Contents

1 Basic Test Results

```
Starting tests...
Tue Dec 31 21:16:02 IST 2013
6d5c22357c1b8591b03384859a880fbaa93d08c5  -


sllist_utils.py
skipi_list.py
README

Testing README...
Done testing README...

Testing sllist_utils.py...
result_code    intersect    20    1
result_code    slgetitemg    32    1
result_code    sliceb    1203    1
result_code    cycle    22    1
result_code    mergelist    36    1
result_code    slgetitemb    25    1
result_code    palindrome    20    1
result_code    sliceg    722    1
result_code    reverse    10    1
result_code    mergesort    19    1
Done testing sllist_utils.py

Testing skipi_list.py...
result_code    skipremlast    34    1
result_code    skgetitemb    272    1
result_code    ctor    5    1
result_code    skgetitemg    740    1
result_code    removelast    336    1
result_code    addlast    370    1
result_code    skipremfirst    34    1
result_code    addfirst    370    1
result_code    removenode    370    1
result_code    removefirst    336    1
Done testing skipi_list.py

Grading summary
---------------
****** palindrome:
Number of failed tests: 0
Total number of tests : 20
Penalty: 0.0
****** cycle:
Number of failed tests: 0
Total number of tests : 22
Penalty: 0.0
****** intersect:
Number of failed tests: 0
Total number of tests : 20
Penalty: 0.0
****** reverse:
Number of failed tests: 0
Total number of tests : 10
Penalty: 0.0
****** mergelist:
Number of failed tests: 0
Total number of tests : 36
```

```
Penalty: 0.0
****** mergesort:
Number of failed tests: 0
Total number of tests : 19
Penalty: 0.0
****** slgetitemg:
Number of failed tests: 0
Total number of tests : 32
Penalty: 0.0
****** slgetitemb:
Number of failed tests: 0
Total number of tests : 25
Penalty: 0.0
****** sliceb:
Number of failed tests: 0
Total number of tests : 1203
Penalty: 0.0
****** ctor:
Number of failed tests: 0
Total number of tests : 5
Penalty: 0.0
****** addfirst:
Number of failed tests: 0
Total number of tests : 370
Penalty: 0.0
****** addlast:
Number of failed tests: 0
Total number of tests : 370
Penalty: 0.0
****** removefirst:
Number of failed tests: 0
Total number of tests : 336
Penalty: 0.0
****** removelast:
Number of failed tests: 0
Total number of tests : 336
Penalty: 0.0
****** removenode:
Number of failed tests: 0
Total number of tests : 370
Penalty: 0.0
****** skgetitemg:
Number of failed tests: 0
Total number of tests : 740
Penalty: 0.0
****** skgetitemb:
Number of failed tests: 0
Total number of tests : 272
Penalty: 0.0
****** skipremfirst:
Number of failed tests: 0
Total number of tests : 34
Penalty: 0.0
****** skipremlast:
Number of failed tests: 0
Total number of tests : 34
Penalty: 0.0
****** sliceg:
Number of passed tests: 722
Total number of tests : 722
Bonus: 10.0
******
Expected automatic grade: 110.0
******
Submission passed!
