CS 3305/W02: Data Structures

Assignment 7 – Sorts-Heaps-Trees Due 10/7/2022 @ 11:59 PM (100 points total *)
*PLUS 25 Bonus Points are available for Part 3 **

GENERAL SUBMISSION REQUIREMENTS

Upload all files individually as specified, not as zip files, to Assignments in D2L. Do not email files.
Make sure your program compiles, runs and produces the correct output.
Ensure you have the correct file name(s), and author header, as specified in the Assignment.
Always use meaningful labels for prompts, inputs, and outputs.
Always use comments, indentation and whitespace as shown in examples.
Note: Never hard-code test data in the test program, unless explicitly stated in the assignment.
Always allow the user to enter the test data using a menu option.

Objectives The purpose of this lab is to reinforce Sorting, Heaps and Tree concepts in Java

Assignment 07 PART 1 Sorting (50 points) - Note Part 1 is a separate deliverable:

Part 1 - The objective of Part 1 is to reinforce the understanding of Sorting data structures.

You must write your own Mergesort code using Linked Lists, not ArrayLists, and you may use Linked List class available in the Java Library. You may not use any predefined sorting methods in the Java Library.

One of the advantages of Mergesort algorithm is that it can easily be adapted to sort a linked list of values. This is because the algorithm retrieves the values from the two lists being merged in the order that they occur in the lists. If the lists are linked lists, then that algorithm can simply move down the list node after node.

Write a program that sorts a linked list of integers using the Mergesort algorithm. The program will read the integers into a linked list, and then sort the linked list using Mergesort. This will require additional linked lists, but you should use linked lists, not arrays, for all your list storage.

You may read data from the mergetest.txt file that we have provided or you may hard code that data, into one linked list. Once you have the linked list, then you begin the Mergesort algorithm.

Do not forget to include author header in each submitted file as shown, no header, no points!

```
// Name:         <your name>
// Class:        CS 3305/ put your section number after the /
// Term:         Fall 2022
// Instructor:   Sharon Perry
// Assignment:   7-Part-1-Sorting
```

Capture a **READABLE** screenshot(s) of your program output and paste into a word/pdf document. Readable means readable! Screenshots ***should not be an entire desktop*** – use some type of snipping tool. After your output screenshots, copy and paste the source code for your program into the word/pdf doc. Save doc as a file named LastName-A7-Part-1-Sorting.docx or .pdf. Last step is to upload word/pdf and .java files to D2L.

MAKE SURE YOUR CODE HAS COMMENTS ! We are getting submissions without comments in the code. No comments = (-20) points *per Part of the assignment*.

Assignment 07 – PART 2 Heaps (50 points) - Note Part 2 is a separate deliverable:

Part 2 - Objective of Part 2 is to reinforce understanding of Heap data structures.

You may use the Java Libraries for solving this problem. We recommend using java.util.* The star " * " acts as a wildcard and allows you to use the entire java.util library. For more information on this library review the tutorial https://docs.oracle.com/javase/tutorial/collections/intro/index.html

The heap presented in our textbook is also known as a max-heap, in which each node is greater than or equal to any of its children. A min-heap is a heap in which each node is less than or equal to any of its children. Min-heaps are often used to implement priority queues. Revise the Heap class in Listing 23.9 (which starts on p.878) to implement a min-heap.

Use the following logic:

1. Write a main method that will accept 5 numbers, and put them in a min-heap
2. Remove them from the min heap, printing one at a time as they are removed, to show that the list is sorted

Hint: the textbook example is for a MAX heap; you must alter that example to reflect a MIN heap – think about it !

No files or data are provided for this part of the assignment.

Do not forget to include author header in each submitted file as shown, no header, no points!

```
// Name:         <your name>
// Class:        CS 3305/ put your section number after the /
// Term:         Fall 2022
// Instructor:   Sharon Perry
// Assignment:   7-Part-2-Heaps
```

Capture a **READABLE** screenshot(s) of your program output and paste into a word/pdf document. Readable means readable! Screenshots ***should not be an entire desktop*** – use some type of snipping tool. After your output screenshots, copy and paste the source code for your program into the word/pdf doc. Save doc as a file named LastName-A7-Part-2-Heaps.docx or .pdf. Last step is to upload word/pdf and .java files to D2L.

MAKE SURE YOUR CODE HAS COMMENTS ! We are getting submissions without comments in the code. No comments = (-20) points *per Part of the assignment*

Assignment 07 – PART 3 TREES (25 Bonus points) - Note Part 3 is a separate deliverable:

Part 3 - Objective of Part 3 is to reinforce understanding of binary tree data structures.

