REPORT FOR ITCS6114 PROJECT 1

In this project different sorting algorithms have been implemented. It includes Merge sort, Heap sort, Quick sort, Modified Quick sort, Insertion sort.
For Quick sort first element is used as a pivot and for modified quick sort median of three logic is used to find the pivot. In case of modified quick sort when size drops below 10 insertion sorting technique is implemented.

INPUT: the program automatically takes input list of random numbers in sizes varying from 100 to 4000 over a step of 500. So, user has no need to manually enter the input

The program automatically finds the sorted list and reverse sorted list to find execution time for for same input by different algorithms.

OUTPUT: since list if size 4000 covers up the screen the program doesn't display the sorted list. However, the user can uncomment the required print statement to see the result.

The output is graphically displayed using Three images,

1) for original input
2) for sorted input
3) for reverse sorted input

CODE:

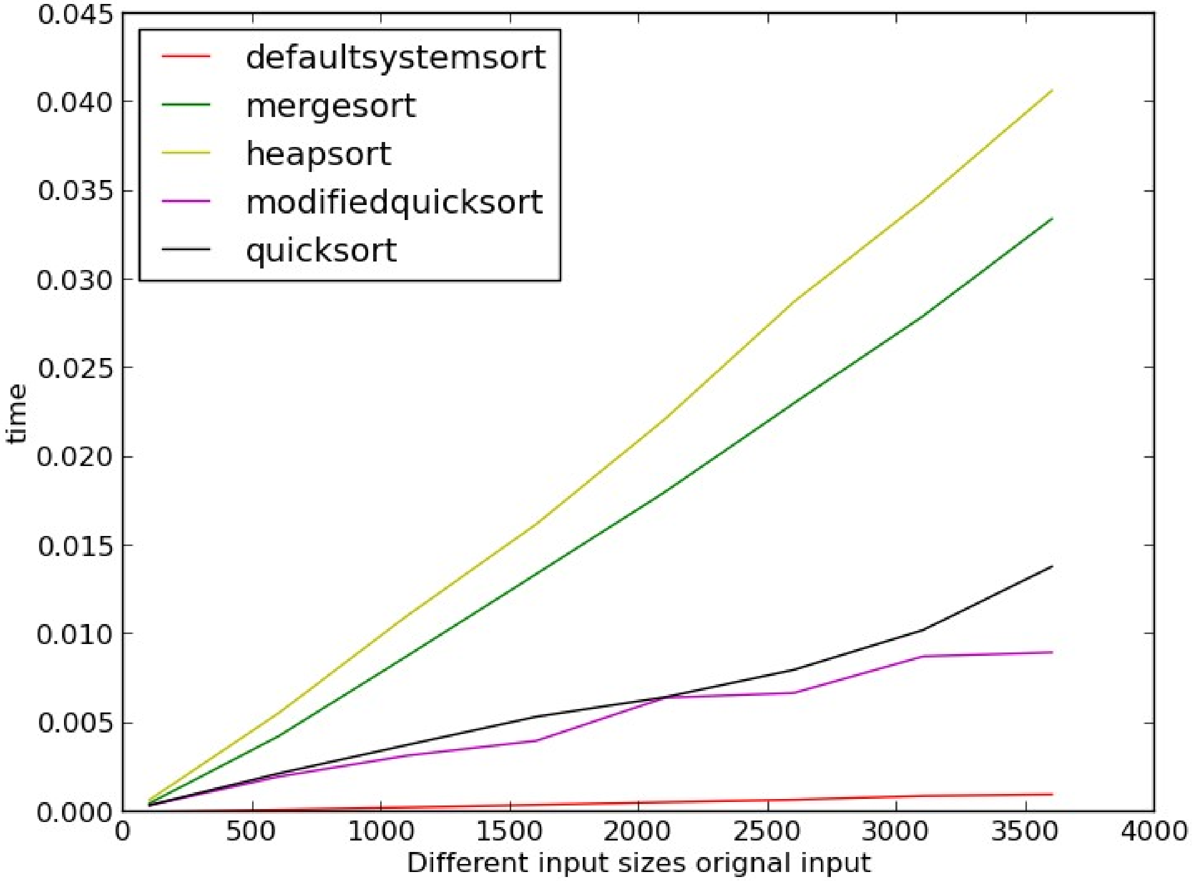
```
from random import randint
import matplotlib.pyplot as p
import time
from PIL import Image
#--------------> execution time list global
#---------------
global exetimeS
global exetimeI
global exetimeH
global exetimeQ
global exetimeM
global exetimemQ
global C
exetimeS = []
exetimeI = []
exetimeH = []
exetimeQ = []
exetimeM = []
exetimemQ = []
C = []
#---------------
#---------------
global sortedexetimeS
global sortedexetimeI
global sortedexetimeH
global sortedexetimeQ
global sortedexetimeM
global sortedexetimemQ
global sortedC
sortedexetimeS = []
sortedexetimeI = []
sortedexetimeH = []
sortedexetimeQ = []
sortedexetimeM = []
sortedexetimemQ = []
sortedC = []
#---------------
#---------------
global revsorexetimeS
global revsorexetimeI
global revsorexetimeH
global revsorexetimeQ
global revsorexetimeM
global revsorexetimemQ
global revsorC
revsorexetimeS = []
revsorexetimeI = []
revsorexetimeH = []
revsorexetimeQ = []
revsorexetimeM = []
revsorexetimemQ = []
revsorC = []
#---------------
