# 3.2

class MinStack(object):

def \_\_init\_\_(self, head = None):

self.head = head

self.\_min = 0

self.\_prevmin = 0

def push(self, data):

if self.head is None:

self.head = Node(data):

self.\_min = data

else:

n = Node(data)

setMin(data)

n.next = self.head

self.head = n

def pop(self):

if self.head is None:

raise EmptyStackException()

else:

n = self.head

self.head = self.head.next

return n.data

def setMin(self, data):

if self.\_min > data:

def min(self):

return self.\_min

class Node(object):

def \_\_init\_\_(self, data):

self.data = data

self.next = None

class Stack(object):

def \_\_init\_\_(self, head = None):

self.head = None

self.capacity = 0

def push(self, data):

if head is None:

node = Node(data)

head = node

self.capacity += 1

else:

node = Node(data)

node.next = head

head = node

self.capacity += 1

def pop(self):

if head is None:

return None

else

self.head = self.head.next

self.capacity -= 1

return self.head.data

#3.3

class SetOfStacks(object):

\_max = 10

def \_\_init\_\_(self):

self.stack\_pointer = 0

self.stacks = []

def push(self, data):

s = getLastStack()

if s.capacity < \_max:

s.push(data)

else:

s = Stack()

self.stacks.append(s)

self.stack\_pointer += 1

s.push(data)

def pop(self):

s = getLastStack()

if s is None:

self.stack\_pointer -= 1

s = getLastStack

val = s.pop()

if val == -1:

def getLastStack(self):

stack\_addr = self.stacks[self.stack\_pointer]

if stack\_addr is None:

return None

return stack\_addr

# implement a queue using two stacks

class MyQueue(object):

def \_\_init\_\_(self):

self.stack\_top = Stack()

self.stack\_bottom = Stack()

def add(self, data):

stack\_top.push(data)

def transfer():

while(stack\_top is not None):

stack\_bottom.push(stack\_top.pop())

def remove(self):

transfer():

return stack\_bottom.pop()

# sorting a stack

def stackSort(st):

temp = Stack()

if st is None:

return st

while st:

val = st.pop()

if temp is None:

temp.push(val)

else:

ele = st.pop().data

while ele.data < temp.peek()

buff = temp.pop()

if buff is None:

temp.push(ele)

break

st.push(buff)

temp.push(ele)

while temp:

st.push(r.pop())

# animal shelter

class Node(object):

def \_\_init\_\_(self, typ):

self.type = typ

self.next = None

class Queue(object):

def \_\_init\_\_(self, head = None, tail = None):

self.head = head

self.tail = tail

def enqueue(self, animal):

node = Node(animal)

if self.head is None:

self.head = node

self.tail.next = node

self.tail = node

def dequeueAny(self):

if self.head == None:

tail = None

raise EmptyQueueException()

val = head.data

head = head.next

class AnimalQueue(object):

dogs = Queue()

cats = Queue()