Clique duas vezes (ou pressione "Enter") para editar

EFC1 GCC253 - Complexidade e Projeto de Algoritmos - 5 pontos.

> **Prof.: Douglas H. S. Abreu**

Aluno: Eric Assis Ribeiro,Fernando Mauricio Roque

Matricula 201910880, 201720355

Turma: 10A, 10A

Link do repositório GitHub: https://github.com/eribito/EFC1-GCC253

● O trabalho deve ser feito em grupos de no máximo 4 componentes (Apenas um deve enviar a atividade no Campus Virtual)

● Trabalhos entregues após a data limite não serão aceitos

● Data limite de entrega: 20 de Novembro de 2022 : 23h55m

● Enviar o trabalho para o campus virtual, do seguinte modo: link do repositório GitHub com acesso ao Notebook.

● O trabalho deve ser desenvolvido no modelo Notebook utilizando preferencialmente a linguagem Python

Importações e Variáveis globais

```
import numpy as np
import time
from math import floor

global_1=0

def trocaElementos(vetor, posA, posB):
    vetor[posA], vetor[posB]=vetor[posB] ,vetor[posA]

def gerarArranjosAleatorios(numeros_aleatorios):
  Aleatorio = np.random.randint(0, 1000, (numeros_aleatorios))
  return Aleatorio
```

Funções de Ordenação

> Utilize este espaço para definir as funções de ordenação.

Insertion Sort

```
def insertionSort(A):
  for j in range (1,len(A)):
    chave=A[j]
    i=j-1
    while i>=0 and A[i] > chave:
      A[i+1]= A[i]
      A[i] = chave
      i = i -1
  A[i+1]=chave
  return A
```

Merge Sort

```
def mergeSort(A):
  if len(A)> 1:
      meio = len(A)//2
      metadeEsquerda = A[:meio]
      metadeDireita = A[meio:]

      mergeSort(metadeEsquerda)
      mergeSort(metadeDireita)

      i=0
      j=0
      k=0
      while i < len(metadeEsquerda) and j < len(metadeDireita):
            if metadeEsquerda[i] < metadeDireita[j]:
                A[k]=metadeEsquerda[i]
                i=i+1
            else:
                A[k]=metadeDireita[j]
                j=j+1
            k=k+1

      while i < len(metadeEsquerda):
            A[k]=metadeEsquerda[i]
            i=i+1
            k=k+1

      while j < len(metadeDireita):
            A[k]=metadeDireita[j]
```

```
            j=j+1
            k=k+1
  return A
```

Selection Sort

```
def selectionSort(A):
  n = len(A)
  # Percorre o arranjo A.
  for i in range(n):
    # Encontra o elemento menor em A.
    menor = i
    for j in range(i + 1, n):
      if A[menor] > A[j]:
        menor = j
  # Coloca o elemento menor na posição correta.
    A[i], A[menor] = A[menor],A[i]

  return A
```

Bubble Sort

```
def bubbleSort(A):
  for j in range(len(A)-1,0,-1):
    for i in range(j):
      if A[i]>A[i+1]:
        A[i], A[i+1] = A[i+1],A[i]
  return A
```

Heap Sort

```
def heapSort(vetor, posFim):
    qtdeComparacoes = 0
    ini= time.time()

    i = int(floor(posFim/2))
    while(i >= 0):
        qtdeComparacoes = criaHeap(vetor, i, posFim, qtdeComparacoes)
        i -= 1

    while(posFim > 0):
        qtdeComparacoes = removeRaiz(vetor, posFim, qtdeComparacoes)
        posFim -= 1

    fim = time.time()
    tempo_exec=fim-ini
```

```
    return tempo_exec, qtdeComparacoes;

def criaHeap(vetor, posInicio, posFim, qtdeComparacoes):
    i = posInicio
    while i <= posFim:
        maiorFilho = i * 2 + 1

        #descobre qual dos filhos é o maior
        if maiorFilho <= posFim:
            qtdeComparacoes += 1
            if (maiorFilho + 1 <= posFim) and (vetor[maiorFilho] < vetor[maiorFilh
                maiorFilho += 1

            #filho é maior que o pai
            qtdeComparacoes += 1
            if vetor[maiorFilho] > vetor[i]:
                trocaElementos(vetor, maiorFilho, i)
                i = maiorFilho
            else:
                i = posFim + 1

        #é uma folha
        else:
            i = posFim + 1

    return qtdeComparacoes


def removeRaiz(vetor, posFim, qtdeComparacoes):
    trocaElementos(vetor, 0, posFim)
    criaHeap(vetor, 0, posFim - 1, qtdeComparacoes)
    return qtdeComparacoes
```

