UNIVERSITY OF SASKATCHEWAN

Department of Computer Science
176 Thorvaldson Building
110 Science Place, Saskatoon, SK, S7N 5C9, Canada
Telephine: (306) 966-4886, Facimile: (306) 966-4884

Assignment 5

Nodes and Node Chains

Date Due: Wednesday, June 23, 11:59pm **Total Marks: 62**

General Instructions

- **This assignment is individual work.** You may discuss questions and problems with anyone, but the work you hand in for this assignment must be your own work.
- This assignment is homework assigned to students and will be graded. This assignment shall not be distributed, in whole or in part, to any person except by the instructors of CMPT 145. Solutions will be made available to students registered in CMPT 145 after the due date. There is no educational or pedagogical reason for tutors or experts outside the CMPT 145 instructional team to provide solutions to this assignment to a student registered in the course. **Students who solicit such solutions are committing an act of Academic Misconduct, according to the University of Saskatchewan Policy on Academic Misconduct.**
- **Assignments are being checked for plagiarism.** We are using state-of-the-art software to compare every pair of student submissions. Plagiarism can include: copying answers from a web page, or from a classmate, or from solutions published in previous semesters. Basically, if you cannot delete your whole assignment and do it again yourself (given adequate time), it's not your work, so don't try to claim credit for it. Your success in this course depends on what you can do, not on what you can hand in.
- Each question indicates what to hand in. You must give your document the name we prescribe for each question, usually in the form aNqM, meaning Assignment N, Question M.
- Read the purpose of each question. Read the Evaluation section of each question.
- Make sure your name and student number appear at the top of every document you hand in. These conventions assist the markers in their work. Failure to follow these conventions will result in needless effort by the markers, and a deduction of grades for you.
- Do not submit folders, or zip files, even if you think it will help. It might help you, but it adds an extra step for the markers.
- Programs must be written in Python 3.
- **Assignments must be submitted to Moodle.** If you are not sure, talk to a Lab TA about how to do this.
- **Moodle will not let you submit work after the assignment deadline.** It is advisable to hand in each answer that you are happy with as you go. You can always revise and resubmit as many times as you like before the deadline; only your most recent submission will be graded. **Do not send late assignment submissions to your instructors, lab TAs, or markers. If you require an extension, request it in advance using email.**

Version History

- **06/21/2021**:
  - The pre-conditions in the doc-string for `merge-sort` were corrected to reflect the specification in the starter file.
- **06/21/2021**:
  - Q2: There was a reference to `to_string()` which should have said `check_chains()`.
  - Q5: The demo examples used a different version of the Node ADT, with a `create()` operation. The examples have been changed to reflect the modern version of Node ADT.
- **06/17/2021**:
  - Minor typos corrected here and there. None of these affect what you have to do.
  - A5Q2, What to hand in: Hand in answers to the questions too!
  - A5Q3, What to hand in: Test script is given, so the value of the question is reduced, and the requirement to submit testing is removed.
- **06/17/2021**: released to students

Question 1 (10 points):

Learning Objectives

Purpose: Students will practice the following skills:

- Debugging a function that works with node chains created using the Node ADT.
- Developing a practical tool for use in future questions.

Degree of Difficulty: Easy.

References: You may wish to review the following:

- Chapter 3: References
- Chapter 12: Nodes

Restrictions: This question is homework assigned to students and will be graded. This question shall not be distributed to any person except by the instructors of CMPT 145. Solutions will be made available to students registered in CMPT 145 after the due date. There is no educational or pedagogical reason for tutors or experts outside the CMPT 145 instructional team to provide solutions to this question to a student registered in the course. Students who solicit such solutions are committing an act of Academic Misconduct, according to the University of Saskatchewan Policy on Academic Misconduct.

On Moodle, you will find a *starter file* called `a5q1.py`. It has a broken implementation of the function `to_string()`, which is a function you could use in ~~used by~~ the rest of the assignment. You will also find a test script named `a5q1_testing.py`. It has a bunch of test cases pre-written for you. Read it carefully! Debug and fix the function `to_string()`. The error in the function is pretty typical of novice errors with this kind of programming task.