#------ System Sorting() ---------------------------->
# input List
# output sorted list by the system
def systemsort(s):
        global exetimeS
        #print "-------The list after System sort-------->"
        start_time = time.time()
        s.sort()
        exetimeS.append( time.time() - start_time )
        #print s
        return s
#------ System Sorting()~ ---------------------------->

#------ Insertion sort() ----------------------------->
def insertionsort(I):
        global exetimeI
        #print "------ The list after insertion sort------->"
        start_time = time.time()
        for x in range(1, len(I)):
                value = I[x]
                y = x
                while (y > 0) and (I[y-1] > value):
                        I[y] = I[y-1]
                        y = y -1
                I[y] = value
        exetimeI.append( time.time() - start_time )
        return I

#------ Insertion sort()~ --------------------------->
#------ Heap sort()---------------------------------->
def heapsort( H ):
        # converting the given list to heap
        length = len( H ) - 1
        bt = length / 2
        for i in range ( bt, -1, -1 ):
                Push_Down( H, i, length )

        for i in range ( length, 0, -1 ):
                if H[0] > H[i]:
                        swap( H, 0, i )
                        Push_Down( H, 0, i - 1 )

        #print H

def Push_Down( H, first, last ):
        largest = 2 * first + 1
        while largest <= last:

                if ( largest < last ) and ( H[largest] < H[largest + 1] ):
                        largest += 1


                if H[largest] > H[first]:
                        swap( H, largest, first )

                        first = largest;
                        largest = 2 * first + 1
                else:
                        return



def swap( Alist, x, y ):
        tmp = Alist[x]
        Alist[x] = Alist[y]
        Alist[y] = tmp

#------ Heap sort()~--------------------------------->
#------ Quick sort()--------------------------------->
def quick_sort(listq):
   pivot = 0 # pivot is the first element
```

