Computer Science 3A

Practical Assignment 3

02 March 2023

Time: 02 March 2023 — 17h00 Marks: 100

Practical assignments must be uploaded to `eve.uj.ac.za` **before** 17h00 in the practical session.

Late submissions **will not be accepted**, and will therefore not be marked. You are **not allowed to collaborate** with any other student. You **must** upload your assignment to Eve for it to be marked. You **must** include your Javadoc for your submission. If you do not include it, **you will get a zero for your entire question**.

1. Doubly-linked lists can be used to realise a List ADT. For this practical you are required to complete the implementation of a Doubly-Linked List. This list should conform to the implementation specifications that have been provided to you. There are a number of functions that have been removed that you should complete.

   You must complete the classes & methods marked by:

   ```
   //COMPLETE CODE HERE
   ```

   Please note that you should not add any additional methods in the `DList` class or the `Test` class. However, you must complete the full implementation in the `Node` class.

   Quicksorting a Doublely-Linked List

   A quicksort on an array is an efficient method of sorting a list of items. You can perform the same type of operation on a Doublely-Linked list. In reality sorting a list in this way is not really that efficient due to the nature of a linked list.

   The basic algorthim for a quicksort is as follows:

   ```
   1. Choose a "pivot" element (this is the first element in the list)
   2. Split the list into three smaller lists
      a. One list that only contains items that are smaller than the pivot
      b. One list that only contains items that are greater than the pivot
      c. One list that only contains items that are equal to the pivot
   3. Call quicksort on the list of smaller elements
   4. Call quicksort on the list of greater elements
   5. Merge the smaller list, the equal lists, and the greater lists together.
   ```

   In order to solve this problem you need to complete the following functions:

1. `splitEqual(...)` — returns a new list of items that are equal to the passed element
2. `splitLess(...)` — returns a new list of items that are less than the passed element
3. `splitGreater(...)` — returns a new list of items that are greater to the passed element
4. `merge(...)` — merges the items in the passed sorted list such that the items in the resulting list are sorted. Merge assumes that the the calling list and the resulting list are sorted.
5. `quicksort(...)`

The above functions should return **new** lists, they should not modify the existing list. You will need to complete the implemntation of a Doublely-linked list before you attempt the implementation of a quicksort.

In order to elaborate on the operation of `merge()` consider the following example. List 1 that contains the elements `[1,3,5]` is merged with List 2 that contains the element `[7,9,10]`. The resulting List should contain `[1,3,5,7,9,10]`.

You have been provided a test class that you can use to test the execution of your solution, the results of your test program should look as follows (there are some random number in the output, so the output might look a bit different):

```
Random Numbers and Inserts
List1: <> <-> <7> <-> <21> <-> <82> <-> <33> <-> <68> <-> <>
List2: <> <-> <68> <-> <33> <-> <82> <-> <21> <-> <7> <-> <>
4th item in list 1: 33
Node: <33>
List1 without item: <> <-> <7> <-> <21> <-> <82> <-> <68> <-> <>
Removed item as expected: true

Testing AddBefore and AddAfter
List: <> <-> <54> <-> <1> <-> <6> <-> <30> <-> <>

Merge and Sorted sort test:
List1: <> <-> <6> <-> <9> <-> <>
List2: <> <-> <2> <-> <8> <-> <>
List3 (merged): <> <-> <2> <-> <6> <-> <8> <-> <9> <-> <>
SortedList: <> <-> <2> <-> <6> <-> <8> <-> <9> <-> <>

Sorting Random List
Unsorted List: <> <-> <40> <-> <15> <-> <32> <-> <0> <-> <92> <-> <52>
<-> <13> <-> <3> <-> <70> <-> <>
Sorted List: <> <-> <0> <-> <3> <-> <13> <-> <15> <-> <32> <-> <40> <->
<52> <-> <70> <-> <92> <-> <>
```

You must provide the appropriate Javadoc documentation for every function, class, and instance variable you complete. Functions, classes, and instance variables without ap-

propriate Javadoc will attract a zero. The Javadoc documentation for your assignment should be generated and uploaded as part of your zip file.

The following files must be submitted to EVE:

1. *studentnumber*_p3.zip

Marksheet

1. Node [10]
2. DList: fromArray [5]
3. DList: toArray [5]
4. DList: clone [5]
5. DList: addAfter [5]
6. DList: addBefore [5]
7. DList: remove [5]
8. DList: search [5]
9. DList: tail [5]
10. DList: toString [5]
11. DList: splitLess [5]
12. DList: splitGreater [5]
13. DList: splitEqual [5]
14. DList: merge [15]
15. DList: quicksort [5]
16. Compilation and Correct execution. [10]