1 7 2 4 1 10 9 3

I = -1

J=0 I=0 xchg 1 cu 1

J=1 nimic

J=2 i=1 xchg 7 2 -> 1 2 7 4 1 10 9 3

J=3 nimic

J=4 i=2 xchg 7 1 -> 1 2 1 4 7 10 9 3

J=5 nimic

J=6 nimic

J=7 i=3 xchg 4 3 -> 1 2 1 3 7 10 9 4

Returnez pozitia buna a partitiei (ultimului element)?

Def partition: pune toate numerele mai mici sau egale cu pivotal (ultimul element) in prima parte a sirului, iar pe cele mai mari la finalul sirului (nesortate), dar acum dupa partitia asta, ultimul element e la locul lui.

Optional: key, reverse, cmp?

1 7 2 4 1 10 9 3

Pivot=1

Low=1

High=7

High=7

Low=7

def cautare\_lab\_prob(self, lab\_prob,index):

        """

        Cauta laboratorul cu lab\_prob dat

        :param lab\_prob: lab\_prob

        :type lab\_prob: str

        :return: laboratorul cu lab\_prob dat, None daca nu exista in lista

        :rtype: laborator

        """

        lista = self.\_\_load\_from\_file()

        if index == len(lista):

            return None

        elif lista[index].getlab\_prob() == lab\_prob:

            return lista[index]

        else:

            return self.cautare\_lab\_prob(lab\_prob,index+1)

T(n) =|0 dacă am ajuns la finalul listei și nu am găsit elementul

|lista(index) dacă am găsit elemental căutat

|cautare\_lab\_prob(lab\_prob, index+1) altfel

T(lab\_prob,index) = 1+T(lab\_prob,index+1)

T(lab\_prob,index+1) = 1+ T(lab\_prob,index+2)

.

.

.

T(lab\_prob,index+n+1) = 0

unde n = len(lista)

complexitatea va fi Θ(n)

Θ (caz favorabil)

Θ(1) – iese din funcție la primul if -> listă goală

O (caz defavorabil) – overall complexity

O(n) – elemental căutat este pe ultima poziție din listă -> se parcurg toate cele n elemente ale listei

Ω (caz mediu)

Ω(n) – o medie între Θ(1) și O(n) este Ω (n/2) care aparține de Ω (n)

import os

import sys

sys.path.append(os.getcwd() + "\\")

from domain.entities import \*

def cmp\_grup(x,y):

    if int(x.getgrup()) > int(y.getgrup()):

        return 1

    elif int(x.getgrup()) < int(y.getgrup()):

        return -1

    else:

        return 0

def cmp\_nume(x,y):

    if x.getnume() > y.getnume():

        return 1

    elif x.getnume() < y.getnume():

        return -1

    else:

        return 0

def partitie(array, start, end):

    pivot = array[start]

    low = start + 1

    high = end

    while True:

        while low <= high and array[high] >= pivot:

            high = high - 1

        while low <= high and array[low] <= pivot:

            low = low + 1

        if low <= high:

            array[low], array[high] = array[high], array[low]

        else:

            break

    array[start], array[high] = array[high], array[start]

    return high

def partitie1(array, start, end, cmp):

    pivot = array[start]

    low = start + 1

    high = end

    while True:

        while low <= high and (cmp(array[high], pivot)==1 or cmp(array[high], pivot)==0):

            high = high - 1

        while low <= high and (cmp(array[low], pivot)==-1 or cmp(array[high], pivot)==0):

            low = low + 1

        if low <= high:

            array[low], array[high] = array[high], array[low]

        else:

            break

    array[start], array[high] = array[high], array[start]

    return high

def QuickSort(array, start, end,reverse = False,cmp=False):

    """

    Metoda de soratre folsind QuickSort.

    array: lista de sortat

    start: index de inceput

    end: index de final

    reverse: daca nu e specificat, e fals by default, daca e specificat True atunci lista va fi sortata descrescator

    cmp: daca nu e specificat, e fala by default, daca e specificat, lista e sortata dupa o anumita functie(definita in prealabil pt. utilizator)

        cu return 1 daca first>second, return 0 daca first=sewcond si return -1 daca first<second

    """

    if start >= end:

        return

    if cmp == False:

        if reverse == False:

            p = partitie(array, start, end)

            QuickSort(array, start, p-1)

            QuickSort(array, p+1, end)

        elif reverse == True:

            p = partitie(array, start, end)

            QuickSort(array, start, p-1)

            QuickSort(array, p+1, end)

            array.reverse()

    elif cmp != False:

        if reverse == False:

            p = partitie1(array, start, end,cmp)

            QuickSort(array, start, p-1,cmp = cmp)

            QuickSort(array, p+1, end,cmp = cmp)

        elif reverse == True:

            p = partitie1(array, start, end,cmp)

            QuickSort(array, start, p-1,cmp = cmp)

            QuickSort(array, p+1, end,cmp = cmp)

            array.reverse()

    return array

def test\_QuickSort():

    lista = [1,3,5,2,8,22,3,5,1,0,10,25,44,1,0]

    s1=student('aa','Sna','4')

    s2=student('bb','Bloom','5')

    s3=student('cc','Catalin','1')

    s4=student('dd','Wiana','3')

    s5=student('ee','Eminescu','2')

    lista1 = [s1,s2,s3,s4,s5]

    lista\_nume = [s2,s3,s5,s1,s4]

    lista\_nume\_reverse = [s4,s1,s5,s3,s2]

    lista\_numere = [0,0,1,1,1,2,3,3,5,5,8,10,22,25,44]

    lista\_numere\_reverse = [44, 25, 22, 10, 8, 5, 5, 3, 3, 2, 1, 1, 1, 0, 0]

    assert(QuickSort(lista,0,len(lista)-1) == lista\_numere)

    assert(QuickSort(lista,0,len(lista)-1,reverse=True) == lista\_numere\_reverse)

    assert(QuickSort(lista1,0,len(lista1)-1,cmp = cmp\_nume) == lista\_nume)

    assert(QuickSort(lista1,0,len(lista1)-1,reverse = True,cmp = cmp\_nume) == lista\_nume\_reverse)

test\_QuickSort()