I have used Node class and linkedlist class which had been taught on the class.

def outputQ(q):

pointer=q.head

while pointer:

print(pointer.element)

pointer=pointer.pointer

this function is for output all the element in a linkedlist

def add\_Linkedlist(g,h,j):

new\_list=Linkedlist()

if isinstance(g,Linkedlist):

pivot=g.head

while pivot:

new\_list.add\_last(pivot.element)

pivot=pivot.pointer

if isinstance(h,Linkedlist):

pivot=h.head

while pivot:

new\_list.add\_last(pivot.element)

pivot=pivot.pointer

if isinstance(j,Linkedlist):

pivot=j.head

while pivot:

new\_list.add\_last(pivot.element)

pivot=pivot.pointer

return new\_list

this function is for adding three linkedlist together one by one

Then my program has 2 versions: for the first one, the input of the function is a linkedlist, and the output of the function is also a linkedlist:

def Linked\_Quicksort(n):

if not (isinstance(n,Linkedlist) or n==None):

print('Your input is wrong')

if n.size<=1:

return n

if n.size>1:

small\_list=Linkedlist()

large\_list=Linkedlist()

mid\_list=Linkedlist()

pivot=n.head

k=n.head.element

while pivot:

if pivot.element<k:

small\_list.add\_last(pivot.element)

elif pivot.element>k:

large\_list.add\_last(pivot.element)

elif pivot.element==k:

mid\_list.add\_last(pivot.element)

pivot=pivot.pointer

small\_list=Linked\_Quicksort(small\_list)

large\_list=Linked\_Quicksort(large\_list)

n=add\_Linkedlist(small\_list,mid\_list,large\_list)

return n

The first number in the linkedlist is the standard value. My method is to search the whole linkedlist, put all the element into 3 linkedlist. For the small\_list, it contains all the numbers smaller than the value. For the mid\_linkedlist, it contais all the numbers equal to the value. For the large\_list, it contains numbers larger than the values. Then we use recursive algorithm to the small\_list and large\_list, only when there’s only 0 or 1 nodes in the linkedlist the recursion will stop. Finally, we use add\_linkedlist function to add them together and output the combined linkedlist.

For the second version, the input is a Node and the output is a node:

def Linked\_Quicksort(n):

if not (isinstance(n,Node) or n==None):

print('Your input is wrong')

return

if n==None:

return n

if n.pointer==None:

m=Linkedlist()

m.add\_first(n.element)

return m

elif not n.pointer==None:

small\_list=Linkedlist()

large\_list=Linkedlist()

mid\_list=Linkedlist()

pivot=n

k=n.element

while pivot:

if pivot.element<k:

small\_list.add\_last(pivot.element)

elif pivot.element>k:

large\_list.add\_last(pivot.element)

elif pivot.element==k:

mid\_list.add\_last(pivot.element)

pivot=pivot.pointer

small\_list=Linked\_Quicksort(small\_list.head)

large\_list=Linked\_Quicksort(large\_list.head)

n=add\_Linkedlist(small\_list,mid\_list,large\_list)

return n

def main(n):

if not (isinstance(n,Node) or n==None):

print('Your input is wrong')

return

else:

return Linked\_Quicksort(n).head

The main part is also the same, we only use node to represent the special situation. However, in order to make the recursive algorithm correct, the output of the quicksort function is still linkedlist because we need use the result of recursive algorithm for the small\_list and large\_list, they must belong to linkedlist club. To solve this problem, I define an extra main function, which use the result of quicksort function and output the head, which is a node, of the linkedlist which has been sorted into the ascending order.

Test for Version1:

a=Linkedlist()

a.add\_last(10)

a.add\_last(8)

a.add\_first(9)

a.add\_last(15)

outputQ(Linked\_Quicksort(a))

Test for Version2:

a=Linkedlist()

a.add\_last(10)

a.add\_last(8)

a.add\_last(9)

a.add\_last(7)

print(main(a.head).element)