The interface for the function is:

```
def to_string(node_chain):
    """
    Purpose:
        Create a string representation of the node chain.  E.g.,
        [ 1 | *-]-->[ 2 | *-]-->[ 3 | / ]
    Pre-conditions:
        :param node_chain:  A node-chain, possibly empty (None)
    Post_conditions:
        None
    Return: A string representation of the nodes.
    """
```

Note carefully that the function does not do any console output. It should return a string that represents the node chain.

Here's how it might be used:

```
empty_chain = None
chain = N.node(1, N.node(2, N.node(3)))

print('empty_chain --->', to_string(empty_chain))
print('chain --------->', to_string(chain))
```

Here's what the above code is supposed to do when the function is working:

```
empty_chain ---> EMPTY
chain ---------> [ 1 | *-]-->[ 2 | *-]-->[ 3 | / ]
```

Notice that the string makes use of the characters `'[ | *-]-->'` to reflect the chain of references. The function also uses the character `'/'` to abbreviate the value `None` that indicates the end of a chain. Note especially that the empty chain is represented by the string `'EMPTY'`.

Questions

Answer the following questions about nodes, and node-chains:

1. Suppose we create a node chain as follows:

   ```
   example = N.node(1, N.node(2, N.node(3)))
   ```

   What happens when we call `print(example)`? Explain what the displayed information is telling you, in your own words.

2. Suppose you are debugging a node-chain application, and you wanted to see the contents of a node-chain named `example`, Would you prefer to use `print(example)` or `print(to_string(example))`? Why? Assume that the `to_string()` is fixed before you use it.

3. Suppose you were writing test cases for a function `to_chain()` that creates a node chain from values stored in a list. For example:

   ```
   example2 = to_chain([1,2,3])
   ```

   You don't have the code for this function, but assume it returns a node chain very similar to the one above. How could you use `to_string()` in test cases for `to_chain()`? Remember that we want the computer to do all the checking. To answer this question, give one test case for the (missing) function `to_chain()` that shows you understand how to use `to_string()` for testing.

What to Hand In

- A file named `a5q1.py` with the corrected definition of the function.
- A file named `a5q1_answers.txt` with the answers to the questions above.

Be sure to include your name, NSID, student number, and course number at the top of all documents.

Evaluation

- 4 marks: The function `to_string()` works correctly
- 6 marks: Your answers to the questions demonstrate that you understand the value of `to_string()` in debugging and testing.

Question 2 (8 points):

Learning Objectives

Purpose: Students will practice the following skills:

- Implementing a function that works with node chains created using the Node ADT.

Degree of Difficulty: Easy.

References: You may wish to review the following:

- Chapter 3: References
- Chapter 12: Nodes

Restrictions: This question is homework assigned to students and will be graded. This question shall not be distributed to any person except by the instructors of CMPT 145. Solutions will be made available to students registered in CMPT 145 after the due date. There is no educational or pedagogical reason for tutors or experts outside the CMPT 145 instructional team to provide solutions to this question to a student registered in the course. Students who solicit such solutions are committing an act of Academic Misconduct, according to the University of Saskatchewan Policy on Academic Misconduct.

Implement the function `check_chains()` . The interface for the function is:

```
def check_chains(chain1, chain2):
    """
    Purpose:
        Checks 2 node chains.
           If they are identical (the same objects),
                returns a string "same chain"
           If they are equal (same data values in the same order),
                returns a string "same values"
           Otherwise, returns a string "different starting at i"
                where i is an integer indicating the first different
                data value;
    Pre-conditions:
        :param chain1: a node-chain, possibly empty
        :param chain2: a node-chain, possibly empty
    Post-conditions:
        None
    Return:
        :return: a string
    """
```

On Moodle, you will find a *starter file* called `a5q2.py` containing the above interface documentation.

A demonstration of the application of the function is as follows:

```
chain1 = N.node(1,
         N.node(1,
         N.node(9)))
chain2 = N.node(2,
         N.node(15))
chain3 = N.node(1,
         N.node(1,
         N.node(9)))
print(check_chains(chain1, chain1))
print(check_chains(chain1, chain2))
print(check_chains(chain1, chain3))
```

The output from the demonstration is as follows:

```
same chain
different starting at 0
same values
```

The integer value given in the second example is an offset. It's zero here because the first data value is different. If two node chains are different only in the second data value, then the offset would be 1. Finally, if there are more data values that are different, then only the offset of the first difference is mentioned.

