Data Structures and Algorithms.
Author: Isabel Segura Bedmar

Unit 7 – Divide and conquer

Problem - Implement a Python function based on the divide and conquer strategy in order to obtain the maximum element of a Python list.

Solution:

```
from random import randint


def findMax(data):
  if len(data)==1:
    return data[0]

  mid=len(data)//2
  #print(data[0:mid])
  #print(data[mid:])
  part1=data[0:mid]
  part2=data[mid:]
  max1=findMax(part1)
  max2=findMax(part2)
  return max(max1,max2)




data=[]
for i in range(10):
  data.append(randint(-50,50))

print("The maximum element in {} is {} ".format(data,findMax(data)))
```

Problem - Given a sorted list and a number, implement a function that returns True if the number is found in the list and False otherwise. The algorithm should be based on the divide and conquer strategy.

Solution:

```
from random import randint


def binarySearch(data,x):

  if len(data)==0:
    return False

  mid=len(data)//2
  #print(data[0:mid])
  #print(data[mid:])

  if x==data[mid]:
    return True

  if x<data[mid]:
    left=data[0:mid]
    return binarySearch(left,x)

  if x>data[mid]:
    right=data[mid+1:]
    return binarySearch(right,x)



data=[]
for i in range(5):
  data.append(randint(0,10))


data.sort()

x=randint(0,10)

print("binarySearch({},{})={}".format(data,x,binarySearch(data,x)))

for i in range(len(data)):

print("binarySearch({},{})={}".format(data,data[i],binarySearch(data,data[i])
))


for i in range(5):
  x=randint(0,10)
  print("binarySearch({},{})={}".format(data,x,binarySearch(data,x)))
```

Problem - Implement the mergesort algorithm in Python.
Solution:

```
"""# 3) Merge sort"""

def mergesort(A):
  if len(A)>1:
    m=len(A)//2
    l1=A[0:m]
    l2=A[m:]
    mergesort(l1)
    mergesort(l2)
    A=merge(l1,l2)




def merge(l1,l2):
  newList=[]
  i=0
  j=0
  while i<len(l1) and j<len(l2):
    if l1[i]<=l2[j]:
      newList.append(l1[i])
      i+=1
    else:
      newList.append(l2[j])
      j+=1

  while i<len(l1):
    newList.append(l1[i])
    i+=1


  while j<len(l2):
    newList.append(l2[j])
    j+=1

  return newList


#Test mergesort

data=[]
for i in range(10):
  data.append(randint(0,10))
```

```
print("data={}".format(data))
mergesort(data)
print("sort={}".format(sortdata))

"""4) Quicksort"""

from random import *
from time import *

data = []
for i in range(0, 10):
    data.append(randint(0, 100))

print(data)
```

Problem - Implement the quicksort algorithm in Python.
Solution:

```
def quicksort(data):
    _quicksort(data,0,len(data)-1)

def _quicksort(data, left, right):
    i = left
    j = right

    m=(left + right) // 2

    p = data[m] # pivot element in the middle

    while i <= j:
        while data[i] < p:
          i += 1
        while data[j] > p:
          j -= 1
        if i <= j: # swap
            data[i], data[j] = data[j], data[i]
            i += 1
            j -= 1

    if left < j: # sort left list
        _quicksort(data, left, j)
    if i < right: # sort right list
        _quicksort(data, i, right)

print(data)
quicksort(data)
print(data)
```