Tests completed
```

2 aaa expected autograde

```
Grading summary
---------------
****** palindrome:
Number of failed tests: 0
Total number of tests : 20
Penalty: 0.0
****** cycle:
Number of failed tests: 0
Total number of tests : 22
Penalty: 0.0
****** intersect:
Number of failed tests: 0
Total number of tests : 20
Penalty: 0.0
****** reverse:
Number of failed tests: 0
Total number of tests : 10
Penalty: 0.0
****** mergelist:
Number of failed tests: 0
Total number of tests : 36
Penalty: 0.0
****** mergesort:
Number of failed tests: 0
Total number of tests : 19
Penalty: 0.0
****** slgetitemg:
Number of failed tests: 0
Total number of tests : 32
Penalty: 0.0
****** slgetitemb:
Number of failed tests: 0
Total number of tests : 25
Penalty: 0.0
****** sliceb:
Number of failed tests: 0
Total number of tests : 1203
Penalty: 0.0
****** ctor:
Number of failed tests: 0
Total number of tests : 5
Penalty: 0.0
****** addfirst:
Number of failed tests: 0
Total number of tests : 370
Penalty: 0.0
****** addlast:
Number of failed tests: 0
Total number of tests : 370
Penalty: 0.0
****** removefirst:
Number of failed tests: 0
Total number of tests : 336
Penalty: 0.0
****** removelast:
Number of failed tests: 0
Total number of tests : 336
Penalty: 0.0
****** removenode:
```

```
Number of failed tests: 0
Total number of tests : 370
Penalty: 0.0
****** skgetitemg:
Number of failed tests: 0
Total number of tests : 740
Penalty: 0.0
****** skgetitemb:
Number of failed tests: 0
Total number of tests : 272
Penalty: 0.0
****** skipremfirst:
Number of failed tests: 0
Total number of tests : 34
Penalty: 0.0
****** skipremlast:
Number of failed tests: 0
Total number of tests : 34
Penalty: 0.0
****** sliceg:
Number of passed tests: 722
Total number of tests : 722
Bonus: 10.0
******
Expected automatic grade: 110.0
******
Submission passed!
```

3 aaa hint result.png

4 README

```
roigreenberg
305571234
Roi Greenberg

I discussed the exercise with: Roi Greenberg, Naama Antebi.


==================
= README for ex8 =
==================

==================
=  Description:  =
==================


Part 1: Hash functions:

Q1:
===
h0-    The function maps everything to zero
h1-    The function transform the first letter of the recivied key into a
        number by using 'ord' function then divide it (module) by m
h2-    The function use 'ord' in order to sum the word letters numeric
        value and divide (module) the sum by m
h3-    The function use 'ord' in order to sum the word letters numeric
        value, in each iteration the function multiplies the current
        sum by 128 and divide (module) the final sum by m
h4-    The function take a random number in the hashtable size range
h5-    There are 3 situations in the function in regard to the key:
         1. string - use "str_to_int" that act on the key's binary code
            in order to recieve an integer.
         2. key is a type of collections.Hashable - return the key's
        "__str__" value
         3. integer - divide (module) the key by m
h6-    The function use the key's length. if it 0, the function return zero
        if it not 0, it use 'ord' in order to get the numeric value of the
        letter, then change the binary value and divide (module) by m
h7-    The function using the cryptographic hash function "md5"
h8-    The function using the cryptographic hash function "sha1"
h9-    The function using the python built-in hash function

Q2:
===
h9 is the best function. as it seen from the last question it have no drawback.