Questions

Answer the following questions about nodes, and node-chains:

1. Suppose we create a pair of node chains as follows:

   ```
   example1 = N.node(1, N.node(2, N.node(3)))
   example2 = N.node(1, N.node(2, N.node(3)))
   ```

   What happens when we call `example1 == example2`? Explain the result of this expression, in your own words.

2. Suppose you were writing test cases for a function `double_chain()` that doubles the numeric value of every node in the node chain. For example:

   ```
   example3 = double_chain(example1)
   example4 = N.node(2, N.node(4, N.node(6)))
   ```

   You don't have the code for this function, but assume it returns a node chain very similar to `example4`. How could you use `check_chains()` in test cases for `double_chain()`? To answer this question, give one test case for the (missing) function `to_chain()` that shows you understand how to use ~~`to_string()`~~ `check_chains()` for testing.

What to Hand In

- A file named `a5q2.py` with the corrected definition of the function.
- Added: A file named `a5q2_answers.txt` with the answers to the questions above.

Be sure to include your name, NSID, student number, and course number at the top of all documents.

Evaluation

- 4 marks: The function `check_chains()` works correctly
- 4 marks: Your answers to the questions demonstrate that you understand the value of `check_chains()` in testing.

Question 3 (18 points):

Note: the points for this question has been reduced, because the testing script was given. There is no longer a requirement to create a separate test script. Use the given one.

Learning Objectives

Purpose: Students will practice the following skills:

- Working with node chains created using the Node ADT.

Degree of Difficulty: Easy to Moderate.

References: You may wish to review the following:

- Chapter 3: References
- Chapter 12: Nodes

Restrictions: This question is homework assigned to students and will be graded. This question shall not be distributed to any person except by the instructors of CMPT 145. Solutions will be made available to students registered in CMPT 145 after the due date. There is no educational or pedagogical reason for tutors or experts outside the CMPT 145 instructional team to provide solutions to this question to a student registered in the course. Students who solicit such solutions are committing an act of Academic Misconduct, according to the University of Saskatchewan Policy on Academic Misconduct.

In this question you'll write three functions for node-chains that are a little more challenging. On Moodle, you can find a *starter file* called `a5q3.py`, with all the functions and doc-strings in place, and your job is to write the bodies of the functions. You will also find a test script named `a5q3_testing.py`. It has a bunch of test cases pre-written for you. Read it carefully!

(The rest of this page is blank so that each of the parts that follow can be presented on a single page.)

(a) (6 points) Implement and test the function `sumnc()` (the `nc` suggests node chain). The interface for the function is:

```
def sumnc(node_chain):
    """
    Purpose:
        Given a node chain with numeric data values, calculate
        the sum of the data values.
    Pre-conditions:
        :param node_chain: a node-chain, possibly empty, containing
                           numeric data values
    Post-condition:
            None
    Return
            :return: the sum of the data values in the node chain
    """
```

A demonstration of the application of the function is as follows:

```
empty_chain = None
chain = N.node(1, N.node(2, N.node(3)))

print('empty chain has the sum', sumnc(empty_chain))
print('chain has the sum', sumnc(chain))
```

The output from the demonstration is as follows:

```
empty chain has the sum 0
chain has the sum 6
```

(b) (6 points) Implement and test the function `count_in()`. The interface for the function is:

```
def count_in(node_chain, value):
    """
    Purpose:
        Counts the number of times a value appears in a node chain
    Pre-conditions:
        :param node_chain: a node chain, possibly empty
        :param value: a data value
    Return:
        :return: The number times the value appears in the node chain
    """
```

A demonstration of the application of the function is as follows:

```
empty_chain = None
chain = N.node(1, N.node(2, N.node(1)))

print('empty chain has', count_in(empty_chain, 1), 'occurrences of the value 1')
print('chain has', count_in(chain, 1), 'occurrences of the value 1')
```

The output from the demonstration is as follows:

```
empty chain has 0 occurrences of the value 1
chain has 2 occurrences of the value 1
```

(c) (6 points) Implement and test the function `replace_in()`. The interface for the function is:

```
def replace_in(node_chain, target, replacement):
    """
    Purpose:
        Replaces each occurrence of the target value with the replacement
    Pre-conditions:
        :param node_chain: a node-chain, possibly empty
        :param target: a value that might appear in the node chain
        :param replacement: the value to replace the target
    Pre-conditions:
        Each occurrence of the target value in the chain is replaced with
        the replacement value.
    Return:
        None
    """
```

