```python
# TP Ordenamiento Avanzado - Ejercicio 1

from linkedlist import *

import random # para poder generar listas aleatorias para probar las funciones

### QUICKSORT ###

def quickSort(L):
    quickSortR(L,0,length(L)-1)

# inicio y fin son los índices que delimitan la parte de la lista que vamos a
# ordenar en cada llamada a quickSortR()
def quickSortR(L,inicio,fin):
    if fin <= inicio: return
    separador = particiona(L, inicio, fin) # dividimos L en 2 sublistas
    quickSortR(L, inicio, separador-1) # ordenamos la de la izquierda
    quickSortR(L, separador+1, fin) # ordenamos la de la derecha

# divide L en dos, intercambia elementos según proceda y devuelve
# nuevo índice de elemento que separa parte izquierda y derecha
def particiona(L, inicio, fin):
    izq = inicio # índice escaneo lado izquierdo
    dcha = fin+1 # índice escaneo lado derecho
    separador = inicio
    print (" -> Particionando sublista", inicio, "-", fin, end='\n')
    while True:
        # seleccionamos el elemento a intercambiar de la izquierda
        while esMenor(access(L,(izq := izq+1)), access(L,separador)):
            if izq == fin: break
        # seleccionamos el elemento a intercambiar de la derecha
        while esMenor(access(L,separador), access(L,(dcha := dcha-1))):
            if dcha == inicio: break
        # cuando se cruzan los punteros izq y dcha salimos del while(True)
        if izq >= dcha:
            break
        # intercambiamos los elementos izq y dcha de esta sublista
        intercambia(L, izq, dcha)
        print ("    -> Intercambiando", izq,"por", dcha, "queda:", end='')
        imprimelista(L)
    # intercambiamos el separador que hemos usado en el while(True) con dcha
    intercambia(L, inicio, dcha)
    print ("    -> Intercambiando", inicio,"por", dcha, "queda:", end='')
    imprimelista(L)
    # devolvemos el índice del nuevo separador
    return dcha

# devuelve True si i < j, caso contrario devuelve False
def esMenor(i, j):
    if i < j:
        return True
    else:
        return False

# dados dos nodos de una lista (por índice), los intercambia
def intercambia(L,i,j):
    # guardamos el valor de i antes de sobrescribirlo con j
    tmp = access(L,i)
    # ponemos el valor de j en la posición i
    update(L,access(L,j),i)
    # y el valor original de i en la posición j
    update(L,tmp,j)

### MERGESORT ###

def mergeSort(L):
    # llamamos a la función recursiva
    mergeSortR(L, 0, length(L)-1)

def mergeSortR(L, inicio, fin):
    if fin <= inicio:
        return
    mitad = inicio + (fin - inicio) // 2
```

```
    mergeSortR(L, inicio, mitad) # dividimos parte izquierda
    mergeSortR(L, mitad+1, fin) # dividimos parte derecha
    unir(L, inicio, mitad, fin) # juntamos las partes ordenándolas en el proceso

# función que junta L[inicio..mitad] con L[mitad+1..fin], estando estas dos mitades
# ordenadas previamente o siendo de length=1 de tal forma que se ordenan al juntarse
def unir(L, inicio, mitad, fin):
    # punteros:
    i = inicio
    j = mitad + 1
    # creamos lista auxiliar
    AUX = LinkedList()
    # copiamos L a AUX, hace falta un bucle "for" porque si hacemos AUX = L se cambi
an ambas listas a la vez
    # también hace falta capiar en aux hasta L[0] para que coincidan los índices i y
 j
    for k in range(fin, -1, -1):
        add(AUX, access(L, k))
    # mostramos por pantalla lo que vamos haciendo:
    print(" -> Uniendo: ( inicio =", inicio, ", mitad =", mitad, ", fin =", fin, ")"
, end='')
    imprimelista(L)
    # copiamos los elementos de AUX a L de manera ordenada uniéndoles según inicio/m
itad/fin
    for k in range(inicio, fin+1):
        # se acabaron elementos de la izquierda: los tomamos de la derecha
        if   (i > mitad):
            update(L, access(AUX, j), k)