You are going to write a program that uses a binary tree to translate morse code into characters. You may use the Java Libraries to solve this problem. We recommend using java.util.*

Morse code is a method used in telecommunication to encode text characters as standardized sequences of two different signal durations, called dots and dashes, or dits and dahs, respectively. Morse Code refers to a system for representing letters of the alphabet, numerals, and punctuation marks by an arrangement of dots, dashes, and spaces, known as the Morse Code Alphabet. A picture of the alphabet is shown below. In telecommunication a space is designated by "time units" between characters so for our purposes, we will designate a space with NULL.

Program Binary Tree

Create a binary tree to establish a morse-code table, and implement encode and decode methods. Build the binary tree, by reading from the file morse.txt.
You will encode and decode messages by searching on THAT tree. Look at the morse-code alphabet image below with yellow background, and the image of the binary tree (what your binary tree should look like when you build it). Notice how if you search LEFT on the tree it registers a dot ? Notice how h translated to morse code moves left 4 times on the tree, therefore h is represented as 4 dots **....**

Data

For your data you need letters and a Boolean (NULL), indicating that the node is empty, since not all nodes encode data. Here a sequence of dots " . " and dashes " - " are used to represent morse-code. Use spaces to separate morse-code 'letters'.

Read the morse-code data from a file called morse.txt . This file contains letters (lowercase), a space, and the morse code. *While our images all show capital letters for morse code and the binary tree, for simplicity you may assume that all letters are lower case only.*

Class and Methods

You need to define a class, MorsecodeTree. This class should contain the following methods:

String encode method - takes a string of characters and encodes it as morse-code. Letters in either upper or lowercase get translated to morse, and the dot " . " and dash " - " pass through unchanged. Any other character is not passed to the output string. Use the space to separate individual morse code letters. **Hint**: you may want to do one that encodes one character. To accomplish all of this you need to search the tree.

String decode method - takes a string of space-separated morse-code letters and produces a string with the corresponding alphabetical letters. The produced string has all lowercase letters spaces between words. You can safely assume the input string contains only dot " . " and dash " - " and spaces. If the combinations of dot " . " and dash " - " for a given character is not valid, no corresponding character should be put on the output string. **Hint**: To accomplish all of this you need to search the tree.

Program Input / Output

Your program should allow the user to select either encode or decode message option. Then, the program will ask for the appropriate message string and print the appropriate corresponding result. The program should then return to the option to select either encode or decode messages.

If user selects option to encode a message, user enters an alphabetic string and system prints the morse-code message result.

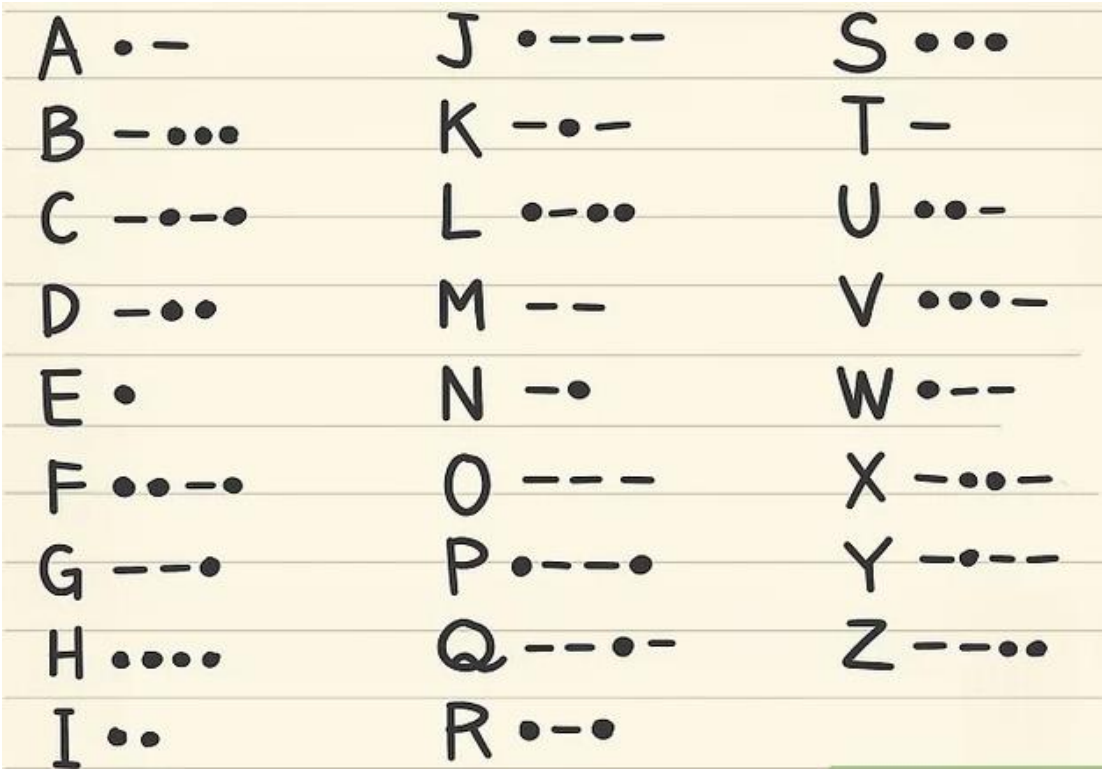
If user selects option to decode a message, user enters a morse-code message and system prints the result translated into an alphabetic message.