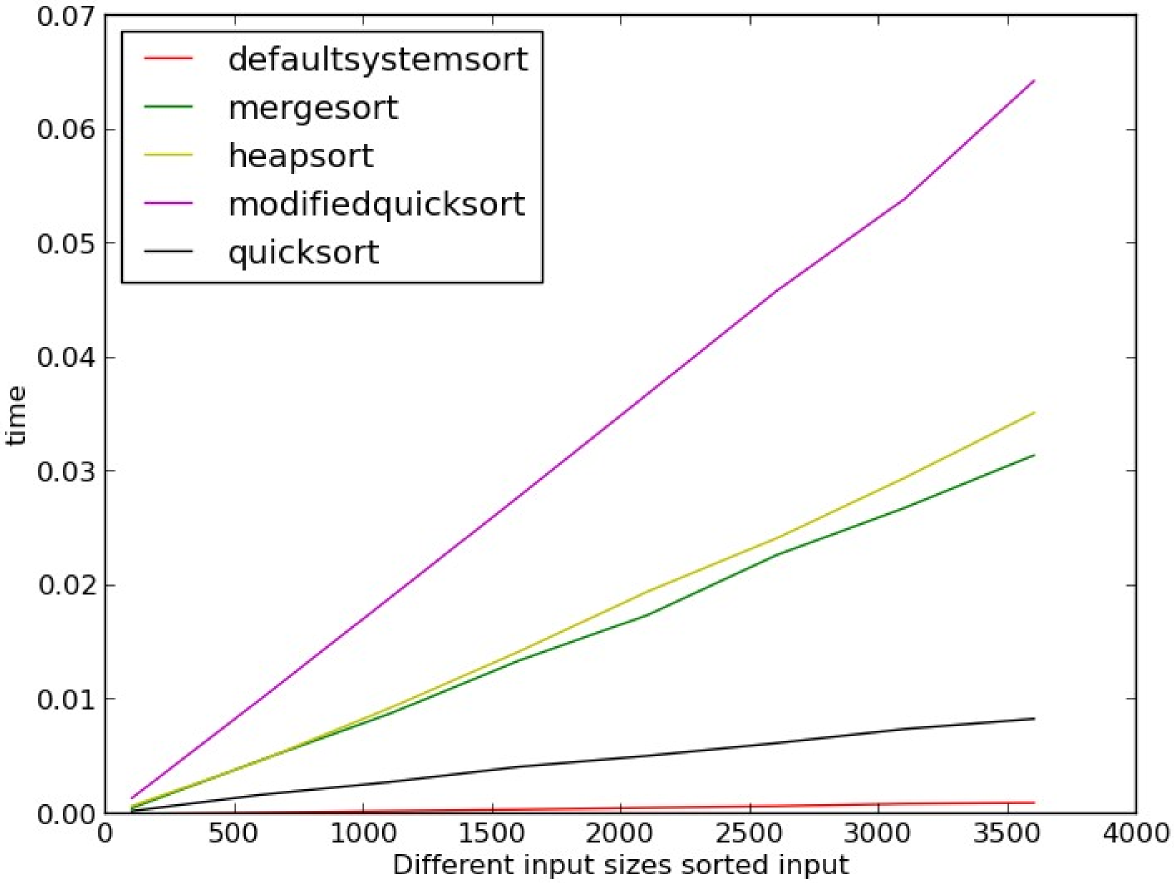
```
    if len(listq) == 0:
        return []
    return  quick_sort(filter( lambda item: item < listq[0],listq)) + [v for v in listq if v==listq[0] ]  + quick_sort( filter( lambda item: item > listq[0], listq))
#------- Quick sort()~-------------------------------->
#------ Modified Quick sort()--------------------------------->
def mquick_sort(listq):
    pivot = getMedian(list(listq), 0, len(list(listq))-1) #pivot is the median
    if len(listq) == 0:
        return []
    if len(listq) <=10:
            return insertionsort(listq) #for size less than 10 use insertionsort
    return  mquick_sort(filter( lambda item: item < listq[pivot],listq)) + [v for v in listq if v==listq[pivot] ]  +  mquick_sort( filter( lambda item: item > listq[pivot], listq))

def getMedian(ql, left, right):

    if len(ql) <= 2:
            return 0
    if len(ql) >= 3:
            mid = (left + right)/2
            if ql[right] < ql[left]:
                        Swap(ql, left, right)
            if ql[mid] < ql[left]:
                        Swap(ql, mid, left)
            if ql[right] < ql[mid]:
                        Swap(ql, right, mid)
            return mid


def Swap(ql, left, right):
    temp = ql[left]
    ql[left] = ql[right]
    ql[right] = temp

#------- Modified Quick sort()~-------------------------------->
#------ Merge sort -------------------------------->
def msort(x):
    result = []
    if len(x) < 2:
        return x
    mid = int(len(x)/2)
    y = msort(x[:mid])
    z = msort(x[mid:])
    while (len(y) > 0) or (len(z) > 0):
        if len(y) > 0 and len(z) > 0:
            if y[0] > z[0]:
                result.append(z[0])
                z.pop(0)
```

