Métodos de Ordenação

1) O código abaixo corresponde à versão iterativa do **algoritmo de ordenação por seleção**. Modifique o código e apresente sua versão recursiva.

```
def selectionSort(array):
    for i in range(len(array)-1):
        min = i
        for j in range(i+1, len(array)):
            if(array[j] < array[min]):
                min = j
        array[min], array[i] = array[i], array[min] # troca
```

2) O código abaixo é uma implementação recursiva do **algoritmo de ordenação por inserção.** Acompanhe o código, sem usar recursos computacionais, e exiba o que vai ser exibido pela instrução print destacada.

```
def insertionSortRecursive(arr,n):
    if n<=1:
        return

    last = arr[n-1]

    j = n-2

    while (j>=0 and arr[j]>last):
        arr[j+1] = arr[j]
        j = j-1

    arr[j+1]=last
    print('last:', last,'array:',arr)

# Main program to test insertion sort
array = [12,11,13,5,6,8,14]
size = len(array)
insertionSortRecursive(array, size)
```

3) O código mostrado a seguir é uma versão do **algoritmo quicksort** em que o pivot escolhido é sempre o último elemento do array ou sub-array**.** Entenda o funcionamento do código e escreva, de forma abstrata, o princípio de funcionamento da função **particiona.**

```
def particiona(array, inicio, fim):
    pivot = array[fim]    # pivot
    i = ( inicio - 1 )    # indice do menor elemento


    for j in range(inicio , fim):

        # Se o elemento corrente é menor ou igual ao pivo, efetua a troca
        if  array[j] <= pivot:

            # incrementa o indice do menor elemento
            i = i+1
            array[i],array[j] = array[j],array[i]

    array[i+1],array[fim] = array[fim],array[i+1]
    return ( i+1 )

def quickSort(array,inicio,fim):
    if inicio < fim:
        # pi é o indice de particionamento, correspondente ao posicionamento do pivot.
        # Assim, o pivot está na sua devida posição
        ip = particiona(array,inicio,fim)
        print(array)

        # Separadamente, aplica o método de ordenação antes e depois do ponto
        # de particionamento
        quickSort(array, inicio, ip-1)
        quickSort(array, ip+1, fim)

array = [54,28,61,12,7,85,97,2,71,44]
print('Array Inicial:',array)
quickSort(array,0,len(array)-1)
print('Array Ordenado:',array)
```

4) Considere o seguinte array: **[42, 7, 32, 21, 96, 45, 19]**. Demonstre a classificação deste array, em cada ciclo de iteração, aplicando os métodos de ordenação **buble sort, selection sort** e **quicksort.**

5) Considere o seguinte array: **[42, 7, 32, 21, 96, 45, 19]**. Para cada método a seguir, utilizando fontes de dados externas como apoio (livros, google, etc.), descreva seu princípio de funcionamento e demonstre como o array é afetado pela classificação dos métodos.

(a) Shellsort
(b) Heapsort
(c) Mergesort
(d) Timsort (híbrido: mergesort + insertionsort)

6) [Cândido] Considere o seguinte array: **[42, 7, 32, 21, 96, 45, 19]**. Implemente um programa em python que exiba uma tabela comparativa dos métodos de classificação de dados a seguir:

- Bolha
- Seleção Direita
- Inserção Direta
- Quicksort
- Shellsort
- Heapsort
- Mergesort
- Timsort

Para cada método, o programa deve determinar

Vetor desordenado
(a) Número médio de comparações média;
(b) Número médio de trocas
(c) Média de unidades gastas para classificação. Considere que uma comparação corresponde a 1 unidade, e uma troca corresponde a 3 unidades.

Vetor ordenado (melhor caso)
(d) Número de comparações
(e) Número de trocas
(f) Tempo gasto para classificação (em unidades)

Vetor ordenado inversamente (pior caso)
(g) Número de comparações
(h) Número de trocas
(i) Tempo gasto para classificação (em unidades)