C-5.19

For appending,

def append(self,obj):

if self.\_n==self.\_capacity:

self.\_resize(2\*self,\_capacity)

self.\_A[self.\_n]=obj

self.\_n+=1

def \_resize(self,c):

B=self.\_make\_array(c)

for k in range(self.\_n):

B[k]=self.\_A[k]

self.\_A=B

self.\_capacity=c

def \_make\_array(self.c):

return (c\*ctypes.py\_object)()

As you can see, for each resizing, we need to go through a k-loop in which k=, so the time for whole loops is  given that we append  times. The time per insert then is  So inserting takes big-Oh(1) times.

For popping, similarly, suppose the original length is a and for each resizing, we need to go through a k-loop in which k=, so the time for whole loops is , given that we popped times. The time per pop then is .

. So popping also takes big-Oh(1) times.

In conclusion, because each time of appending or popping takes big-Oh(1) times, n times of appending or popping will take big-Oh(n) times.