Problem 1

In [8]:

```
def Merge(L1, L2):
    if L1 and L2:
        if L1[0]>L2[0]:
            return [L2[0]]+Merge(L2[1:],L1)
        return [L1[0]]+Merge(L1[1:],L2)
    return L1+L2

def MergeSort(items):
    if (len(items)<=1):
        return items

    return Merge(MergeSort(items[:(len(items)//2)]),MergeSort(items[(len(items)//2):]))

# Testing for part 1c
mylist1=[]
print("Unsorted List: ",mylist1)
newlist1=MergeSort(mylist1)
print("Sorted List: ", newlist1)
print()
#L2
mylist2=[159,43,2,1,3,7,8,5,4,9,43,10,57,98,33,12]
print("Unsorted List: ",mylist2)
newlist2=MergeSort(mylist2)
print("Sorted List: ", newlist2)
print()
#L3
mylist1=[33,77,93,35,16,32,45,91,2,20,60,71,100,55,73]
print("Unsorted List: ",mylist1)
newlist1=MergeSort(mylist1)
print("Sorted List: ", newlist1)
print()
#L4
mylist4=[10,9,8,7,6,5,4,3,2,1]
print("Unsorted List: ",mylist4)
newlist4=MergeSort(mylist4)
print("Sorted List: ", newlist4)
print()
#L5
mylist5=[100,98,77,21,-500]
print("Unsorted List: ",mylist5)
newlist5=MergeSort(mylist5)
print("Sorted List: ", newlist5)
print()
#L6
mylist6=[-6,-8,-49,-80,-31,-97,-44,-98,-74,-95,84,-26,50,2,65,-62,-28,16,-94,60,72]
print("Unsorted List: ",mylist6)
newlist6=MergeSort(mylist6)
print("Sorted List: ", newlist6)
print()
#L7
mylist7=[14.74,53.44,-49.19,-24.68,-83.66,-43.26,-30.34,-52.73,57.29,73.34,-29.8,21.07,
print("Unsorted List: ",mylist7)
newlist7=MergeSort(mylist7)
print("Sorted List: ", newlist7)
```

```
print()
#L8
mylist8=[99.56,-54.59,-61.4,22.04,-66.98,92.24,36.87,-49.66]
print("Unsorted List: ",mylist8)
newlist8=MergeSort(mylist8)
print("Sorted List: ", newlist8)
```

```
Unsorted List:  []
Sorted List:  []

Unsorted List:  [159, 43, 2, 1, 3, 7, 8, 5, 4, 9, 43, 10, 57, 98, 33, 12]
Sorted List:  [1, 2, 3, 4, 5, 7, 8, 9, 10, 12, 33, 43, 43, 57, 98, 159]

Unsorted List:  [33, 77, 93, 35, 16, 32, 45, 91, 2, 20, 60, 71, 100, 55, 73]
Sorted List:  [2, 16, 20, 32, 33, 35, 45, 55, 60, 71, 73, 77, 91, 93, 100]

Unsorted List:  [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
Sorted List:  [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

Unsorted List:  [100, 98, 77, 21, -500]
Sorted List:  [-500, 21, 77, 98, 100]

Unsorted List:  [-6, -8, -49, -80, -31, -97, -44, -98, -74, -95, 84, -26, 50, 2, 65, -6
2, -28, 16, -94, 60, 72]
Sorted List:  [-98, -97, -95, -94, -80, -74, -62, -49, -44, -31, -28, -26, -8, -6, 2, 1
6, 50, 60, 65, 72, 84]

Unsorted List:  [14.74, 53.44, -49.19, -24.68, -83.66, -43.26, -30.34, -52.73, 57.29, 7
3.34, -29.8, 21.07, -43.84]
Sorted List:  [-83.66, -52.73, -49.19, -43.84, -43.26, -30.34, -29.8, -24.68, 14.74, 21.
07, 53.44, 57.29, 73.34]

Unsorted List:  [99.56, -54.59, -61.4, 22.04, -66.98, 92.24, 36.87, -49.66]
Sorted List:  [-66.98, -61.4, -54.59, -49.66, 22.04, 36.87, 92.24, 99.56]
```