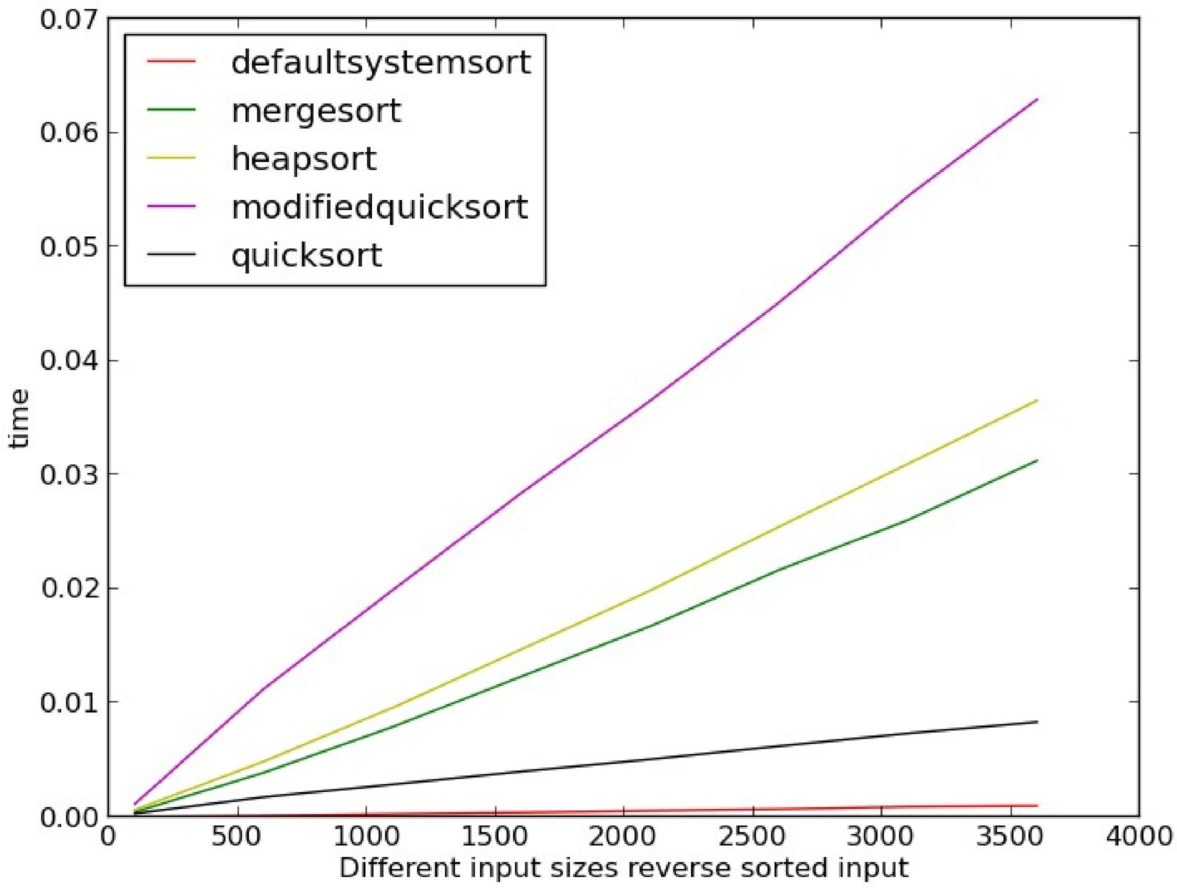
```
      else:
        result.append(y[0])
        y.pop(0)
    elif len(z) > 0:
      for i in z:
        result.append(i)
        z.pop(0)
    else:
      for i in y:
        result.append(i)
        y.pop(0)
  return result
#----------------------------------------------------->

def main():
      print " 1) This program automatically takes random inputs of sizes from 100 to 4000 over a step
of 500"
      print " "
      print "2) It executes Merge, Heap, Quick first element as pivot, Modified Quick (for size < 10
Insertion sort ) (pivot from median)"
      print " "
      print "3) Please uncomment the respective print statement if you wish see the result"
      print  " "
      print "4) This program saves the results in plot.png, reversortedplot.png, sortedplot.png for
orignal, reverse sorted and sorted input respectively"
      for L in range(100, 4000, 500):
            C.append(L)
            global exetimeM
            global exetimeH
            global exetimeQ
            global exetimemQ
            data = []
            sorteddata = []
            reversesorteddata = []
            length = L
            minvalue = 0
            maxvalue = 100
            for i in range(0,length):
                  data.append(randint(minvalue,maxvalue))
            #print "-------The orignal list of numbers-------->"
            #print data
            sorteddata = systemsort(list(data))
            reversesorteddata = list (sorteddata)
            reversesorteddata.reverse()
            #print "-------The sorted list of numbers-------->"
            #print sorteddata
            #print "-------The reverse sorted list of numbers-------->"
            #print reversesorteddata
            #------------------------------------ Merge sort region
            start_time = time.time()
            m = msort(list(data))#mergesort
            exetimeM.append( time.time() - start_time )
            start_time = time.time()
            sm = msort(list(sorteddata))#mergesort
            sortedexetimeM.append( time.time() - start_time )
            start_time = time.time()
            revsm = msort(list(reversesorteddata))#mergesort
            revsorexetimeM.append( time.time() - start_time )
            #-------------------------------------------------------
            #print "-------The list after mergesort-------->"
            #print m
            #print "-------The list after heapsort--------->"
            #-----------------------------------  Heap sort region
            start_time = time.time()
            heapsort(list(data))#heapsort
            exetimeH.append( time.time() - start_time )
            start_time = time.time()
            heapsort(list(sorteddata))#heapsort for sorted
            sortedexetimeH.append( time.time() - start_time )
            start_time = time.time()
            heapsort(list(reversesorteddata))#heapsort for reverse sorted
            revsorexetimeH.append( time.time() - start_time )
            #------------------------------------------------------
            #print "-------The list after quicksort-------->"
            #------------------------------------------ Quick sort region
            start_time = time.time()
            q = quick_sort(list(data))#quicksort
            exetimeQ.append( time.time() - start_time )
            start_time = time.time()
            sq = quick_sort(list(sorteddata))#sortedquicksort
            sortedexetimeQ.append( time.time() - start_time )
            start_time = time.time()