also for string type of data, h5 works good as well

Q3:
===
in case M greater then N the collisions number reduces
in case N greater then M it increase (exeppt in h0 which had no difference)

Q4:
===
by using prime number, the collisions will reduce because the division remainder
will spread more

Q5:
===
```

```
h0-     the function is not effective at all because it maps everything to 0
h1-    advantage- all values orderd in groups by their first latter
    drawbacks- the collisions are big, not work on numbers
h2-    advantage- less collision then h1 function
    drawbacks- same-letters words will get the same value also not work
                  on numbers
h3-    advantage- less collision the h2 function because of noticing to
           the order of the latter
    drawbacks- not work on numbers
h4-    advantage- in the best situation the function will map the keys equelly
    drawbacks- in the worst situation the function will map the keys to the
               same value
h5-    advantage- the function is efficient because it divides the cases by
           the inputs types
    drawbacks- not working well with numbers
h6-    advantage- the function is efficient because it change the binary value
           will decrease the collision
    drawbacks- if many key's with a length of 0 the collision will increase
h7-    advantage- the collision is very small
    drawbacks- take a long time to get the hash value
h8-    advantage- the collision is very small
    drawbacks- take a long time to get the hash value
h9-    advantage- the function have only advantage, it run fast, take care of
                  any type of data and have a big range of hash value


Part 2: sllist_utils.py

Description: implement some function for a linked lists
1. reverse - reverse the list
    complexity - O(n) - the function run on each node one time
2. merge_lists - merge to lists into one
    complexity - O(n) - the function run on each node from both list one time
3. contains_cycle - check if list have a cycle
    complexity - O(n) - the function run in a loop and do one action each time
4. get_item - return the data of the k'th element
    complexity - O(n) -  the function use list_len which is O(n)
5. is_palindrome - check if list s palindrome
    complexity - O(n) - the function run on each node constant number of times
6. have_intersection - check if 2 lists connect in same point
    complexity - O(n) -  the function use reverse() which is O(n)
7. slice - slice the list according to the given arguments
    complexity  - O(n) - the function run on each node one time
8. merge_sort - sort a list in merge_sort algorithem
    complexity - O(n*log(n)) - the recursion split log(n) times and for each time
             sort the list by O(n) actions
9. list_len(help function) - find the length of the list
    complexity - O(n) - the function run on each node one time

Part 3: skipi_list.py

Description: implement a class for skipilist
the class has the following functions:
0. __init__ - Constructs an empty SkipiList
1. add_first - add new node to the begining of the list
2. remove_first - remove the first node
3. add_last - add new node to the end of the list
4. remove_last - remove the last node
5. remove_node - remove the given node
6. getitem - return the data of the k'th element


by using skipi list in part 2 the the complexity will be:

1. reverse -           O(n) - the function still need to run on each node one time
2. merge_lists -       O(n) - the function still need to run on each node from
                              both list one time
3. contains_cycle - in skipi list because of the tail it not supposed be cycle
```

-1 you should mention that the method h4 cant be used as a hash function, since the same value can get sent to different cells of the table.

get the k'th item is O(k), if k<0 then it O(n).

line 128: its the right answer.

```
                (anther option - O(1) to check if tail.next is not None)
4. get_item -            O(n) - in worse case function need to run over all the list
5. is_palindrome -     O(n) - the function still need to run on each node constant number of times
6. have_intersection - O(1) - the function only need to compare the tails
7. slice -             O(n) - the function still need to run on each node one time
8. merge_sort -        O(n*log(n)) - the recursion still need to split log(n) times
                                     and for each time sort the list by O(n) actions


============================
=  List of submitted files: =
============================

README
sllist_utils.py
skipi_list.py
```

5 skipi list.py

```
1  ##################################################################
2  # FILE: skipi_list.py
3  # WRITER: Roi Greenberg + roigreenberg + 305571234
4  # EXERCISE : intro2cs ex8 2013-2014
5  # Description: implement a class for skipilist
6  # the class has the following functions:
7  # 0. __init__ - Constructs an empty SkipiList
8  # 1. add_first - add new node to the begining of the list
9  # 2. remove_first - remove the first node
10 # 3. add_last - add new node to the end of the list
11 # 4. remove_last - remove the last node
12 # 5. remove_node - remove the given node
13 # 6. getitem - return the data of the k'th element
14 ##################################################################
15
16 from sllist import SkipiNode as Node
17
18
19 class SkipiList:
20
21     """
22     This class represents a special kind of a doubly-linked list
23     called a SkipiList. A SkipiList is composed of Nodes (SkipiNode from
24     sllist).cIn addition to the data, each Node has one pointer to the
25     next Node in the list, and another pointer to the prev-prev Node in the
26     list (hence the name "skipi"). The only data members the class contains
27     are the head and the tail of the list.
28     """
29
30     def __init__(self):
31         """Constructs an empty SkipiList."""