            j += 1
        # se acabaron elementos de la derecha: los tomamos de la izquierda
        elif (j > fin):
            update(L, access(AUX, i), k)
            i += 1
        # si elemento de la dcha. es menor que el de la izq. lo insertamos en L
        elif esMenor(access(AUX,j),access(AUX,i)):
            update(L, access(AUX, j), k)
            j += 1
        # si elemento de la izq. es menor o igual que el de la dcha. lo insertamos e
n L
        else:
            update(L, access(AUX, i), k)
            i += 1

### funciones para PRUEBAS ###

# para generar una lista con valores aleatorios
def listaAleatoria(tamanho,min,max):
    L = LinkedList()
    for i in range(tamanho):
        add(L,random.randint(min,max))
    return L

# para imprimir una lista por pantalla
def imprimelista(L):
    nodo=L.head
    print("   ", end='')
    for i in range(length(L)):
        print(nodo.value, " ", end='', sep='')
        nodo = nodo.nextNode
    print("")

# PRUEBAS de las funciones:

# quickSort
L = listaAleatoria(10,-99,99)
print ("-> Ordenando con quickSort la lista: ", end='')
imprimelista(L)
print("")
quickSort(L)
print("\nLista ordenada: ", end='')
imprimelista(L)

# mergeSort
```

```
L = listaAleatoria(10,-99,99)
print ("\n\n-> Ordenando con mergeSort esta lista: ", end='')
imprimelista(L)
print("")
mergeSort(L)
print("\nLista ordenada: ", end='')
imprimelista(L)
```

```
correo@bunta:~/UNC/AyED1/TP07/Ordenamiento_Avanzado% python3.9 ejercicio1.py
-> Ordenando con quickSort la lista:    -20 91 21 -69 -58 -31 71 -79 -15 -39

 -> Particionando sublista 0 - 9
    -> Intercambiando 1 por 9 queda:   -20 -39 21 -69 -58 -31 71 -79 -15 91
    -> Intercambiando 2 por 7 queda:   -20 -39 -79 -69 -58 -31 71 21 -15 91
    -> Intercambiando 0 por 5 queda:   -31 -39 -79 -69 -58 -20 71 21 -15 91
 -> Particionando sublista 0 - 4
    -> Intercambiando 0 por 4 queda:   -58 -39 -79 -69 -31 -20 71 21 -15 91
 -> Particionando sublista 0 - 3
    -> Intercambiando 1 por 3 queda:   -58 -69 -79 -39 -31 -20 71 21 -15 91
    -> Intercambiando 0 por 2 queda:   -79 -69 -58 -39 -31 -20 71 21 -15 91
 -> Particionando sublista 0 - 1
    -> Intercambiando 0 por 0 queda:   -79 -69 -58 -39 -31 -20 71 21 -15 91
 -> Particionando sublista 6 - 9
    -> Intercambiando 6 por 8 queda:   -79 -69 -58 -39 -31 -20 -15 21 71 91
 -> Particionando sublista 6 - 7
    -> Intercambiando 6 por 6 queda:   -79 -69 -58 -39 -31 -20 -15 21 71 91

Lista ordenada:    -79 -69 -58 -39 -31 -20 -15 21 71 91


-> Ordenando con mergeSort esta lista:    -35 31 55 22 86 -14 17 -27 7 42

 -> Uniendo: ( inicio = 0 , mitad = 0 , fin = 1 )   -35 31 55 22 86 -14 17 -27 7 42
 -> Uniendo: ( inicio = 0 , mitad = 1 , fin = 2 )   -35 31 55 22 86 -14 17 -27 7 42
 -> Uniendo: ( inicio = 3 , mitad = 3 , fin = 4 )   -35 31 55 22 86 -14 17 -27 7 42
 -> Uniendo: ( inicio = 0 , mitad = 2 , fin = 4 )   -35 31 55 22 86 -14 17 -27 7 42
 -> Uniendo: ( inicio = 5 , mitad = 5 , fin = 6 )   -35 22 31 55 86 -14 17 -27 7 42
 -> Uniendo: ( inicio = 5 , mitad = 6 , fin = 7 )   -35 22 31 55 86 -14 17 -27 7 42
 -> Uniendo: ( inicio = 8 , mitad = 8 , fin = 9 )   -35 22 31 55 86 -27 -14 17 7 42
 -> Uniendo: ( inicio = 5 , mitad = 7 , fin = 9 )   -35 22 31 55 86 -27 -14 17 7 42
 -> Uniendo: ( inicio = 0 , mitad = 4 , fin = 9 )   -35 22 31 55 86 -27 -14 7 17 42

Lista ordenada:    -35 -27 -14 7 17 22 31 42 55 86
correo@bunta:~/UNC/AyED1/TP07/Ordenamiento_Avanzado%
correo@bunta:~/UNC/AyED1/TP07/Ordenamiento_Avanzado%
correo@bunta:~/UNC/AyED1/TP07/Ordenamiento_Avanzado%
correo@bunta:~/UNC/AyED1/TP07/Ordenamiento_Avanzado%
correo@bunta:~/UNC/AyED1/TP07/Ordenamiento_Avanzado%
correo@bunta:~/UNC/AyED1/TP07/Ordenamiento_Avanzado%
correo@bunta:~/UNC/AyED1/TP07/Ordenamiento_Avanzado%
```