Problem 2

Statement	Reason	
Q={a,b$	$(a+b)\div$b=a}	Defintion
0∈Q?	Yes because...	
(0+0)∸b = 0∸b	Defintion of Addition	
Assume a, b∈Q	Induction	
Are a' & b' ∈Q?	Yes because...	
(a'+b')∸b'=a'	Substitution	
(a'+b')∸b = (a+b')'∸b'	Definition of Addition	
= ((a+b')b')'	Definition of Addition	
= a'	Induction	
So a' & b' ∈Q		
Q=N	4th Postulate	
For all a,b in N (a+b)∸b=a	Q=N and induction of a,b makes it true for any q in N	

Problem 3

Sample input file input.fsa (derived from aabc example in class)

In [ ]:

```
A abc
S start,0,a,0,aa,0,aab,0,aabc,1
B start
D start,a,a,start,b,start,start,c,start
D a,a,aa,a,b,start,a,c,start
D aa,a,aa,aa,b,aab,aa,c,start
D aab,a,a,aab,b,start,aab,c,aabc
D aabc,a,aabc,aabc,b,aabc,aabc,c,aabc
T aabc
O
T aaaaa
O
T abcba
O
T aaaabcab
O
T abcbaababca
O
T ccbaacaabcabb
O
T cbaa
O
T
O
T abc
O
```

Python code file prob3.py

In [7]:

```
class FSA:
  #default constructor
  def __init__(self):
    self.a=[]
    self.s=[]
    self.b=""
    self.d=[]
    self.t=[]
    self.o=[]
  def p(self): #used to test if FSA had desired info
    print(self.a)
    print(self.s)
    print(self.b)
    print(self.d)
    print(self.t)
    print(self.o)

  #setters
  def setA(self,str):
    self.a=list(str)
    self.a.pop() #remove endl
  def setS(self,L):
    self.s=self.s+L
  def setB(self,str):
```

```
    self.b=str
  def setD(self,L):
    self.d=self.d+L
  def setT(self,str):
    self.t.append(str)

  def checkTape(self,str):
    #str=self.t[0]
    if str=='': #check for empty string
      self.o.append('rejected')
    else:
      #go through every state of fsa
      curr_state=self.b
      for c in str: #get every letter of T
        let=c
        for subL in self.d:
          if subL[0]==curr_state and subL[1]==let:
              curr_state=subL[2]
              break #we only want this to be true once for every letter

      #check to see if curr_state is final
      for subLs in self.s:
        if subLs[0]==curr_state:
          if subLs[1]=='1':
            self.o.append('accepted')
          else:
            self.o.append('rejected')

  def checkAllTapes(self): #check every tape in self.t
    for ele in self.t:
      self.checkTape(ele)
  def reconstruct(self,f):
    for x in range(0,len(self.t)):
      f.write("T ")
      strt=self.t[x]+'\n'
      f.write(strt)
      f.write("O ")
      stro=self.o[x]+'\n'
      f.write(stro)


#main----------------------------------------------------------------------
infile = open("input.fsa","r")
outfile = open("output.fsa","w")

#initalize variables in FSA class-------------------------------------------
fobj=FSA()

stri_a=infile.readline()
fobj.setA(stri_a[2:])
outfile.write(stri_a)

lines=infile.readlines()
for cl in lines:
  if cl[0]=='S' or cl[0]=='B' or cl[0]=='D': #start reconstructing output file
    outfile.write(cl)

  if cl[0]=='S':
    cl=cl[2:]
    Li_s=cl.rstrip('\n').split(",") #list of words for state S seperated and no \n
```

```
    Li_sdouble=[Li_s[i:i+2] for i in range(0,len(Li_s),2)]
    fobj.setS(Li_sdouble)
  if cl[0]=='B':
    fobj.setB(cl[2:].rstrip('\n'))
  if cl[0]=='D':
    cl=cl[2:]
    Li_d=cl.rstrip('\n').split(",")
    Li_dtriple=[Li_d[i:i+3] for i in range(0,len(Li_d),3)]
    fobj.setD(Li_dtriple)
  if cl[0]=='T':
    cl=cl[2:].rstrip('\n')
    fobj.setT(cl)

#check the tapes if accepted or rejected
fobj.checkAllTapes()
#fobj.p()

#write T and O lines
fobj.reconstruct(outfile)
outfile.close()
```