32         self.head = None
33         self.tail = None
34
35     def add_first(self, data):
36         """
37         Adds an item to the beginning of a list.
38         data - the item to add
39         """
40         # define the head as the new Node
41         self.head = Node(data, next=self.head)
42         # if list was empty define th tail as the head
43         if self.tail is None:
44             self.tail = self.head
45         # set the skip back pointer if needed
46         if self.head.next is not None:
47             if self.head.next.next is not None:
48                 self.head.next.next.skip_back = self.head
49
50     def remove_first(self):
51         """
52         Removes the first Node from the list and return its data.
53         Returns that data of the removed node
54         """
55         # return None if there are no Nodes
56         if self.head is None:
57             return None
58         # save and disconect the first Node from the list
59         # and set the head to the next Node
```

```
60          removed = self.head
61          self.head = self.head.next
62          removed.next = None
63          # set the tail as None if list got empty
64          if self.head is None:
65              self.tail = None
66          # remove the skip back pointer from the second Node if needed
67          elif self.head.next is not None:
68              self.head.next.skip_back = None
69
70          return removed.data
71
72      def add_last(self, data):
73          """
74          Adds an item to the end of a list.
75          data - the item to add
76          """
77          # if list empty set head and tail as the new Node
78          if self.head is None:
79              self.tail = Node(data, next=None)
80              self.head = self.tail
81          # else set new tail
82          else:
83              self.tail.next = Node(data, next=None)
84              # set the skip back pointer if needed
85              if self.head != self.tail:
86                  if self.tail.skip_back is None:
87                      self.tail.next.skip_back = self.head
88                  else:
89                      self.tail.next.skip_back = self.tail.skip_back.next
90              # set the tail to the new one
91              self.tail = self.tail.next
92
93      def remove_last(self):
94          """
95          Removes the last Node from the list and return its data.
96          The data of the removed node
97          """
98          # return None if no Node to remove
99          if self.tail is None:
100             return None
101         # save the tail
102         removed = self.tail
103         # set the new tail
104         if self.tail != self.head:
105             if self.tail.skip_back is None:
106                 self.tail = self.head
107             else:
108                 self.tail = self.tail.skip_back.next
109             self.tail.next = None
110         else:
111             self.tail = self.head = None
112
113         return removed.data
114
115     def remove_node(self, node):
116         """
117         Removes a given Node from the list, and returns its data.
118         Assumes the given node is in the list. Runs in O(1).
119         """
120         # remove the first Node
121         if node == self.head:
122             return self.remove_first()
123         # remove the last Node
124         elif node == self.tail:
125
126             return self.remove_last()
127         # set the skip back pointers after removing the Node and set the
```

```
        # preview Node to point on the next one(skip the removing node)
        if node.next != self.tail:
            if node.skip_back is not None:
                node.next.next.skip_back = node.skip_back.next
            else:
                node.next.next.skip_back = self.head
        if node.skip_back is not None:
            node.next.skip_back = node.skip_back
            node.skip_back.next.next = node.next
        else:
            self.head.next = node.next
            node.next.skip_back = None
        # disconnect the Node from the list
        node.next = None
        return node.data

    def __getitem__(self, k):
        """
        Returns the data of the k'th item of the list.
        If k is negative return the data of k'th item from the end of the list.
        If abs(k) > length of list raise IndexError.