A demonstration of the application of the function is as follows:

```
chain1 = N.node(1,
         N.node(1,
         N.node(9)))
chain2 = N.node(2,
         N.node(7,
         N.node(15)))
print('chain1 before:', to_string(chain1))
replace_in(chain1, 1, 10)
print('chain1 after:', to_string(chain1))

print('chain2 before:', to_string(chain2))
replace_in(chain2, 7, 1007)
print('chain2 after:', to_string(chain2))
```

The output from the demonstration is as follows:

```
chain1 before: [ 1 | *-]-->[ 1 | *-]-->[ 9 | / ]
chain1 after: [ 10 | *-]-->[ 10 | *-]-->[ 9 | / ]
chain2 before: [ 2 | *-]-->[ 7 | *-]-->[ 15 | / ]
chain2 after: [ 2 | *-]-->[ 1007 | *-]-->[ 15 | / ]
```

What to Hand In

- A file named `a5q3.py` with the definitions of the three functions.
- ~~A file named `a5q3_testing.py` with a test script with test cases for these three functions.~~

Be sure to include your name, NSID, student number, and course number at the top of all documents.

Evaluation

- 6 marks: Your function `sumnc()`:
  - Uses the Node ADT to sum the data values in the chain correctly.
  - Works on node-chains of any length.
  - If your function uses for-loops, Python lists, or the Python `sum()` function, you will get ZERO marks.
- 6 marks: Your function `count_in()`:
  - Uses the Node ADT to count the data values in the chain correctly.
  - Works on node-chains of any length.
  - If your function uses for-loops, or Python lists, you will get ZERO marks.
- 6 marks: Your function `replace_in()`:
  - Uses the Node ADT to replace the data values in the chain correctly.
  - Works on node-chains of any length.
  - Does not create any new nodes.
  - If your function uses for-loops, or Python lists, you will get ZERO marks.
- ~~9 marks: Your test cases for the three functions are good.~~
  - ~~Beginning with A5, test cases will be scrutinized more carefully, so make sure you think carefully about which cases you include.~~

Question 4 (12 points):

Learning Objectives

Purpose: Students will practice the following skills:

- Working with node chains created using the Node ADT.

Degree of Difficulty: Moderate.

References: You may wish to review the following:

- Chapter 3: References
- Chapter 12: Nodes

Restrictions: This question is homework assigned to students and will be graded. This question shall not be distributed to any person except by the instructors of CMPT 145. Solutions will be made available to students registered in CMPT 145 after the due date. There is no educational or pedagogical reason for tutors or experts outside the CMPT 145 instructional team to provide solutions to this question to a student registered in the course. Students who solicit such solutions are committing an act of Academic Misconduct, according to the University of Saskatchewan Policy on Academic Misconduct.

In this question you'll write ~~four~~ two functions for node-chains that are a little more challenging. On Moodle, you can find a *starter file* called `a5q4.py`, with all the functions and doc-strings in place, and your job is to write the bodies of the functions. You will also find a test script named `a5q4_testing.py`. It has a bunch of test cases pre-written for you. Read it carefully!

(a) (4 points) Implement and test the function `copync()`. The interface for the function is:

```
def double_up(node_chain):
    """
    Purpose:
        Modifies the node chain so that every node is doubled.
        E.g., given 1 -> 2 -> 3
              changed to 1 -> 1 -> 2 -> 2 -> 3 -> 3
    Pre-conditions:
        :param node_chain: a node-chain, possibly empty

    Post-conditions:
        The chain is modified to have each node repeated once.
    Return:
```

A demonstration of the application of the function is as follows:

```
chain1 = N.node(1,
         N.node(1,
         N.node(9)))

chain2 = copync(chain1)
print(check_chains(chain1, chain2))
```

The output from the demonstration is as follows:

```
same values
```

(b) (4 points) Implement and test the function `double_up()`. The interface for the function is:

```
```

A demonstration of the application of the function is as follows:

```
chain1 = N.node(1,
          N.node(2,
          N.node(9)))

before_str = to_string(chain1)
double_up(chain1)
after_str = to_string(chain1)

print('before:', before_str)
print('after:', after_str)
```

The output from the demonstration is as follows:

```
before: [ 1 | *-]-->[ 2 | *-]-->[ 9 | / ]
after: [ 1 | *-]-->[ 1 | *-]-->[ 2 | *-]-->[ 2 | *-]-->[ 9 | *-]-->[ 9 | / ]
```

What to Hand In

A file named `a5q4.py` with the definitions of the ~~three~~ two functions. Be sure to include your name, NSID, student number, and course number at the top of all documents.