sample output file used output.fsa

In [ ]:

```
A abc
S start,0,a,0,aa,0,aab,0,aabc,1
B start
D start,a,a,start,b,start,start,c,start
D a,a,aa,a,b,start,a,c,start
D aa,a,aa,aa,b,aab,aa,c,start
D aab,a,a,aab,b,start,aab,c,aabc
D aabc,a,aabc,aabc,b,aabc,aabc,c,aabc
T aabc
O accepted
T aaaaa
O rejected
T abcba
O rejected
T aaaabcab
O accepted
T abcbaababca
O rejected
T ccbaacaabcabb
O accepted
T cbaa
O rejected
T
O rejected
T abc
O rejected
```

Problem 4

sample input file input.ndfsa

In [ ]:

```
A 1,0
S a,0,b,0,c,0,e,0,f,0,d,1
```

```
B a
D a,1,b,a,0,e
D b,1,c,b,@,d
D c,1,d
D e,@,b,e,@,c,e,0,f
D f,0,d
T 0
O
T 1
O
T 11
O
T 10
O
T 01
O
T 111
O
T 00
O
T 000
O
T 011
O
T 1111
O
T 001
O
```

Code for problem 4 (prob4.py)

In [ ]:

```
class NDFSA:
    def __init__ (self):
        self.A=[] #only works if letter in alphabet are seperated by commas
        self.S=[]
        self.B=""
        self.D=[]
        self.T=[]
        self.O=[]

    def getA(self,line):
        while (line!=""):
            subline=line.partition(',') #returns a tuple seperating what is before , an
            let=subline[0]
            line=subline[2]

            if (let!='@'): #the free state can not be part of the alphabet
                self.A+=[let]
            else:
                return False #if it is then contents of file become invalid
        return True

    def getS(self,line):
        while (line!=""):
            subline=line.partition(',')
            state=subline[0]
            line=subline[2]

            subline=line.partition(',')
```

```
            state_val=subline[0]
            line=subline[2]

            self.S+=[[state,state_val]]

    def getB(self,line):
        self.B = line

    def getD(self,line):
        while (line!=""):
            subline=line.partition(',')
            start_state=subline[0]
            line=subline[2]

            subline=line.partition(',')
            trans_state=subline[0]
            line=subline[2]

            subline=line.partition(',')
            final_state=subline[0]
            line=subline[2]

            self.D+=[[start_state,trans_state,final_state]]

    def getT(self,line):
        self.T+=[line]

    def getO(self,line):
        if (line==""):
            self.O+=[self.tape_accepted(len(self.T)-1)]
        else:
            self.O+=[line]

    def print(self): #used to test if NDFSA has desired info
        print(self.A)
        print(self.S)
        print(self.B)
        print(self.D)
        print(self.T)
        print(self.O)

    #checks to see if each tape will be accepted or rejected
    def tape_accepted(self,T_pos):
        for i in range(len(self.T[T_pos])): #go through tape letter by letter
            found=False
            for j in range(len(self.A)):
                if (self.T[T_pos][i]==self.A[j]):
                    found=True
            if (found==False):
                return "rejected"

        #see if the potential transition is possible
        tape=self.T[T_pos]
        state=self.B #start state

        #Check all potential paths of NDFSA for this tape
        accepted=self.check_paths(tape,state)
        return accepted

    #checks all possible paths for a given tape recursively
```

```
    def check_paths(self,tape,state):
        accepted="rejected" #O line
        if tape=="":
            for i in range(len(self.S)):
                if self.S[i][0]==state:
                    if self.S[i][1]=='1': #if state is final then stop
                        accepted="accepted"
                        return accepted
                    elif self.S[i][1]=='0':
                        for i in range(len(self.D)):
                            state_check=self.D[i]
                            if (state_check[0]==state and state_check[1]=='@'):
                                accepted=self.check_paths(tape,state_check[2])
                                if accepted=="accepted":
                                    return accepted
