Universidade Federal de Uberlândia

Faculdade de Engenharia Elétrica
Programação Script
Prof. Felipe A. Louza

Lista 13

Busca sequencial; Ordenação;

Questão 1

Analise a função de busca sequencial abaixo.

```
def busca_sequencial(seq, x):
  for i in range(len(seq)):
    if seq[i] == x:
      return True
  return False
```

O que precisa ser mudado na função `busca_sequencial()` para que ela retorne a posição onde o elemento `x` se encontra dentro da lista `seq` ou -1 caso ele não exista.

Questão 2

Altere a função `busca_sequencial()` do exercício anterior para retornar todas as ocorrências de `x` em `seq` (se existir), não apenas a primeira.

Questão 3

Escreva um programa que dada uma sequência com N **números reais** imprime a sequência eliminando os elementos repetidos. Esse exercício deve ser dividido em 2 partes:

```
def acha(seq, x):
    ''' (list, float) -> int
        retorna a posicao em que x ocorre na lista, ou -1 caso contrario
        '''
    # escreva a funcao
```

```
def main():
    ''' programa que le uma sequencia com N elementos e a imprime
        sem repeticoes.
        '''
    # escreva o programa
main()
```

Altere a sua solução para imprimir apenas os números que **não são repetidos**.

Questão 4

Escreva a função `crescente(lista)`, que recebe uma lista com números inteiros como parâmetro e devolve o booleano `True` se a lista estiver ordenada e `False` se a lista não estiver ordenada.

```
def main():
    seq = [2, 4, 5, 8, 9, 11, 17, 21]
    print(seq)

    if crescente(seq):
      print("eh crescente")
    else:
      print("nao eh crescente")
```

Questão 5

Analise o algoritmo de ordenação por **Seleção Direta** (*Selection Sort*) implementado abaixo e considere a lista números abaixo.

```
def selection_sort(lista):
    tam = len(lista)
    for i in range(tam-1):
        menor_pos = i
        for j in range(i+1, tam):
            if(lista[j]<lista[menor_pos]):
                menor_pos = j
        #swap
        lista[i], lista[menor_pos] = lista[menor_pos], lista[i]
    return lista
```

```
numeros = [5,3,0,90,-43,10,7,2,11]
```

Quantas vezes o valor 2 será trocado de lugar, dentro da lista `numeros`, até ser colocado na posição certa da lista ordenada?

Questão 6

Assinale as alternativas **incorretas** sobre o algoritmo de ordenação por **Seleção Direta**.

(a) Este algoritmo ordena **apenas** dados numéricos existentes dentro de uma lista.

(b) Após a primeira iteração, o menor elemento estará na primeira posição da lista, após a segunda iteração, teremos o segundo menor elemento na segunda posição da lista **e assim sucessivamente** até que a lista esteja totalmente ordenada.

(c) No algoritmo de ordenação por **Seleção Direta**, **cada vez** que um menor elemento é encontrado, ele é reposicionado para a posição em que se está ordenando.

(d) Este algoritmo ordena a lista de trás para frente.

Questão 7

Escreva a função `lista_grande(n)`, que recebe como parâmetro um número inteiro n e devolve uma lista contendo n números inteiros aleatórios no intervalo $[0, 1000]$.

```
>>> lista_grande(2)
[81, 567]
>>> lista_grande(8)
[379, 481, 768, 966, 622, 413, 70, 720]
>>> lista_grande(16)
[37, 655, 704, 34, 162, 86, 46, 892, 9, 60, 568, 2, 23, 973, 289, 42]
```

Dica: utilize o módulo `random` do **Python**: `www.w3schools.com/python/module_random.asp`

Questão 8

Implemente o algoritmo de ordenação da **Bolha** (*Bubble Sort*) de acordo com o seguinte **pseudo-código**:

- Repita n **vezes**, isto é, $i = 1, 2, ..., n$:
  - varra a lista com `pos` variando de $N - 1$ **até** i:
    - **Se** `seq[pos]` for menor que o elemento anterior então:
      - **troca** o elemento em `seq[pos]` com o seu anterior

```
def main():
    seq1 = [54, 2, 11, 4, 17, 7, 21, 1]

    print(seq1)
    bolha(seq1)
    print(seq1)

    if not crescente(seq1):
        print("nao ", end='')
    print("eh crescente")

def bubble_sort( seq ):
    ''' (list) -> list
        ordena a lista seq usando o algoritmo de bolha
        '''

    # escreva a funcao

main()
```

Questão 9

Quando o algoritmo de ordenação *Bubble Sort* não realiza nenhuma alteração em uma lista durante uma iteração completa (uma passagem por toda a lista), isto significa que **a lista já está ordenada**.

(a) Implemente a **versão melhorada** do *Bubble Sort* que considera esse fato.

```
bubble_sort_melhorado([3, 2, 5, 6, 2, 1])
# deve devolver [1, 2, 2, 3, 5, 6]
```

(b) Além de devolver uma lista ordenada, ao longo do processamento sua função **deve imprimir** o estado atual da lista toda vez que fizer uma alteração em seus elementos.

```
bubble_sort_melhorado([3, 2, 5, 6, 2, 1])
[3, 2, 5, 6, 1, 2]
[3, 2, 5, 1, 6, 2]
[3, 2, 1, 5, 6, 2]
[3, 1, 2, 5, 6, 2]
[1, 3, 2, 5, 6, 2]
[1, 3, 2, 5, 2, 6]
[1, 3, 2, 2, 5, 6]
[1, 2, 3, 2, 5, 6]
[1, 2, 2, 3, 5, 6]
# deve devolver [1, 2, 2, 3, 5, 6]
```

Questão 10

O algoritmo de **ordenação por Inserção** (*Insertion Sort*) é muito comum, por exemplo, para ordenar um baralho de cartas. A ideia é a seguinte:

1. Assuma que a primeira posição da lista `seq` (1a carta do baralho) está ordenada;
2. novo = 2
3. sabemos que **todos os elementos anteriores** (`seq[0:novo-1]`) estão ordenados;
4. insira o elemento novo (`seq[novo]`) na posição adequada na sub lista `seq[0:novo]`, e desloque os demais elementos, para obter a lista ordenada **até o elemento novo**.
5. incremente novo e repita a partir do **passo 3** até o último elemento da lista

Escreva a função `insertion_sort()` que implementa esse algoritmo.

Uma característica importante para a **eficiência de um algoritmo** de ordenação é a capacidade de **utilizar a mesma lista de entrada**, rearranjando os elementos ao invés de criar uma nova lista ordenada.

Faça isso na sua função.