Evaluation

- 4 marks: Your function `copync()`:
  - Uses the Node ADT to create a copy of the chain correctly.
  - Works on node-chains of any length.
  - If your function uses for-loops, or Python lists, or Python functions `copy()` or `deep_copy()`, you will get ZERO marks.
- 4 marks: Your function `double_up()`:
  - Uses the Node ADT to create duplicate nodes in the chain correctly.
  - Works on node-chains of any length.
  - If your function uses for-loops, Python lists, or the Python `sum()` function, you will get ZERO marks.

Question 5 (18 points):

Learning Objectives

Purpose: Students will practice the following skills:

- Working with node chains created using the Node ADT to implement slightly harder functionality.

Degree of Difficulty: Moderate to Tricky

References: You may wish to review the following:

- Chapter 3: References
- Chapter 12: Nodes

Restrictions: This question is homework assigned to students and will be graded. This question shall not be distributed to any person except by the instructors of CMPT 145. Solutions will be made available to students registered in CMPT 145 after the due date. There is no educational or pedagogical reason for tutors or experts outside the CMPT 145 instructional team to provide solutions to this question to a student registered in the course. Students who solicit such solutions are committing an act of Academic Misconduct, according to the University of Saskatchewan Policy on Academic Misconduct.

In this question you'll implement merge sort for node chains! Recall, merge sort is a divide-and-conquer technique that uses recursion to sort a given sequence (in this case a node chain). A overview of the algorithm is given below.

```
Algorithm mergeSort(NC)
    # sorts node chain NC using merge sort
    # NC - a node chain of data items
    # return : new sorted node chain of NC

    if NC contains 0 or 1 data items:
        return NC

    # divide
    NC1 = first half of NC
    NC2 = second half of NC

    # recursively sort NC1 and NC2
    NC1 = mergeSort(NC1)
    NC2 = mergeSort(NC2)

    # conquer !!!
    NC = merge(NC1, NC2)

    return NC
```

In order to implement merge sort, you are going to write three functions that will make your job easier. On Moodle, you can find a *starter file* called `a5q5.py`, with all the functions and doc-strings in place, your job is to write the bodies of the functions. You will also find a test script named `a5q5_testing.py`. It has a bunch of test cases pre-written for you. Read it carefully!

(a) (5 points) Implement the function `split_chain()`. The interface for the function is:

```
def split_chain(node_chain):
    """
    Purpose:
        Splits the given node chain in half, returning the second half.
        If the given chain has an odd length, the extra node is part of
        the second half of the chain.
    Pre-conditions:
        :param node_chain: a node-chain, possibly empty
    Post-conditions:
        the original node chain is cut in half!
    Return:
        :return: A tuple (nc1, nc2) where nc1 and nc2 are node-chains
         each containing about half of the nodes in node-chain
    """
```

This function should not create any nodes, and should return a tuple of references to the two halves of the chain. The tricky part of this is only to remember to put a `None` at the right place, to terminate the original node-chain about half way through.

A demonstration of the application of the function is as follows:

```
chain5 = N.node(3,
         N.node(1,
         N.node(4,
         N.node(1,
         N.node(5)))))
print('chain5 Before:', to_string(chain5))
a, b = split_chain(chain5)
print('chain5 After: ', to_string(a))
print('Second half:  ', to_string(b))
```

The output from the demonstration is as follows:

```
chain5 Before: [ 3 | *-]-->[ 1 | *-]-->[ 4 | *-]-->[ 1 | *-]-->[ 5 | / ]
chain5 After:  [ 3 | *-]-->[ 1 | / ]
Second half:   [ 4 | *-]-->[ 1 | *-]-->[ 5 | / ]
```

Notice how the second half of the list is a little bit larger.