Quick Sort com pivo sendo o ultimo elemento do arranjo

> pivo = *A[A-comprimento]*

```
def quickSort(a, ini=0, fim=None):
    if fim is not None:
      fim = fim
    else:
      fim = len(a)

    if ini < fim:
        pp = particao(a, ini, fim)
        quickSort(a, ini, pp)
        quickSort(a, pp + 1, fim)
    return a

def particao(a, ini, fim):
```

```
    pivo = a[fim-1]
    for i in range(ini, fim):
        if a[i] > pivo:
            fim += 1
        else:
            fim += 1
            ini += 1
            a[i], a[ini - 1] = a[ini - 1], a[i]
    return ini - 1
```

Questões

1. Escolha pelo menos 3 arranjos. Ex: ***A[5,...,1000,...,100]*** e mostre o funcionamento dos Algoritmos realizando a ordenação.

Insertion Sort Resposta

```
insertionSort([5,3,6,3999,0,65,99,0.8,-9,10])
```

```
[-9, 0, 0.8, 3, 5, 6, 10, 65, 99, 3999]
```

```
insertionSort([10000,-999,88,2888888888888,11111111111,-8888])
```

```
[-8888, -999, 88, 10000, 11111111111, 2888888888888]
```

```
insertionSort([1,4,3,2,6,77,99,66,55,44,33])
```

```
[1, 2, 3, 4, 6, 33, 44, 55, 66, 77, 99]
```

```
insertionSort([100,-76,56,1,0,33,-9,567,333,111,8,9,6,5,4,33,22,11])
```

```
[-76, -9, 0, 1, 4, 5, 6, 8, 9, 11, 22, 33, 33, 56, 100, 111, 333, 567]
```

```
insertionSort([154,-99,848,2,1,88,5,76,66,55,44,33,22,11,7])
```

```
[-99, 1, 2, 5, 7, 11, 22, 33, 44, 55, 66, 76, 88, 154, 848]
```

Merge Sort Respostas

```
mergeSort([5,3,6,3999,0,65,99,0.8,-9,10])
```

```
[-9, 0, 0.8, 3, 5, 6, 10, 65, 99, 3999]
```

```
mergeSort([10000,-999,88,2888888888888,11111111111,-8888])
```

```
[-8888, -999, 88, 10000, 11111111111, 2888888888888]
```

```
mergeSort([1,4,3,2,6,77,99,66,55,44,33])
```

```
[1, 2, 3, 4, 6, 33, 44, 55, 66, 77, 99]
```

```
mergeSort([100,-76,56,1,0,33,-9,567,333,111,8,9,6,5,4,33,22,11])
```

```
[-76, -9, 0, 1, 4, 5, 6, 8, 9, 11, 22, 33, 33, 56, 100, 111, 333, 567]
```

```
mergeSort([154,-99,848,2,1,88,5,76,66,55,44,33,22,11,7])
```

```
[-99, 1, 2, 5, 7, 11, 22, 33, 44, 55, 66, 76, 88, 154, 848]
```