31/5/25

T.P. ORDENAMIENTO AVANZADO - EJERCICIO 2

Ecuación de recurrencia de MergeSort:

$$T(n) = \begin{cases} C_1 & / n = 1 \\ 2T(n/2) + C_2 n & / n > 1 \end{cases} \quad // \quad T(n) = 2T\left(\frac{n}{2}\right) + n \cdot C_2 \quad (1)$$

$$n = n/2 \Rightarrow T\left(\frac{n}{2}\right) = 2T\left(\frac{n}{4}\right) + \frac{n}{2}C_2 \quad (2)$$

Reemplazando (2) en (1):

$$T(n) = 4T\left(\frac{n}{4}\right) + 2nC_2$$

Si continuamos expandiendo la serie:

$$T(n) = 8T(n/8) + 3nC_2$$

$$\ldots = \ldots$$

$$= 2^k T\left(\frac{n}{2^k}\right) + k \cdot n \cdot C_2 \quad (3)$$

Obtenemos el caso base: $T(1)$

$$\frac{n}{2^k} = 1 \Rightarrow n = 2^k \Rightarrow k = \log_2(n) \quad (4)$$

Reemplazando (4) en (3)

$$T(n) = 2^k T\left(\frac{n}{2^{\log_2 n}}\right) + n \log_2(n) \cdot C_2$$

Reemplazando el primer término por (4):