        """
        # set a pinter
        # in case k non negetive
        if k >= 0:
            # set a pinter
            node = self.head
            # move the pointer to the right position
            for i in range(k):
                if node != self.tail:
                    node = node.next
                # raise an error if k > length of list
                else:
                    raise IndexError
            return node.data
        # in case k negetive
        else:
            # set a pinter
            node = self.tail
            # move the pointer to the right position
            for i in range(-k // 2):
                if node is not None and node != self.head:
                    node = node.skip_back
                # raise an error if abs(k) > length of list
                else:
                    raise IndexError
            # return the head if one if k is odd amd pointer move before head
            if node is None:
                if k % 2 == 0:
                    return self.head.data
                # raise an error if abs(k) > length of list
                else:
                    raise IndexError
            # move the pointer to the next one if k is odd
            if k % 2 == 0:
                node = node.next
            return node.data
```

6 sllist utils.py

```
1  ######################################################################
2  # FILE: sllist_utils.py
3  # WRITER: Roi Greenberg + roigreenberg + 305571234
4  # EXERCISE : intro2cs ex8 2013-2014
5  # Description: implement some function for a linked lists
6  # 1. reverse - reverse the list
7  # 2. merge_lists - merge to lists into one
8  # 3. contains_cycle - check if list have a cycle
9  # 4. get_item - return the data of the k'th element
10 # 5. is_palindrome - check if list s palindrome
11 # 6. have_intersection - check if 2 lists connect in same point
12 # 7. slice - slice the list according to the given arguments
13 # 8. merge_sort - sort a list in merge_sort algorithem
14 ######################################################################
15
16 from sllist import List, Node
17
18
19 def list_len(sll):
20     """ function recieve a list and return the number of node in the list
21     """
22     # set pointer to the head
23     node = sll.head
24     # reset a counter
25     counter = 0
26     # run until point to None and count every run
27     while node is not None:
28         counter += 1
29         node = node.next
30     return counter
31
32
33 def reverse(sll):
34     """
35     Reverses the given list (so the head becomes the last element, and every
36     element points to the element that was previously before it). Runs in O(n).
37     No new object is created.
38     """
39     # set first pointer to the head
40     node1 = sll.head
41     # stop the function in no Nodes
42     if node1 is None:
43         return None
44     # set second pointer to the second Node and set thr first Node to point to
45     # None as the end of the list
46     if node1.get_next() is not None:
47         node2 = node1.get_next()
48         node1.next = None
49     # stop the function in only one Node
50     else:
51         return None
52     # run until reach the end of the list
53     while node2.get_next() is not None:
54         # set third pointer to the next Node
55         node3 = node2.get_next()
56         # set the second node to point to the first one
57         node2.next = node1
58         # move the first and second pointers to the next one's
59         node1, node2 = node2, node3
```

```
    # set the head to the last Node
    sll.head = node2
    # set the new head to point to his preview Node
    sll.head.next = node1


def merge_lists(first_list, second_list):
    """
    Merges two sorted (in ascending order) lists into one new sorted list in
    an ascending order. The resulting new list is created using new nodes
    (copies of the nodes of the given lists). Assumes both lists are sorted in
    ascending order. The original lists should not be modified.
    """
    # create a new list
    new_list = List()
    # set pointers to the head of the lists
    fnode = first_list.head
    snode = second_list.head
    # run until both pointer reach the end of the lists
    while fnode is not None or snode is not None:
        # add the new list node with the data from the first list node
        # and move the pointer forword
        if fnode is not None and (snode is None or\
                                  fnode.get_data() < snode.get_data()):
            new_list.add_first(fnode.get_data())
            fnode = fnode.get_next()
        # add the new list node with the data from the second list node
        # and move the pointer forword
        elif snode is not None and (fnode is None or\
                                    snode.get_data() < fnode.get_data()):
            new_list.add_first(snode.get_data())
            snode = snode.get_next()
        # add the new list nodes with the data from both lists nodes
        # and move the pointers forword
        elif snode.get_data() == fnode.get_data():
            new_list.add_first(fnode.get_data())
            new_list.add_first(snode.get_data())
            fnode = fnode.get_next()
            snode = snode.get_next()
    # reverse the list so it be in ascending order
    reverse(new_list)
    return new_list


def contains_cycle(sll):
    """
    Checks if the given list contains a cycle.
    A list contains a cycle if at some point a Node in the list points to
    a Node that already appeared in the list. Note that the cycle does not
    necessarily contain all the nodes in the list. The original list should
    not be modified.