(b) (5 points) Implement the function `merge()`. The interface for the function is:

```
def merge(nc1, nc2):
    """
    Purpose:
        Combine the two sorted node-chains nc1 and nc2 into a single
        sorted node-chain.
    Pre-conditions:
        :param nc1: a node-chain, possibly empty,
        containing values sorted in ascending order.
        :param nc2: a node-chain, possibly empty,
        containing values sorted in ascending order.
    Post-condition:
        None
    Return:
        :return: a sorted node chain (nc) that contains the
        values from nc1 and nc2. If both node-chains are
        empty an empty node-chain will be returned.
    """
```

This one is tricky, as there are a few special cases.
A demonstration of the application of the function is as follows:

```
chain1 = N.node(1,
         N.node(1,
         N.node(9)))
chain2 = N.node(2,
         N.node(7,
         N.node(15)))
print('chain1 before:', to_string(chain1))
print('chain2 before:', to_string(chain2))
merged_chain = merge(chain1, chain2)
print('chain1 after:', to_string(chain1))
print('chain2 after:', to_string(chain2))
print('merged_chain after:\n', to_string(merged_chain))
```

The output from the demonstration is as follows:

```
chain1 before: [ 1 | *-]-->[ 1 | *-]-->[ 9 | / ]
chain2 before: [ 2 | *-]-->[ 7 | *-]-->[ 15 | / ]
chain1 after: [ 1 | *-]-->[ 1 | *-]-->[ 9 | / ]
chain2 after: [ 2 | *-]-->[ 7 | *-]-->[ 15 | / ]
merged_chain after:
[ 1 | *-]-->[ 1 | *-]-->[ 2 | *-]-->[ 7 | *-]-->[ 9 | *-]-->[ 15 | / ]
```

Note: There are two ways to solve this. The easier way is to create a chain of new nodes, using the data values in `nc1` and `nc2`. However, creating new nodes takes a small amount of time, and memory. The more efficient way to implement this function is to re-use the given nodes, and just make the arrows point differently. Only do this if (1), you have the less efficient way working, and (2) if you want a programming challenge.

(c) (3 points) Implement the function `merge_sort()`. The interface for the function is:

```
def merge_sort(node_chain):
    """
    Purpose:
        Sorts the given node chain in ascending order using the
        merge sort algorithm.
    Pre-conditions:
        :param node_chain: a node-chain, possibly empty,
        containing only numbers
    Post-condition:
        the original node_chain may be modified and will likely
        not contain all the original elements
    Return
        :return: the node-chain sorted in ascending order.
        Ex: 45->1->21->5. Becomes 1->5->21->45
```

This one might be a little difficult, since it uses recursion.

22/06/2021 Note: The post-conditions in the above are corrected to reflect the starter file given for the assignment.

A demonstration of the application of the function is as follows:

```
chain = N.node(10,
        N.node(9,
        N.node(12,
        N.node(7,
        N.node(11,
        N.node(8))))))
print('chain before:\n', to_string(chain))
sorted_node_chain = merge_sort(chain)
print('merge_sort results:\n', to_string(sorted_node_chain))
```

The output from the demonstration is as follows:

```
chain before:
[ 10 | *-]-->[ 9 | *-]-->[ 12 | *-]-->[ 7 | *-]-->[ 11 | *-]-->[ 8 | / ]
merge_sort results:
[ 7 | *-]-->[ 8 | *-]-->[ 9 | *-]-->[ 10 | *-]-->[ 11 | *-]-->[ 12 | / ]
```

(d) (5 points) Before you submit your work, review it, and edit it for programming style. Make sure your variables are named well, and that you have appropriate (not excessive) internal documentation (do not change the doc-string, which we have given you).

What to Hand In

A file named `a5q5.py` with the definition of your functions. Be sure to include your name, NSID, student number, and course number at the top of all documents.

Evaluation

- 5 marks: Your function `split_chain()`:
  - Does not violate the Node ADT.
  - Uses the Node ADT to divide the existing node chain in two roughly equal halves.
  - Works on node-chains of any length.
- 5 marks: Your function `merge()`:
  - Does not violate the Node ADT.
  - Uses the Node ADT to create a new node-chain, when given two node chains already in ascending order. The resulting node-chain is also sorted in ascending order (node-chain only contains numbers).
  - Works on node-chains of any length.
- 3 marks: Your function `merge_sort()`:
  - Does not violate the Node ADT.
  - Uses the functions above to sort a given node-chain in ascending order.
  - Works on node-chains of any length.
- 5 marks: Overall, you used good programming style, including:
  - Good variable names
  - Appropriate internal comments (outside of the given doc-strings)