                        return accepted
        else:
            found=False

            for i in range(len(self.D)): #find next possible state
                state_check=self.D[i]
                if (state_check[0]==state and state_check[1]==tape[0]):
                    found=True
                    accepted=self.check_paths(tape[1:],state_check[2])
                    if accepted=="accepted":
                        return accepted
                elif (state_check[0]==state and state_check[1] == '@'):
                    found=True
                    accepted=self.check_paths(tape,state_check[2])
                    if accepted=="accepted":
                        return accepted
            if (found==False):
                return "rejected"
            return accepted

    def get_input(self):
        formatok=True
        for line in infile.readlines():
            if line[0]=='A':
                formatok=self.getA(line[2:-1]) #we don't want letter, space or \n
            elif line[0]=='S':
                self.getS(line[2:-1])
            elif line[0]=='B':
                self.getB(line[2:-1])
            elif line[0]=='D':
                self.getD(line[2:-1])
            elif line[0]=='T':
                self.getT(line[2:-1])
            elif line[0]=='O':
                if line[-1]=='\n': #new line signifies end of file
                    self.getO(line[2:-1])
                else:
                    self.getO(line[2:])
            else:
                formatok=False
                return formatok

            if line[1]!=' ': #make sure there is a space after first letter
                formatok=False
                return formatok
```

```
        return formatok

    def reconstruct(self):
        line="A "
        for i in range(len(self.A)):
            line += self.A[i]+","
        if line[-1]==',': #remove extra quote if last thing
            line=line[:-1]
        line+="\n"
        outfile.write(line)

        line="S "
        for i in range(len(self.S)):
            line += self.S[i][0]+","+self.S[i][1]+","
        if line[-1]==',':
            line=line[:-1]
        line+="\n"
        outfile.write(line)

        line="B "+self.B+"\n"
        outfile.write(line)

        line="D "
        for i in range(len(self.D)):
            line="D "
            line += self.D[i][0]+","+self.D[i][1]+","+self.D[i][2]+","
            if line[-1]==',':
                line=line[:-1]
            line+="\n"
            outfile.write(line)

        for i in range(len(self.T)):
            line="T "+self.T[i]+"\n"
            outfile.write(line)
            line="O "+self.O[i]
            if i!=(len(self.T)-1):
                line+="\n"
            outfile.write(line)

#main----------------------------------------------------------------
infile=open("input.ndfsa","r")
machine=NDFSA()
format_ok=machine.get_input()
infile.close()
if format_ok:
    outfile=open("output.ndfsa","w")
    machine.reconstruct()
    outfile.close()
else:
    print("The input file was not formatted correctly, so it was rejected.")
```

sample output file output.ndfsa

In [ ]:

```
A 1,0
S a,0,b,0,c,0,e,0,f,0,d,1
B a
D a,1,b
D a,0,e
D b,1,c
```

```
D b,@,d
D c,1,d
D e,@,b
D e,@,c
D e,0,f
D f,0,d
T 0
O accepted
T 1
O accepted
T 11
O rejected
T 10
O rejected
T 01
O accepted
T 111
O accepted
T 00
O rejected
T 000
O accepted
T 011
O accepted
T 1111
O rejected
T 001
O rejected
```

Problem 5

In [ ]:

```
A a,b
S Empty,0,oddB,1,BB,0,oddBA,1,evenB,0,evenBA,0
B Empty
D Empty,a,Empty,Empty,b,oddB,oddB,b,BB,oddB,a,oddBA,BB,a,BB
D BB,b,BB,oddBA,a,oddBA,oddBA,b,evenB,evenB,a,evenBA
D evenB,b,BB,evenBA,a,evenBA,evenBA,b,oddB
T bab
O rejected
T babab
O accepted
T abab
O rejected
T aaaaa
O rejected
T bababababababababababa
O accepted
T abababababababababababababababababababababab
O accepted
T bababababab
O accepted
T abaaabab
O accepted
T abababababababababababababababababababababababab
O accepted
T abababababababab
O accepted
```