    Returns true iff the list contains a cycle
    Return False if any Node point to None
    """
    # set the slow moving node as the head
    if sll.head is not None:
        slow_node = sll.head
    else:
        return False
    # set the fast moving node as the head.next
    if sll.head.get_next() is not None:
        fast_node = sll.head.get_next()
    else:
        return False
    # run until the 2 pointers point to the same one or point to None
    while slow_node != fast_node:
        # move the slow node in 1 move and the fast in 2
        slow_node = slow_node.get_next()
```

```
128             if fast_node.get_next() is not None:
129                 if fast_node.next.get_next() is not None:
130                     fast_node = fast_node.next.get_next()
131                 else:
132                     return False
133             else:
134                 return False
135         return True
136 
137 
138 def get_item(sll, k):
139     """
140     Returns the k'th element from of the list.
141     If k > list_size returns None, if k<0 returns the k element from the end.
142     """
143     # set a variables so the phrase in the range be accurate
144     absolute = 1
145     correction = 0
146     # if k negetive reverse the list and change the variables so it be same as
147     # positive k
148     if k < 0:
149         reverse(sll)
150         absolute = -1
151         correction = 1
152     # set pointer to the head of the list
153     node = sll.head
154     # None if list empty
155     if node is None:
156         return None
157     # run until reach the k'th element or return None if reach the end of the
158     # list
159     for node_num in range(absolute*k - correction):
160         if node.get_next() is None:
161             # reverse the list back to original in k negetive
162             if k < 0:
163                 reverse(sll)
164             return None
165         node = node.get_next()
166     # reverse the list back to original in k negetive
167     if k < 0:
168         reverse(sll)
169     return node.get_data()
170 
171 
172 def is_palindrome(sll):
173     """
174     Checks if the given list is a palindrome. A list is a palindrome if
175     for j=0...n/2 (where n is the number of elements in the list) the
176     element in location j equals to the element in location n-j.
177     Note that you should compare the data stored in the nodes and
178     not the node objects themselves. The original list should not be modified.
179     Returns true iff the list is a palindrome
180     """
181     # set pointers to head
182     head = node_from_end = node = sll.head
183     # return True if list empty
184     if node is None:
185         return True
186     # find the list length
187     length = list_len(sll)
188     # move node_from_end pointer to the last Node and set the list head to the
189     # middle
190     for node_pos in range(length - 1):
191         if node_pos == (length//2 - 1):
192             sll.head = node_from_end.get_next()
193         node_from_end = node_from_end.get_next()
194     # reverse the second half of the list
195     reverse(sll)
```

-1
in the ex description it says not to change the list nodes.
you can find the k<0 by getting the k+length(list) node.

```
196         # compare the first half and the reversed second half
197         for node_pos in range(length // 2):
198             # in case list isn't palindrome reset the list to the original state
199             if node.get_data() != node_from_end.get_data():
200                 reverse(sll)
201                 sll.head = head
202                 return False
203             # move the pointers forward
204             node, node_from_end = node.get_next(), node_from_end.get_next()
205         # reset the list to the original state
206         reverse(sll)
207         sll.head = head
208         return True
209 
210 
211     def have_intersection(first_list, second_list):
212         """
213         Checks if the two given lists intersect.
214         Two lists intersect if at some point they start to share nodes.
215         Once two lists intersect they become one list from that point on and
216         can no longer split apart. Assumes that both lists does not contain cycles.
217         Note that two lists might intersect even if their lengths are not equal.
218         No new object is created, and niether list is modified.
219         Returns true iff the lists intersect.
220         """
221         # reverse the first list, set pointer to the last Node and reverse back
222         reverse(first_list)
223         node1 = first_list.head
224         reverse(first_list)
225         # reverse the second list, set pointer to the last Node and reverse back
226         reverse(second_list)
227         node2 = second_list.head
228         reverse(second_list)
229         # return True if bose pointer are identiacal and not None
230         if node1 == node2 and node1 is not None and node2 is not None:
231             return True
232         return False
233 
234 
235     def slice(sll, start, stop=None, step=1):
236         """ Returns a new list after slicing the given list from start to stop
237         with a step.
