Metropolitan State University

ICS 240 - 01: Introduction to Data Structures

Worksheet Wb: Recursive Sorting

Total Points: 10

Out: Tue Apr 08

Due: Fri Apr 11, 11:59 PM

## What to submit?

Upload only one word or PDF document to the designated D2L folder.

# Exercise 1: Quick Sort

[4 pts] Show the array below after the first two sets of partitions for quicksort. For the first partition, use the first element of the array as the pivot. For the second partition, use the first element of each subarray as the partitions. Note: See the example in Appendix Q, with a step-by-step guide to getting the answer.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 46 | -7 | 56 | 51 | 85 | -9 | 15 | 76 | 73 | 91 | 29 | 52 | 49 | 47 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

# Exercise 2: Merge Sort

[3 pts] Perform Merge Sort on the array shown. Do this by drawing a picture analogous to that shown in Appendix M.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 10 | 35 | 29 | 43 | 47 | 77 | -6 | 33 | 92 | -1 |

# Exercise 3: More Recursion Practice

(3 pts) A Deque (pronounced “deck”) is a double-ended queue that can behave as either a Stack or a Queue. [**Here is a link to the SE 17 Deque API**](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/util/Deque.html). Write a recursive method that returns the sum of a Deque of Integers by extracting the first and last elements of the Deque, adding them together, then adding their sum to the recursively-defined sum of the rest of the Deque. The recursion bottoms out when there is zero or one element left in the Deque. The signature of the method should be

**public Integer recursiveSumDeque( Deque<Integer> d )**

For example, suppose d = [1, 4, 8, 3, 2]. Then the method calls should proceed as follows:

recursiveSumDeque( [1,4,8,3,2] ) =

3 + recursiveSumDeque( [4,8,3] ) =

10 + recursiveSumDeque( [8] ) =

18

Or suppose d = [8,7,6,5]. Then the method calls should proceed as follows:

recursiveSumDeque( [8,7,6,5] ) =

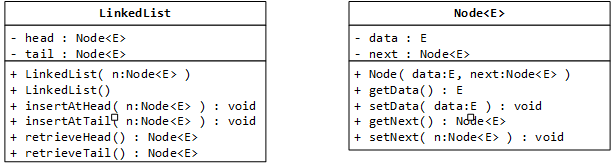
13 + recursiveSumDeque( [7,6] ) =

26 + recursiveSumDeque( null ) =

26

# Exercise 4: Linked Lists

The following UMLs represent a LinkedList class, which is a list of Node<E>’s that hold objects of generic type E, and the Node<E> class. It is a singly linked list with both head and tail nodes. Note that the insertAt… methods add a new Node<E> to the list, but that Node<E> is passed as a parameter so you dno’t need to create it. The retrieve… methods remove the head or tail from the list and return it.



Write the insertAtHead(node<E>), insertAtTail(Node<E>), retrieveHead(), and retrieveTail() methods.

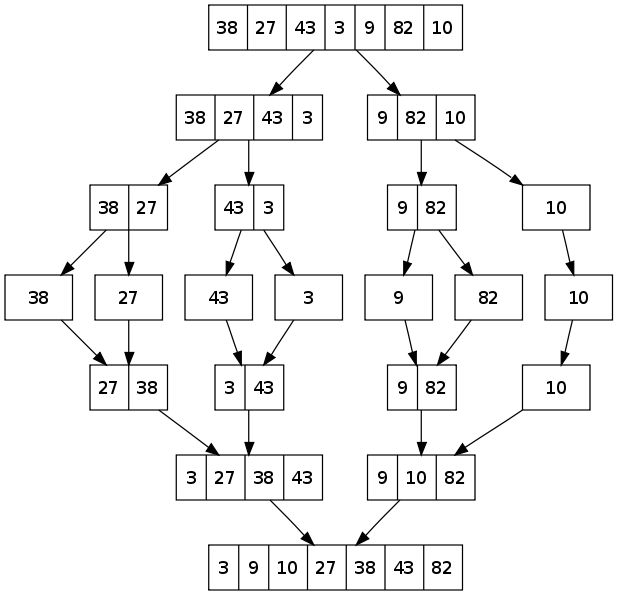
Hint: the insert methods took me 2 lines of code each, the retrieveHead() method took me 3 lines of code, and the retrieveTail() method took me 8 lines of code [that’s the only one that required iterating through the entire list].

# Exercise 5: Complete Binary Tree

Part A: Draw a ***complete binary tree*** of integers with 6 elements that is ***not*** a binary search tree.

Part B: Draw a ***binary search tree*** of integers with 6 elements that is ***not*** a complete binary tree.

## Appendix M: Merge Sort template for question 2



## Appendix Q: Quick Sort template for question 1

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 12 | 50 | 98 | 86 | 0 | 26 | 27 | 17 | 66 | 97 | 96 | 31 | 39 | 3 | 41 |
| 0 | 3 | 12 | 86 | 98 | 26 | 27 | 17 | 66 | 97 | 96 | 31 | 39 | 50 | 41 |
| 0 | 3 | 12 | 31 | 41 | 26 | 27 | 17 | 66 | 50 | 39 | 86 | 96 | 97 | 98 |

The above is what I’m looking for in an answer. Here is a step-by-step guide to how I got it.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 12 | 50 | 98 | 86 | 0 | 26 | 27 | 17 | 66 | 97 | 96 | 31 | 39 | 3 | 41 | FIRST STEP | |
| 12 | 50 | 98 | 86 | 0 | 26 | 27 | 17 | 66 | 97 | 96 | 31 | 39 | 3 | 41 | Pivot = 12 | |
| 12 | 50 | 98 | 86 | 0 | 26 | 27 | 17 | 66 | 97 | 96 | 31 | 39 | 3 | 41 | Start swapping small and large elements | |
| 12 | 3 | 98 | 86 | 0 | 26 | 27 | 17 | 66 | 97 | 96 | 31 | 39 | 50 | 41 |  | |
| 12 | 3 | 0 | 86 | 98 | 26 | 27 | 17 | 66 | 97 | 96 | 31 | 39 | 50 | 41 | Pointers cross | |
| 0 | 3 | 12 | 86 | 98 | 26 | 27 | 17 | 66 | 97 | 96 | 31 | 39 | 50 | 41 | pivot in final position | |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | SECOND STEP (left & right) | |
| 0 | 3 | 12 | 86 | 98 | 26 | 27 | 17 | 66 | 97 | 96 | 31 | 39 | 50 | 41 | Left pivot 0 | Right pivot 86 |
| 0 | 3 | 12 | 86 | 98 | 26 | 27 | 17 | 66 | 97 | 96 | 31 | 39 | 50 | 41 | Sorted, done | Start swapping elements |
| 0 | 3 | 12 | 86 | 41 | 26 | 27 | 17 | 66 | 97 | 96 | 31 | 39 | 50 | 98 | Left side done round 2 |  |
| 0 | 3 | 12 | 86 | 41 | 26 | 27 | 17 | 66 | 50 | 96 | 31 | 39 | 97 | 98 |  |  |
| 0 | 3 | 12 | 86 | 41 | 26 | 27 | 17 | 66 | 50 | 39 | 31 | 96 | 97 | 98 |  | Pointers cross |
| 0 | 3 | 12 | 31 | 41 | 26 | 27 | 17 | 66 | 50 | 39 | 86 | 96 | 97 | 98 |  |  |