Selection Sort Resposta

```
selectionSort([5,3,6,3999,0,65,99,0.8,-9,10])
```

```
[-9, 0, 0.8, 3, 5, 6, 10, 65, 99, 3999]
```

```
selectionSort([10000,-999,88,2888888888888,11111111111,-8888])
```

```
[-8888, -999, 88, 10000, 11111111111, 2888888888888]
```

```
selectionSort([100,-76,56,1,0,33,-9,567,333,111,8,9,6,5,4,33,22,11])
```

```
[-76, -9, 0, 1, 4, 5, 6, 8, 9, 11, 22, 33, 33, 56, 100, 111, 333, 567]
```

```
selectionSort([1,4,3,2,6,77,99,66,55,44,33])
```

```
[1, 2, 3, 4, 6, 33, 44, 55, 66, 77, 99]
```

```
selectionSort([154,-99,848,2,1,88,5,76,66,55,44,33,22,11,7])
```

```
[-99, 1, 2, 5, 7, 11, 22, 33, 44, 55, 66, 76, 88, 154, 848]
```

Bubble Sort Resposta

```
bubbleSort([5,3,6,3999,0,65,99,0.8,-9,10])
```

```
[-9, 0, 0.8, 3, 5, 6, 10, 65, 99, 3999]
```

```
bubbleSort([10000,-999,88,2888888888888,11111111111,-8888])
```

```
[-8888, -999, 88, 10000, 11111111111, 2888888888888]
```

```
bubbleSort([1,4,3,2,6,77,99,66,55,44,33])
```

```
[1, 2, 3, 4, 6, 33, 44, 55, 66, 77, 99]
```

```
bubbleSort([100,-76,56,1,0,33,-9,567,333,111,8,9,6,5,4,33,22,11])
```

```
[-76, -9, 0, 1, 4, 5, 6, 8, 9, 11, 22, 33, 33, 56, 100, 111, 333, 567]
```

```
bubbleSort([154,-99,848,2,1,88,5,76,66,55,44,33,22,11,7])
```

```
[-99, 1, 2, 5, 7, 11, 22, 33, 44, 55, 66, 76, 88, 154, 848]
```

Heap Sort Resposta

```
i = 5
while i <= 10:
  vetor = gerarArranjosAleatorios(i)
  print("Vetor criado: ",str(vetor))
  heapSort(vetor, i-1)
  print("Vetor ordenado: ",str(vetor), "\n")
  i += 1
```

```
Vetor criado:  [946 513 368 754 531]
Vetor ordenado:  [368 513 531 754 946]

Vetor criado:  [544 707 383 412 353 157]
Vetor ordenado:  [157 353 383 412 544 707]

Vetor criado:  [547  72 929 154 314 967 547]
Vetor ordenado:  [ 72 154 314 547 547 929 967]

Vetor criado:  [206 682 958 415 319 661 952  55]
Vetor ordenado:  [ 55 206 319 415 661 682 952 958]

Vetor criado:  [988 365 724 492 788 794 369  15 729]
Vetor ordenado:  [ 15 365 369 492 724 729 788 794 988]

Vetor criado:  [642 485 312  47 683 579 284 342 561 823]
Vetor ordenado:  [ 47 284 312 342 485 561 579 642 683 823]
```

Quick Sort com pivo sendo o ultimo elemento do arranjo

```
quickSort([982,281,274,20,38,50 -21, 3,7,10, 35])
```

```
[3, 7, 10, 20, 29, 35, 38, 274, 281, 982]
```

```
quickSort([8,29,76,20,10,87,20,21,7,30,829,200,98])
```

```
[7, 8, 10, 20, 20, 21, 29, 30, 76, 87, 98, 200, 829]
```

```
quickSort([89,20,31,98,10,12,30,15,90,15,2])
```

```
[2, 10, 12, 15, 15, 20, 30, 31, 89, 90, 98]
```

```
quickSort([25,33,98,100,285,98,33,84,75,98,75])
```

```
[25, 33, 33, 75, 75, 84, 98, 98, 98, 100, 285]
```

```
quickSort([238,-98,-18,-2,387,392,17,928,30,298,0])
```

```
[-98, -18, -2, 0, 17, 30, 238, 298, 387, 392, 928]
```

```
print("Boa sorte!!!")
```

```
Boa sorte!!!
```