$$T(n) = 2^k T(1) + n \log_2(n) \cdot C_2$$

$$T(n) = n \cdot C_1 + n \cdot \log_2(n) \cdot C_2$$

$$\therefore \boxed{T(n) \in O(n \cdot \log(n))}$$

```
# TP Ordenamiento Avanzado - Ejercicio 3

import linkedlist
import myqueue

from random import randint

class Bheap:
    """ Estructura Bheap, lo unico que tiene una referencia a una lista.
    """
    bheaplist=linkedlist.LinkedList()
    def __str__(self):
            """ Permite hacer un print a una estructura Bheap
            """
            str_list=""
            current=self.bheaplist.head.nextNode
            while current!=None:
                str_list= str_list+str(current.value)+" "
                current=current.nextNode
            return(str_list)

def heapSort(L):
    if L.head != None:
        BH = Bheap()
        heapify(BH, L)
        L.head = None
        heapSortR(BH, L, linkedlist.length(BH.bheaplist))

def heapSortR(BH, L, size):
    k = delMax(BH)
    linkedlist.add(L, k)
    shiftDown(BH, 1)
    if size > 2:
        heapSortR(BH, L, size-1)

def delMax(BH):
    """ Recupera el mayor elemento del heap. Este siempre se encontrara al comienzo
(posicion 1).
        Para manter la esctrucura del arbol binario se reemplaza el nodo raiz por el
 ultimo nodo.

        Luego mediante la funcion shiftDown se va recorriendo el arbol hasta encontr
ar la posicion
        correcta de dicho nodo. De esta manara se garantiza la propiedad heap.
    """

    if linkedlist.length(BH.bheaplist)>1:
        retval = linkedlist.access(BH.bheaplist, 1)
        value = myqueue.dequeue(BH.bheaplist)
        linkedlist.update(BH.bheaplist, value, 1)
        shiftDown(BH, 1)
        return retval


def shiftUp(BH,i):
    """ Recorre el arbol desde los nodos hacia la raiz y va reemplazando el nodo i p
or su padre
        siempre y cuando i sea mayor. La operacion matematica i // 2 nos permite rap
idamente encontrar al padre.
    """
    if i !=1:
        node = i
        while i // 2 > 0:
            k = i // 2
            p = linkedlist.access(BH.bheaplist, k)
            h = linkedlist.access(BH.bheaplist, i)
            if p < h:
                linkedlist.update(BH.bheaplist, h, k)
                linkedlist.update(BH.bheaplist, p, i)
            i = i // 2
        shiftUp(BH, node-1)

def shiftDown(BH,i,currentsize=None):
```

```
    """ Recorre el arbol desde la raiz y  hacia los nodos (arriba hacia abajo) va re
emplazando el nodo i por sus hijos
        siempre y cuando alguno de sus hijos sea mayor.
    """
    if currentsize==None:
        currentsize=linkedlist.length(BH.bheaplist)-1
    while (i * 2) <= currentsize:
        mc = maxChild(BH,i,currentsize)
        k = i*2
        p = linkedlist.access(BH.bheaplist, i)
        h = linkedlist.access(BH.bheaplist, mc)
        if h != None and p < h:
            linkedlist.update(BH.bheaplist, h, i)
            linkedlist.update(BH.bheaplist, p, mc)
        i = mc
        currentsize = currentsize-1
        shiftDown(BH,i,currentsize)

def maxChild(BH,i,currentsize):
    """ Determina dado un nodo i, cual de sus hijos es el mayor y devuelve la posici
on
    """
    if i * 2 + 1 > currentsize:
        return i * 2
    else:
        if linkedlist.access(BH.bheaplist,i*2) > linkedlist.access(BH.bheaplist,i*2+
1):
            return i * 2
        else:
            return i * 2 + 1

def insert(BH,k):
    """ Inserta un elemento en el heap. Si la lista esta vacia, se crea un elemento
0. Este ultimo no se utiliza,
        pero facilita las operaciones matematicas para acceder a los padres e hijos.

    """
    pos=linkedlist.length(BH.bheaplist)
    if pos==0:
        linkedlist.add(H.bheaplist,0)
        pos=pos+1
    linkedlist.insert(BH.bheaplist,k,pos)
    currentsize=linkedlist.length(BH.bheaplist)-1
    shiftUp(BH,currentsize)

def heapify(BH,L):
    """ Dada una lista crea un heap con complejidad temporal O(n)

    """
    i = linkedlist.length(L) // 2
    BH.bheaplist.head = L.head
    linkedlist.add(BH.bheaplist,0)
    while (i > 0):
        shiftDown(BH,i)
        i = i - 1

def length(BH):
    return linkedlist.length(BH.bheaplist)-1

### funciones para PRUEBAS ###

# para generar una lista con valores aleatorios
def listaAleatoria(tamanho,min,max):
    L = linkedlist.LinkedList()
    for i in range(tamanho):
        linkedlist.add(L,randint(min,max))
    return L

# para imprimir una lista por pantalla
def imprimelista(L):
    nodo=L.head
    print("   ", end='')
    for i in range(linkedlist.length(L)):
```

```
        print(nodo.value, " ", end='', sep='')
        nodo = nodo.nextNode
    print("")

################################

### PRUEBAS ###

if __name__ == "__main__":
    H=Bheap()
    insert(H,8)
    insert(H,1)
    insert(H,5)
    insert(H,4)
    print(H)
    minimun=delMax(H)
    print(H)
    print("----")
    L=linkedlist.LinkedList()
    linkedlist.add(L,4)
    linkedlist.add(L,3)
    linkedlist.add(L,2)
    linkedlist.add(L,1)
    linkedlist.add(L,12)
    linkedlist.add(L,255)
    linkedlist.add(L,1000)
    print(L)
    heapify(H,L)
    print(H)


# heapSort
L = listaAleatoria(10,-99,99)
print ("\n-> Ordenando con heapSort la lista: ", end='')
imprimelista(L)
print("")
heapSort(L)
print("\nLista ordenada: ", end='')
imprimelista(L)
```

```
correo@bunta:~/UNC/AyED1/TP07/Ordenamiento_Avanzado% python3.9 ejercicio3
.py
8 4 5 1
5 4 1
----
<linkedlist.LinkedList object at 0x8007a4040>
1000 255 12 1 2 3 4

-> Ordenando con heapSort la lista:    78 27 49 61 -15 -72 -25 45 -81 -99



Lista ordenada:    -99 -81 -72 -25 -15 27 45 49 61 78
correo@bunta:~/UNC/AyED1/TP07/Ordenamiento_Avanzado%
```