238         Imitates the behavior of slicing regular sequences in python.
239         slice(sll, [start], stop, [step]):
240         With 4 arguments, behaves the same as using list[start:stop:step],
241         With 3 arguments, behaves the same as using list[start:stop],
242         With 2 arguments, behaves the same as using list[:stop],
243         """
244         # create a new list
245         new_list = List()
246         # set a variable to know if stop if before the head of the list
247         stop_beyond_first = 0
248 
249         # if only 2 arguments given, set the start as 0 and stop as the second
250         # argument
251         if stop is None:
252             start, stop = 0, start
253 
254         # set the start and stop to a non negetive point if needed and return empty
255         # list if start,stop and step giving no range of nodes
256         if stop < 0:
257             stop = list_len(sll) + stop
258             if stop < 0:
259                 stop = 0
260                 stop_beyond_first = 1
261         if start < 0:
262             start = list_len(sll) + start
263             if start < 0:
```

-1
in the ex description it says not to change the list nodes.
just run on both list to get the last node and compare them.

```
            start = 0
            stop_beyond_first = 0
    if start <= stop and stop <= 0 and step < 0 and stop_beyond_first == 0:
        return new_list
    if start >= stop and step > 0:
        return new_list
    if stop > list_len(sll):
        stop = list_len(sll)
    if start >= list_len(sll) and step < 0:
        start = list_len(sll)-1
    if start == 0 and stop == -1:
        return new_list

    # calculte the number of steps needed to do in the list
    step_div = (stop-start-stop_beyond_first) / step
    step_mod = (stop-start-stop_beyond_first) // step
    if step_div > step_mod:
        steps = step_mod + 1
    else:
        steps = step_mod

    # add new Nodes with the data from the given list in the right places
    for i in range(steps):
        data = get_item(sll, start + i*step)
        if data is not None:
            new_list.add_first(get_item(sll, start + i*step))

    # reverse the list to be in the right direction
    reverse(new_list)
    return new_list


def merge_sort(sll):

    """
    Sorts the given list using the merge-sort algorithm.
    Resulting list should be sorted in ascending order. Resulting list should
    contain the same node objects it did originally, and should be stable,
    i.e., nodes with equal data should be in the same order they were in in the
    original list. You may create a constant number of new to help sorting.
    """

    def length_from_node(head):
        """the function recieve a Node and return the length of the list
        from the Node to the end of the list
        """
        # set a counter
        counter = 0
        # count until reach the end of the list
        while head is not None:
            counter += 1
            head = head.next
        return counter

    # return None if list length >= 1
    if sll.head is None:
        return None
    if sll.head.next is None:
        return None

    # create new Node to point on the sorted list
    new_node = Node(None)

    def spliter(head):
        """Sorts the given list using the merge-sort algorithm.
        Resulting list should be sorted in ascending order."""

        # take the length of the list
```

```
        length = length_from_node(head)
        # set pointer to the head
        node = head

        # return the head if list length == 1
        if length == 1:
            return head
        # move the pointer to the middle
        for i in range(length//2 - 1):
            node = node.next
        if length % 2 == 1:
            node = node.next
        # set pointers to head and middle of the list and disconnect the halves
        left = head
        right = node.next
        node.next = None

        # continue spliting the halves
        left = spliter(left)
        right = spliter(right)

        # set pointer for the sorted list
        sort_node = new_node

        # merge the nodes in ascending order
        while left and right:
            if left.data <= right.data:
                sort_node.next = left
                left = left.next
            else:
                sort_node.next = right
                right = right.next
            sort_node = sort_node.next
        # if only one head remained, add him to the sorted pointer
        sort_node.next = right if right else left

        # return the head of the sorted nodes
        return new_node.next
    # set the head to point on the sorted list
    sll.head = spliter(sll.head)

```