Getting Started

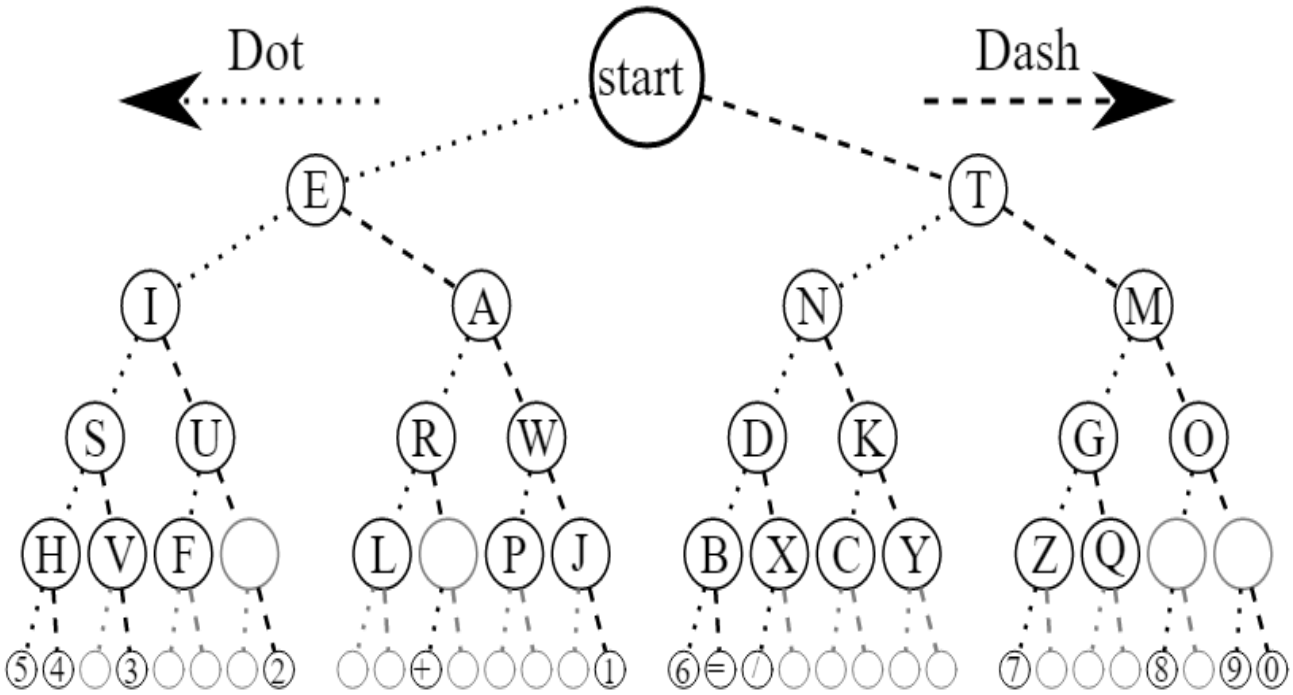
1. Review the Morse code alphabet and binary tree images below.
2. Review the attached file called morse.txt which represents the letters of the alphabet that you will use to build your binary tree. Your program should read this data from a file to build the tree.
3. Once you build your binary tree, it would have the form shown in the binary tree image below.

4. Allow users to select a message option (encode or decode) and have your program print the translated message out to the screen.

MORSE CODE ALPHABET - these images are capital letters but it doesn't matter – assume capital and lower case are the same



BINARY TREE FOR MORSE CODE



Do not forget to include author header in each submitted file as shown, no header, no points!

```
// Name:         <your name>
// Class:        CS 3305/ put your section number after the /
// Term:         Fall 2022
// Instructor:   Sharon Perry
// Assignment:   07-Part-3-Bonus-Trees
```

Capture a **READABLE** screenshot(s) of your program output and paste into a word/pdf document. Readable means readable! Screenshots ***should not be an entire desktop*** – use some type of snipping tool. After your output screenshots, copy and paste the source code for your program into the word/pdf doc. Save doc as a file named LastName-A7-Part-3-Bonus-Trees.docx or .pdf. Last step is to upload word/pdf and .java files to D2L.

MAKE SURE YOUR CODE HAS COMMENTS ! We are getting submissions without comments in the code. No comments = (-20) points *per Part of the assignment*