            revsq = quick_sort(list(reversesorteddata))#revsortedquicksort
            revsorexetimeQ.append( time.time() - start_time )
            #-------------------------------------------------------------
            #------------------------------------------ Modified Quick sort region
            start_time = time.time()
            mq = mquick_sort(list(data))#modifiedquicksort
            exetimemQ.append( time.time() - start_time )
            start_time = time.time()
            mq = mquick_sort(list(sorteddata))#modifiedquicksortsorted
            sortedexetimemQ.append( time.time() - start_time )
            start_time = time.time()
            mq = mquick_sort(list(reversesorteddata))#modifiedquicksort reverse sorted
            revsorexetimemQ.append( time.time() - start_time )
            #-------------------------------------------------------------
            #print q
      #print "--------- Exection times--------------->"
```

```
        #print "execetion time of default system sort:", exetimeS
        #print "execetion time of insertion sort:", exetimeI
        #print "execetion time of Heap sort:", exetimeH
        #print "execetion time of Merge sort:", exetimeM
        #print "execetion time of Quick sort:", exetimeQ
        #------------------------------------------------ PLOT 1
        p.plot(C,exetimeS,'r',C,exetimeH,'g',C,exetimeM,'y',C,exetimeQ,'m',C,exetimemQ,'k')
        p.legend(['defaultsystemsort','mergesort','heapsort','modifiedquicksort','quicksort'],loc=2)
        p.show()
        p.xlabel('Different input sizes orignal input')
        p.ylabel('time')
        p.savefig('plot.png')
        p.clf()
        #------------------------------------------------------
        #------------------------------------------------ PLOT 2

p.plot(C,exetimeS,'r',C,sortedexetimeH,'g',C,sortedexetimeM,'y',C,sortedexetimeQ,'m',C,sortedexetime
mQ,'k')
        p.legend(['defaultsystemsort','mergesort','heapsort','modifiedquicksort','quicksort'],loc=2)
        p.show()
        p.xlabel('Different input sizes sorted input')
        p.ylabel('time')
        p.savefig('sortedplot.png')
        p.clf()
        #------------------------------------------------------
        #------------------------------------------------ PLOT 3

p.plot(C,exetimeS,'r',C,revsorexetimeH,'g',C,revsorexetimeM,'y',C,revsorexetimeQ,'m',C,revsorexetime
mQ,'k')
        p.legend(['defaultsystemsort','mergesort','heapsort','modifiedquicksort','quicksort'],loc=2)
        p.show()
        p.xlabel('Different input sizes reverse sorted input')
        p.ylabel('time')
        p.savefig('reversesortedplot.png')
        img = Image.open('reversesortedplot.png')
        img.show()
        img = Image.open('sortedplot.png')
        img.show()
        img = Image.open('plot.png')
        img.show()
        #------------------------------------------------------
if __name__ == "__main__":
   main()
```

OUTPUT:


defaultsystemsort
mergesort
heapsort
modifiedquicksort
quicksort
time
0.045
0.040
0.035
0.030
0.025
0.020
0.015
0.010
0.005
0.000
0
500
1000
1500
2000
2500
3000
3500
4000
Different input sizes orignal input


defaultsystemsort
mergesort
heapsort
modifiedquicksort
quicksort
time
0.00
0.01
0.02
0.03
0.04
0.05
0.06
0.07
0
500
1000
1500
2000
2500
3000
3500
4000
Different input sizes sorted input


0.07
0.06
0.05
0.04
0.03
0.02
0.01
0.00
time
0
500
1000
1500
2000
2500
3000
3500
4000
Different input sizes reverse sorted input
defaultsystemsort
mergesort
heapsort
modifiedquicksort
quicksort