities for:**

**a.       Binary Search?** Binary Search has a time complexity of O(log n) where n is the number of elements in the array. This is because binary search repeatedly divides the search interval in half, reducing the search space by half at each iteration.

**b.       Quick Sort?** Quick Sort has an average time complexity of O(n log n) and a worst-case time complexity of O(n^2), where n is the number of elements in the array. The average case occurs when the partitioning is balanced, but the worst-case occurs when the partitioning is unbalanced and the pivot chosen is either the smallest or largest element.

**c.       Heap Sort?** Heap Sort has a worst-case time complexity of O(n log n), where n is the number of elements in the array. This is because heap sort first creates a max heap in O(n) time, and then repeatedly extracts the maximum element from the heap and restores the heap property in O(log n) time. This operation is performed n times, resulting in a time complexity of O(n log n).

**All must be correct for full credit (3pts)**

1. **Using the tree below, draw the resulting binary search tree after the following operations. You may assume this is not a self-balancing tree, and the remove method must use the minimum value in the larger subtree as demonstrated in lecture. (3pts)**

**Actions**

**ADD 63**

50

/ \

25 75

/ \ / \

15 null 65 100

/ \ / \ / \

null null null null null 63

**ADD 64** 50

/ \

25 75

/ \ / \

15 null 65 100

/ \ / \ / \

null null null null 63 64

**REMOVE 25**

50

/ \

63 75

/ \

65 100

**REMOVE 50**

63

\

75

/ \

65 100

**ADD 13**

63

\

75

/ \

65 100

/ \

13 null

**A picture containing text, businesscard

Description automatically generated**

1. **Draw the tree structure of a Min Heap after the following operations. (3pts)**

**Actions**

**ADD 10, 5, 3, 12, 7, 16, 9**

**REMOVE 3 Times**

**ADD 6, 1, 8**

**REMOVE 2 Times**

Min Heap:

1

/ \

5 6

/ \ / \

10 7 16 8

\

12